Induction of Classification Rules and Decision Trees using Genetic Algorithms

NG Sai-Cheong

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

Supervised by

Prof. LEUNG Kwong Sak

December 2004

利用遺傳算法建構分類法則及決策樹

伍世昌

論文摘要

數據發掘是指從大量數據中尋找隱藏的知識。監督式分類是多種數據發掘算法之一。我們可利用監督式分類建構一組模型作爲知識表示方法，以預測未見的資料項之分類。本論文會討論其中兩個監督式分類算法，包括分類法則及決策樹。不過，分類法則及決策樹之搜索空間可能十分龐大。我們會運用遺傳算法以便建構這些模型。

SCION是一個以分類法則爲基礎之專家系統，利用遺傳算法推論一組分類法則，而每個法則爲一組串聯的一元不等式。我們以SCION爲藍本，提出一個以遺傳算法爲基礎的推論分類法則之算法，稱爲GA-based CPRLS，而每個法則之前提爲一組串聯的邏輯算式。這些邏輯算式包括由連續屬性組成的一次不等式和由一個離散屬性及其數值組成之等式。GA-based CPRLS提供一種適用於離散及連續屬性之建構分類法則的算法。

BTGA[9]是一個以遺傳算法爲基礎的線性決策樹之算法。我們以BTGA爲藍本，提出一個以遺傳算法爲基礎的二次決策樹，稱爲GA-based QDT。它在每一個非終結節點運用遺傳算法找尋最佳的二次決策函數。線性決策樹只不過是二次決策樹之一種。實驗結果證明GA-based QDT較其他單變數及線性決策樹更能近似非線性的分類邊界。

當我們利用較大的資料集建構線性或二次決策樹時，可於每個非終結節點決定最佳決策函數前建構一棵k-d樹或廣義四分樹。爲了利用k-D樹或廣義四分樹建構線性或二次決策樹，我們需計算一個合適的線性或二次函數在某個超矩形內之極端值。

雖然我們可準確地求出一個線性函數在某個超矩形內之極端值，但難以準確地計算某二次函數在某個超矩形內之極端值。因此我們介紹了三種估計一個二次函數在某超矩形內之極端值的方法。這三種方法均不會影響建構出的二次決策樹之質素。

我們分別選擇BTGA及GA-based QDT來測試利用k-d樹及廣義四分樹建構線性及二次決策樹時之表現。因此我們修改BTGA並提出BTGA with k-D Trees及BTGA with Quadtrees。同樣地我們以GA-based QDT爲基礎，提出GA-based QDT with k-D Trees及GA-based QDT with Quadtrees。除此之外，我們亦評估將剛提出的三種估計一個二次函數在超矩形內之極端值的方法應用於GA-based QDT with k-D Trees及GA-based QDT with Quadtrees時之表現。實驗結果證明當我們遇上較大的資料集時，我們可利用k-d樹或廣義四分樹以加快建構線性或二次決策樹之速度，而不會影響建構出的決策樹之質素。

Abstract of thesis entitled:

Induction of Classification Rules and Decision Trees using Genetic Algorithms

Data mining is the process of discovering hidden knowledge from large amounts of data. Supervised classification is a kind of data mining algorithms. In supervised classification, a set of models as knowledge representation is constructed to predict the class label of an unseen data. In the thesis, supervised classification techniques including classification rules and decision trees are presented. However, the search space of a classification rule or a decision tree can be very large. Genetic algorithms (GAs) are applied to facilitate the induction of these models.

In a rule-based expert system called SCION, a set of classification rules is evolved using GAs. The antecedent part of a classification rule is a conjunctive set of inequalities with one numeric attribute only. A genetic algorithm-based rule induction algorithm, called GA-based CPRLS, is proposed by extending SCION. In the GA-based CPRLS, the antecedent part of a classification rule is a conjunction of linear inequalities with several numeric attributes and nominal attribute-value pairs. The GA-based CPRLS provides an alternative algorithm to construct a set of classification rules for both numeric and nominal attributes.

In BTGA [9], the optimal linear decision function is found at each node of a linear decision tree. A genetic-algorithm based quadratic decision tree algorithm, called GA-based QDT, is proposed by extending BTGA. At each non-leaf node of a quadratic decision tree, GAs are applied to search for the op-

timal quadratic decision function. Experiments show that the GA-based QDT provides a better approximation to non-linear class boundaries when compared with univariate and linear decision tree algorithms.

Spatial data structures including k-D trees and generalized quadtrees are applied to speed up the construction of oblique and quadratic decision trees provided that the size of a dataset is sufficiently large. In order to construct an oblique or a quadratic decision tree using k-D trees or generalized quadtrees, it is necessary to determine the extreme values of a linear or a quadratic function within a hyperrectangle.

It is straight forward to determine the extreme values of a linear function within a hyperrectangle correctly. Nevertheless, it is difficult to calculate the extreme values of a quadratic function within a hyperrectangle correctly. Three methods of estimating the extreme values of a quadratic function within a hyperrectangle are introduced. These methods ensure that the classification accuracy of a constructed quadratic decision tree is preserved.

BTGA and GA-based QDT are chosen to evaluate the performance when an oblique and a quadratic decision tree are respectively constructed using k-D trees and generalized quadtrees. Two oblique decision tree algorithms, called BTGA with k-D Trees and BTGA with Quadtrees, are extended from BTGA. Moreover, two quadratic decision tree algorithms, called GA-based QDT with k-D Trees and GA-based QDT with Quadtrees, are introduced by extending the GA-based QDT. The performance of the variants of the GA-based QDT is evaluated using the three methods of estimating the extreme values of a quadratic function within a hyperrectangle. Experiments show that the construction of oblique and quadratic decision trees can be accelerated with the aid of k-D trees or generalized quadtrees provided that the size of a dataset is sufficiently large, without sacrificing the quality of a constructed decision tree.

Acknowledgement

I would like to take this opportunity to acknowledge the help and encouragement from my supervisor, markers, friends, colleagues, as well as my family.

First I would like to express my sincere gratitude and appreciation to my thesis supervisor, Prof. Kwong-Sak Leung. He constantly provides guidance, advice and encouragement to my research. It is impossible to complete the thesis without his open-mindedness.

I would also like to express my appreciation to my markers, Prof. Hanqiu Sun and Prof. Tien Tsin Wong, for providing valuable comments and suggestions on my research in the past years.

I am deeply grateful to my friends and colleagues for their assistance and encouragement. The discussion with Jin Huidong and Liang Yong gives me a lot of new insights to my research. I would also like to acknowledege the help of Shum Wing Ho for providing valuable suggestions on the presentation slides for my oral examination.

Finally, I am indebted to my parents and my younger sister, for giving me a family with full of love, support and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Data Mining

Data mining is the process of extracting or mining hidden knowledge from large amounts of data. It is becoming more popular in academic organizations and large corporations. With the rapid development of computer hardware, it is more feasible for academic organizations and companies to collect and maintain large volumes of data. Moreover, the popularity of the World Wide Web (WWW) enables us to access lots of data and information. However, it is impractical for human brains to search for complex relationships in tremendous amount of data. On the other hand, it is more economical to apply automated data mining systems instead of employing a team of highly trained professionals to perform analysis on large volumes of data. Although data mining systems cannot entirely solve complex problems without humans, it simplifies the process of extracting knowledge from data. The information discovered from data mining systems can be applied to business decision making, marketing analysis, transactional analysis and so on.

Data mining is regarded as an essential step in the process of Knowledge Discovery in Databases (KDD). Some people may treat data mining as a synonym for KDD. KDD is composed of an iterative sequence of the following steps [21]:

- Data cleansing where inconsistent data are removed,
- Data integration where multiple data sources are combined if necessary,
- Data selection where data relevant to the analysis are retrieved from the database,
- Data transformation where data are transformed into forms which are suitable for data mining,
- Data mining which is an essential process to extract hidden patterns and knowledge, and
- Data interpretation which identifies the truly interesting patterns representing knowledge based on some measures of interestingness.

Data mining tasks can be divided into several categories, including supervised classification, unsupervised classification, association analysis, data characterization and so on. Supervised classification searches for a set of models in order to predict the class label of an unseen data item. The possible models include classification rules, decision trees, mathematical formulae and so on. The proposed algorithms in the thesis are examples of supervised classification algorithms. Unsupervised classification is the process of analyzing a set of data items without class labels. The data items are grouped so as to maximize the intra-class similarity and minimize the interclass similarity. The data items within the same group should be similar to each other. But a pair of data items from distinct groups should have low similarity. Association analysis discovers a set of association rules describing the attribute-value conditions which occur frequently together in a dataset [21]. Data characterization is the process of summarizing the general characteristics of a specified class of data [21].

1.2 Problem Specifications and Motivations

A rule-based expert system for continuous input attributes called SCION was proposed by Leung et al. [33], [34]. In this system, a set of classification rules is evolved using Genetic Algorithms (GAs). The antecedent part of a classification rule is a conjunctive set of inequalities involving one numeric attribute, which is equivalent to a hyperrectangle in the attribute space. On most datasets, the class boundaries are not axis-parallel and there are one or more nominal input attributes. Therefore, it is necessary to propose a rule learning algorithm such that the antecedent part of a rule may consist of linear inequalities involving one or more numeric attributes and nominal attribute-value pairs. GAs are applied to evolve a set of such classification rules because its search space can be very large.

Murthy et al. proposed OC1 algorithm, using a combination of hill-climbing and randomization to find the optimal hyperplane at each internal node of an oblique decision tree [39], [38], which uses one or more linear functions to perform classifications. Ittner et al. introduced Non-linear Decision Trees (NDT) by extending the OC1 algorithm [26]. The decision function at each non-leaf node is equivalent to a quadratic hypersurface [26]. A quadratic hypersurface provides a better approximation to a non-linear class boundary than a straight line. At each non-leaf node of an NDT, the optimal quadratic hypersurface is determined using a combination of hill-climbing and randomization. On the other hand, Chai et al. proposed Binary-Tree Genetic Algorithm (BTGA), where the optimal hyperplane is evolved using GAs at each non-terminal node [9]. Experiments show that BTGA outperforms the OC1 algorithm in most cases. Therefore, a novel GA-based quadratic decision tree algorithm is proposed by extending BTGA.

An oblique or a quadratic decision tree is usually constructed

using a top-down approach. At each non-leaf node of an oblique or a quadratic decision tree, the optimal linear or quadratic decision function is found using a heuristic. The optimality of a decision function is usually defined as the impurity reduction after partitioning a set of training samples into two disjoint subsets. Several impurity measures such as the Gini-index [6] and the Twoing value [6] can be used to evaluate the impurity of a set of training samples.

To determine the impurity reduction, it is necessary to find the number of training samples for each class satisfying the linear or the quadratic decision function to be considered. This task can be performed by evaluating the sign for each of these training samples. However, this approach is time consuming on large datasets.

Alternatively, spatial data structures such as k-D trees and generalized quadtrees may be constructed before searching for the optimal linear or quadratic decision function at each non-leaf node of an oblique or a quadratic decision tree. At each node of a k-D tree or a generalized quadtree, there is the corresponding smallest hyperrectangle containing the set of training samples arriving at that node. When a linear or a quadratic decision function does not intersect the smallest hyperrectangle containing the set of training samples arriving at a node of a k-D tree or a generalized quadtree, all of these training samples either satisfy or violate the decision function. In this case, it is unnecessary to decide whether the decision function is satisfied for each of these training samples. Otherwise, the descendants of the node are considered if necessary until a leaf node is reached.

It is straight forward to decide whether a linear decision function intersects a hyperrectangle correctly using a suitable linear function. However, it is difficult to determine whether a quadratic decision function intersects a hyperrectangle correctly. Therefore, it is necessary to choose the suitable methods to es-

timate the minimum and the maximum values of a quadratic function within a hyperrectangle so that the quality of a constructed quadratic decision tree is preserved using k-D trees or generalized quadtrees.

1.3 Contributions of the Thesis

The contributions of the thesis are summarized as follows:

- A novel rule induction algorithm is proposed. A set of classification rules is evolved using a genetic algorithm, where the antecedent part of a rule is a conjunction of logical expressions. The possible logical expressions include nominal attribute-value pairs and linear inequalities with one or more continuous attributes.
- A novel quadratic decision tree algorithm is introduced, where genetic algorithms are applied to search for the optimal quadratic decision function at each non-leaf node of a quadratic decision tree.
- k-D trees and generalized quadtrees are proposed and applied to speed up the induction of an oblique and a quadratic decision tree on a sufficiently large dataset, without sacrificing the quality of a constructed decision tree.
  - The optimality of a linear or a quadratic decision function is usually defined as the impurity reduction after dividing a set of training samples into two disjoint subsets. It is necessary to calculate the number of training samples for each class satisfying the linear or the quadratic decision function to be considered. An alternative method to perform this task is introduced using a constructed k-D tree or generalized quadtree.

- Three methods of estimating the minimum and the maximum values of a quadratic function within a hyperrectangle, which are useful for evaluating the impurity reduction due to a quadratic decision function, are introduced. Although all of these methods may overestimate its maximum value and underestimate its minimum value, neither its maximum value is underestimated nor its minimum value is overestimated. Therefore, the impurity reduction due to a quadratic decision function can be evaluated accurately.

1.4 Thesis Roadmap

In this chapter, the background of data mining is presented. The motivation and the contributions of the thesis are also described.

In chapter 2, supervised classification algorithms including decision trees and rule induction are introduced. Moreover, the four kinds of evolutionary algorithms (EAs), including genetic algorithms (GAs), genetic programming (GP), evolution strategies (ES) and evolutionary programming (EP), are presented. Several EA-based rule induction and decision tree algorithms are described. In addition, spatial data structures such as k-D trees and generalized quadtrees, as well as their applications, are given.

In chapter 3, a novel rule induction algorithm, called Genetic Algorithm-based Convex Polytope Rule Learning System (GA-based CPRLS), is introduced by extending SCION [33], [34]. A set of classification rules is evolved using GAs. The antecedent part of each classification rule is a conjunction of logical expressions, including linear inequalities and nominal attribute-value pairs. Token competition [33], [34] is employed to remove redundant rules. The performance of the GA-based CPRLS is evaluated and compared with that of several supervised classi-

fication algorithms. In addition, the quality of the classification rules evolved by the GA-based CPRLS is compared with that generated by random search. Moreover, the effects of token competition and rule migration on the performance of the GA-based CPRLS are investigated.

In chapter 4, a novel decision tree algorithm, called Genetic Algorithm-based Quadratic Decision Tree (GA-based QDT), is proposed by extending Binary Tree-Genetic Algorithm (BTGA) [9]. At each non-terminal node of a GA-based QDT, the optimal quadratic decision function is evolved using GAs. The performance of the GA-based QDT is evaluated and compared with that of various supervised classification algorithms. Moreover, the performance of the GA-based QDT is compared with that of Random Search-based Quadratic Decision Tree (RS-based QDT). Moreover, the effect of changing the parameters of the GA-based QDT is investigated. The effect of noise on the GA-based QDT is also studied.

In chapter 5, two spatial data structures including k-D trees and generalized quadtrees are employed to speed up the construction of oblique and quadratic decision trees provided that the size of a dataset is sufficiently large. The structures of a k-D tree and a generalized quadtree for the construction of a linear or a quadratic decision tree are described first. The algorithm to construct the k-D trees and the generalized quadtrees are then presented. After that, the algorithm to evaluate the optimality of a linear decision function with the aid of k-D trees and generalized quadtrees is described. The algorithm to calculate the optimality of a quadratic decision function using k-D trees and generalized quadtrees is presented. Furthermore, three methods of estimating the maximum and the minimum values of a quadratic function within a hyperrectangle are introduced.

Two linear decision tree algorithms, called Binary Tree-Genetic Algorithm with k-D trees (BTGA with k-D Trees) and Binary-

Tree Genetic Algorithm with Quadtrees (BTGA with Quadtrees), are modified from BTGA. Similarly, two quadratic decision tree algorithms, called Genetic Algorithm-based Quadratic Decision Tree with k-D Trees (GA-based QDT with k-D Trees) and Genetic Algorithm-based Quadratic Decision Tree with Quadtrees (GA-based QDT with Quadtrees), are introduced by extending the GA-based QDT. The performance of all of the variants of BTGA and the GA-based QDT, is evaluated and compared with various supervised classification algorithms. The effects on these proposed algorithms are investigated when the number of training samples at each node of a k-D tree or a generalized quadtree is changed and the size of a dataset is increased.

In chapter 6, a conclusion of the thesis is provided, including its contributions. Possible research directions of the thesis are also described.

□ **End of chapter.**

Chapter 2

Related Work

2.1 Supervised Classification Techniques

There are various kinds of supervised classification techniques, including classification rules, decision trees, artificial neural networks, nearest neighbor classifiers, Bayesian classifiers and so on. In this section, classification rules and decision trees will be discussed because my research work is closely ralated to them.

2.1.1 Classification Rules

A rule may be used to express useful knowledge in the form of an "if-then" statement. It consists of two parts, including the antecedent and the consequent parts. The antecedent part of a rule specifies the necessary conditions so that a conclusion, which is stated in the consequent part of the rule, can be drawn.

A classification rule is a rule which can be used to perform supervised classifications. The antecedent part of a rule states the conditions of one or more of the input attributes of a sample. The consequent part of a classification rule determines the class label when the conditions specified in the antecedent part are satisified. In most cases, it is necessary to employ more than one classification rule to perform supervised classifications.

To learn a set of rules, a dataset is partitioned into two sub-

sets, including a training and a testing sets. The training set is applied to induce a set of classification rules. The quality of a set of classification rules is evaluated on the testing set. A good set of classification rules should minimize the number of misclassifications on the testing set, rather than the training set.

There are various algorithms to learn a set of classification rules, including sequential covering algorithms [36], C4.5RULES [44] and so on.

In sequential covering algorithms, a set of positive and negative training samples is considered. A single rule is produced in each iteration. After a single rule is learnt, the positive training samples covered by the rule are removed and the remaining training samples are employed to induce another rule. The process is repeated iteratively until the fraction of the positive training samples covered by the disjunctive set of the generated rules is greater than a threshold. A generated rule should have high classification accuracy, rather than high coverage.

After a disjunctive set of rules is learnt, they can be sorted so that more accurate rules will be considered first when they are applied to classify unseen testing samples. Sequential covering algorithms are greedy algorithms so that the best or the smallest set of rules cannot be guaranteed to be found. CN2 [10] is an example sequential covering algorithm.

C4.5RULES is a component of C4.5 algorithm [44]. C4.5RULES transforms a decision tree constructed by the C4.5 algorithm into a set classification rules. In a decision tree, a path from the root to a leaf node is equivalent to a classification rule. The antecedent part of a rule is the conjunction of the decision functions at the non-leaf nodes in the corresponding path. The consequent part is the class label associated with the leaf node. However, a rule dervied from a decision tree can be very complicated and it may be necessary to simplify the rule by removing one or more conditions.

Some of the simplified rules may be redundant. For each class, a subset of rules is chosen from the set of rules classifying that class using the minimum description length (MDL) [46] principle.

2.1.2 Decision Trees

Decision trees are tree structures which classify an input sample into one of its possible classes. Since the last century, decision trees have been applied successfully in various tasks, including character recognition, remote sensing, medical diagnosis and so on [47].

In a decision tree, there are two or more child nodes at each non-terminal node. There is a decision criterion to select the appropriate child node at each non-terminal node. One or more input attributes are involved in a decision criterion. There is a class label at each terminal node to classify an input sample arrived at it.

In order to classify an input sample using a decision tree, the root node should be considered first. The decision criterion associated with the root node determines which child node (or subtree) should be chosen next. The process is repeated recursively at each selected descendant until a terminal node is encountered. The input sample is classified according to the class label associated with the terminal node. Figure 2.1 shows an example decision tree.

To construct a decision tree, a set of input samples is partitioned into two or three disjoint subsets, called training set, testing set, and validation set if necessary. The training set is used to construct a decision tree. A constructed decision tree should minimize the number of misclassifications on the testing set, instead of the training set. The validation set sometimes can be used for postpruning such as to decide whether to replace a

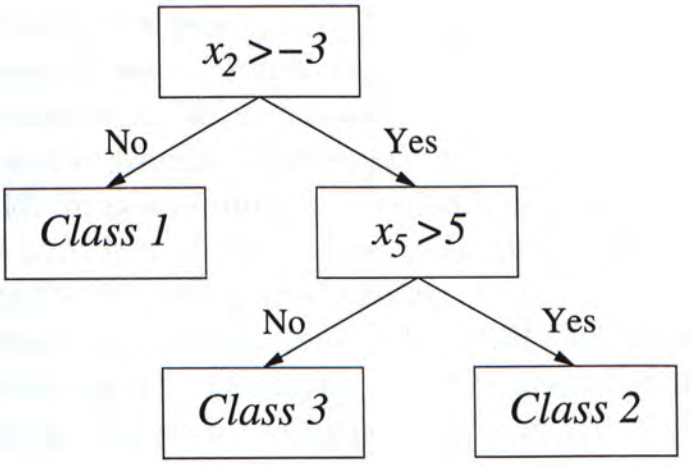

Figure 2.1: An Example Decision Tree

subtree by a node or not.

Usually, a decision tree is constructed using a top-down approach [16]. To build a decision tree, the root node is created first. The optimal decision criterion associated with the root node is determined to partition the training set into two or more disjoint subsets. Several impurity measures, including information gain [43], the Gini index [6], the Twoing rule [6] and so on, can be applied to evaluate the optimality of a decision criterion. After determining the optimal decision criterion at the root node, two or more child nodes are created and the training set is partitioned into two or more disjoint subsets such that each subset is associated with one child node. The process is repeated recursively for each child node until a termination criterion is satisfied. An important issue of constructing a decision tree is overfitting of the training samples. Overfitting occurs when there exists a simplier decision tree such that it has higher classification rate on the testing set. Although a decision tree may classify all the training samples correctly, it may not classify the unseen testing samples well. It is necessary to control the growth of a decision tree. The process of controlling the

growth of a decision tree is called pruning.

There are two major approaches of pruning, including prepruning and postpruning. In prepruning, no child nodes are created if the number of training samples arriving at a node is less than a threshold. In postpruning, a decision tree with no misclassification is built first. Later, a subtree is replaced by a terminal node using the validation set if necessary.

Decision trees whose decision criterion at each non-terminal node depends on only one input attribute are called univariate decision trees. Decision trees whose decision criterion involves more than one input attribute at each non-terminal node are called multivariate decision trees. The following describes several decision tree algorithms which construct univariate or multivariate decision trees.

Univariate Decision Trees

In univariate decision trees, the decision criterion at each non-terminal node considers one input attribute only. If the decision criterion at a non-terminal node considers a continuous attribute, it is equivalent to a hyperplane which is parallel to one of the coordinate axes. Decision tree algorithms, such as Iterative Dichotomiser 3 (ID3) [43], C4.5 [44] and Classification and Regression Trees (CART) [6], can construct univariate decision trees. The decision tree shown in Figure 2.1 is an example univariate decision tree.

Decision trees constructed by the ID3 algorithm are capable of classifying an input sample consists of categorical attributes only. The number of child nodes at each non-terminal node equals the number of possible values of the corresponding attribute associated with the node. No pruning is employed in the ID3 algorithm. To construct a decision tree using the ID3 algorithm, the root node is created first. After that, the information gain of the training set due to each attribute is evaluated.

The attribute which induces the maximum information gain is chosen as the decision function associated with the root node. The training set is partitioned into several subsets using the selected attribute. The process is repeated recursively in each subset until all the training samples arriving at a node belong to one single class or they cannot be further partitioned using any input attribute.

Quinlan introduced the C4.5 algorithm [44] so that it can build decision trees which are capable of handling continuous input attributes. At each non-terminal node whose decision function considers a categorical attribute, the number of child nodes equals the number of possible values of the attribute. There are two child nodes at each non-terminal node whose decision function considers a continuous attribute. The gain ratio is applied to evaluate the optimality of a decision function because the information gain tends to favor attributes which induce more partitions [44].

Breiman et al. introduced the CART algorithm in 1984 [6]. Decision trees produced by the CART algorithm are binary trees, in which each non-terminal node has two children. The Gini index or the Twoing value [6] is used to evaluate the optimality of a decision function. Decision trees constructed by the CART algorithm can handle categorical and continuous input attributes. At each terminal node, the corresponding class label is determined as the class with the maximum number training samples arriving at it. The process is repeated recursively until the impurity reduction is less than a user-defined threshold. The CART algorithm employes minimal cost-complexity pruning [6] to control the growth of a decision tree.

Multivariate Decision Trees

In a multivariate decision tree, the decision function is a linear or non-linear combination of more than one input attribute.

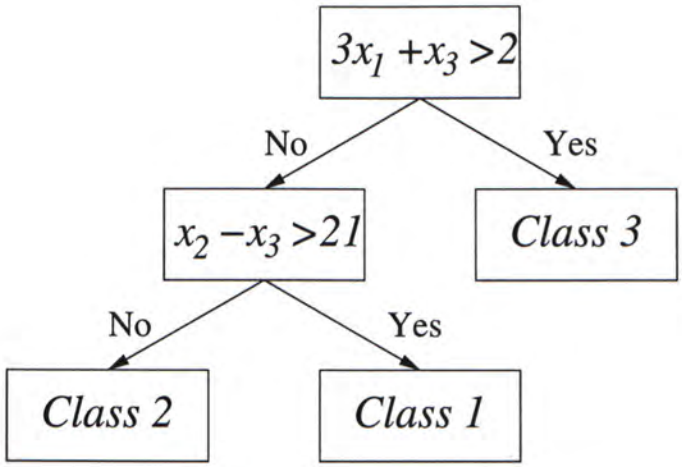

Figure 2.2: An Example Oblique Decision Tree

Most of the multivariate decision tree algorithms construct binary trees. In an oblique decision tree [39], [38], the decision function at each non-terminal node is a linear combination of all input attributes. The decision function at each non-terminal node is equivalent to a hyperplane with an arbitrary orientation in the attribute space [38]. Oblique decision trees are also known as linear decision trees [20] or perceptron decision trees [2]. Figure 2.2 shows an example oblique decision tree.

The problem of finding the optimal hyperplane which minimizes the number of misclassifications after dividing a set of training samples into two disjoint subsets is NP-hard [23]. Oblique decision tree algorithms use a heuristic to attempt to search for the optimal hyperplane at each non-terminal node.

In a non-linear decision tree, the decision function at each non-terminal node is a non-linear combination of all input attributes. A special case of a non-linear decision tree is a quadratic decision tree. In a quadratic decision tree, the decision function at each non-terminal node is equivalent to a quadratic hypersurface in the attribute space. Figure 2.3 shows an example non-linear decision tree.

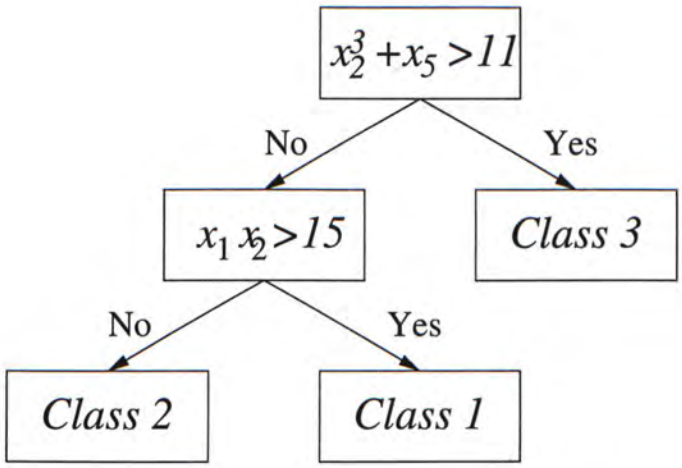

Figure 2.3: An Example Non-linear Decision Tree

The CART algorithm is also capable of constructing a linear decision tree. At each non-terminal node, the associated decision function is allowed to be a linear combination of continuous input attributes. In order to find the optimal linear decision function, the CART algorithm cycles thorugh the continuous input attributes sequentially in each iteration. The cycling continues until the change in the impurity reduction is less than a predefiend threshold. If the number of training samples arriving at a node is less than a user-defined threshold, the decision function at that node considers one of the input attributes only.

Murthy et al. proposed Oblique Classifier 1 (OC1) to construct an oblique decision tree by extending the CART algorithm [38]. To find the optimal hyperplane at each non-terminal node, the OC1 algorithm searches for the best axis-parallel hyperplane first. A hyperplane is said to be axis-parallel if it is parallel to one of the coordinate axes in the attribute space. Then the optimal hyperplane with an arbitrary orientation is determined. To improve the search for the optimal hyperplane, the OC1 algorithm attempts to escape from local optima in the coefficient space using randomization [39], [38]. When a local optimum is

encountered, the coefficients of a linear decision function (including the constant term) are added to a random vector in the coefficient space, in which each point specifies a linear decision function.

Brodley et al. [7] introduced Linear Machine Discriminant Trees (LMDT) to construct multivariate decision trees. At each non-leaf node, there are C child nodes, where C is the number of possible classes. There are C linear discriminant functions at each non-terminal node.

Shah et al. [51] introduced Alopex Perceptron Decision Tree (APDT) algorithm to construct oblique decision trees. At each non-terminal node, linear separability is used to evalaute the optimality of a linear decision function. At each non-terminal node, the initial hyperplane is perpendicular to the vector joining two randomly chosen input samples of distinct classes arriving at it. Alopex algorithm [54] is then applied to find the optimal linear decision function. However, the APDT algorithm is capable of handling two-class problems only.

Gama et al. proposed Ltree to construct oblique decision trees [17]. At each node, a set of linear discriminant functions are employed to create new attributes for each training sample arriving at that node. A new training sample is created using the original attributes and the newly created input attributes for each original training sample. The C4.5 algorithm is used to find the optimal hyperplane using the set of training samples containing the newly created input attributes [17]. After determining the optimal hyperplane, the training samples are partitioned into two disjoint subsets. The process is repeated recurisvely and the newly created input attributes are propagated to the descendants of the currently processed node.

Iyengar introduced a method of constructing oblique decision trees which can be incorporated into most of the existing univariate decision tree algorithms [27]. To construct an

Algorithm	Binary Tree?	Multivariate?	Deterministic?	Can Handle Continuous Attributes?	Can Handle Multiclass Problems?
ID3	No	No	Yes	No	Yes
C4.5	No	No	Yes	Yes	Yes
CART	Yes	Yes	Yes	Yes	Yes
OC1	Yes	Yes	No	Yes	Yes
LMDT	No	Yes	Yes	Yes	Yes
APDT	Yes	Yes	Yes	Yes	No
Ltree	Yes	Yes	Yes	Yes	Yes

Table 2.1: An Comparison of Various Decision Tree Algorithms

oblique decision tree, a univariate decision tree is constructed first. The constructed decision tree is then pruned if necessary. After that, candidate oblique vectors are found using the decision tree. For each training sample, new input attributes are created from these oblique vectors. The process is repeated for a predefined number of iterations.

Yıldız et al. introduced linear discriminant trees [58]. Each non-terminal node employes linear discriminant analysis (LDA) to determine the hyperplane dividing a set of training samples into two disjoint subsets. Before constructing a decision tree, each categorical attribute is transformed into K binary attributes, where K is the number of possible values of the categorical attribute. One of these K binary attributes is set to one and the remaining ones are set to zero, depending on the value of the corresponding categorical attribute.

Table 2.1 provides a comparison of various decision tree algorithms. Note that the column 'Deterministic' shows whether a decision tree algorithm always produces the same decision tree when the same set of samples are applied in the training stage.

2.2 Evolutionary Algorithms

Evolutionary algorithms (EAs) are stochastic optimization algorithms inspired by the principles of natual selection and genetics [12]. There are four kinds of EAs, including genetic algorithms (GAs), genetic programming (GP), evolution strategies (ES) and evolutionary programming (EP) [56].

An evolutionary algorithm maintains a population of chromosomes. Each chromosome represents a candidate solution to a problem. A fitness function is employed to measure the strength of a chromosome, which reflects the quality of the corresponding solution to a problem. Offspring chromosomes are generated using selection, mutation and recombination operators.

The different kinds of evolutionary algorithms differ mainly in the choices of the evolution models, the fitness functions, the evolutionary operators and the selection methods [14].

2.2.1 Genetic Algorithms

Genetic algorithms (GAs) were proposed by Holland [25]. In traditional GAs, each individual is a binary string with a fixed length. Nowadays, many GAs use a real-valued vector to represent an individual in a population. GAs which employ a binary string to represent a chromosome are called binary-coded genetic algorithms (BCGAs). GAs which use a real-valued vector to represent an individual are called real-coded genetic algorithms (RCGAs) [24].

To solve a problem using GAs, a population of chromosomes is initialized first. The fitness value of each chromosome is evaluated. Chromosomes are selected and replicated to the mating pool. Chromosomes with higher fitness values are more likely to be copied to the mating pool. Chromosomes in the mating pool undergo crossover and mutation. The processes of fitness evaluation, selection, crossover and mutation are repeated until

one of the predefined termination criteria is satisifed. An elitist strategy is used to make sure that the best individual in the current generation is preserved in the next generation. The following outlines the steps of a GA:

1. Set $\tau = 0$, where τ is the current generation number.
2. Initialize a population P_τ of L chromosomes.
3. Evaluate the fitness value of each chromosome in the population P_τ.
4. Set θ_{best} as the best chromosome in the population P_τ.
5. While all of the termination criteria are not satisfied,
   (a) Select L chromosomes from the population P_τ. The selected chromosomes are then copied to the mating pool M.
   (b) Set $M' = \text{Crossover}(M)$.
   (c) Set $M'' = \text{Mutation}(M')$.
   (d) Evaluate the fitness value of each chromosome in M''.
   (e) Let θ_{worst} be the worst chromosome in the population M''.
   (f) Set $P_{\tau+1} = (M'' \setminus \{\theta_{worst}\}) \cup \{\theta_{best}\}$.
   (g) Set θ_{best} as the best chromosome in the population $P_{\tau+1}$.
   (h) Set $\tau = \tau + 1$.
6. The best chromosome in the population P_τ is selected as the solution to a problem.

Selection

There are several methods to select a chromosome to be replicated to the mating pool, including roulette wheel selection [11], rank based selection [1], tournament selection [22] and so on. Stronger chromosomes are more likely to be replicated to the mating pool in these selection methods.

In roulette wheel selection, the probability p_i of selecting the i^{th} chromosome in a population P is:

$$p_i = \frac{f_i}{\sum_{j=1}^{d} f_j} \tag{2.1}$$

where f_i, $i = 1, 2..., L$, is the fitness value of the i^{th} chromosome.

In rank based selection, the chromosomes in a population are sorted in the descending order of their fitness values first. The probability of a chromosome selected for replication depends on the rank of its fitness value. A chromosome with a higher ranking is more likely to be copied to the mating pool.

In tournament selection, two or more chromosomes are randomly selected first. Each chromosome in a population is selected with an equal probability. The best chromosome among the selected ones is allowed to be copied to the mating pool.

Crossover

There are several crossover operators in BCGAs, including simple crossover [25], [19], uniform crossover [53] and so on. Given two parent chromosomes $\mathbf{b_1} = (b_{1,1}, b_{1,2}..., b_{1,d})$ and $\mathbf{b_2} = (b_{2,1}, b_{2,2}..., b_{2,d})$ in a BCGA, where d is the length of a chromosome, the following outlines the steps to generate two offspring chromosomes $\mathbf{b'_1} = (b'_{1,1}, b'_{1,2}..., b'_{1,d})$ and $\mathbf{b'_2} = (b'_{2,1}, b'_{2,2}..., b'_{2,d})$ using these crossover operators.

- Simple Crossover

1. Generate a random integer $c \in \{1, 2..., d-1\}$.
2. Set $b'_{1,i} = b_{1,i}$ and $b'_{2,i} = b_{2,i}$ for $i = 1, 2..., c$.
3. Set $b'_{1,i} = b_{2,i}$ and $b'_{2,i} = b_{1,i}$ for $i = c+1, c+2..., d$.

- Uniform Crossover
  FOR $i = 1$ TO d DO

1. Generate a random bit $c \in \{0, 1\}$.
2. Set $b'_{1,i} = b_{1,i}c + b_{2,i}\overline{c}$.
3. Set $b'_{2,i} = b_{1,i}\overline{c} + b_{2,i}c$.

There are various crossover operators in RCGAs, including arithmetical crossover [35], BLX-α crossover [13], linear crossover [57], discrete crossover [37], Wright's heuristic crossover [57] and so on. Note that each crossover operator in RCGAs may generate a different number of offspring. Suppose two parent chromosomes $\theta_1 = (\theta_{1,1}, \theta_{1,2}..., \theta_{1,d})$ and $\theta_2 = (\theta_{2,1}, \theta_{2,2}..., \theta_{2,d})$ are chosen to undergo crossover, the following paragraphs describe the effect when different crossover operators are applied in RCGAs.

In arithmetical crossover, two offspring chromosomes $\theta'_1 = (\theta'_{1,1}, \theta'_{1,2}..., \theta'_{1,d})$ and $\theta'_2 = (\theta'_{2,1}, \theta'_{2,2}..., \theta'_{2,d})$ are generated such that:

$$\theta'_{1,i} = \lambda\theta_{1,i} + (1-\lambda)\theta_{2,i} \tag{2.2}$$

$$\theta'_{2,i} = \lambda\theta_{2,i} + (1-\lambda)\theta_{1,i} \tag{2.3}$$

where λ is a constant and $i = 1, 2..., d$.

In BLX-α crossover, one offspring $\theta' = (\theta'_1, \theta'_2..., \theta'_d)$ is generated such that θ'_i, $i = 1, 2..., d$, is a uniform random number within the ragne $[min(\theta_{1,i}, \theta_{2,i}) - r_i\alpha, max(\theta_{1,i}, \theta_{2,i}) + r_i\alpha]$, where $r_i = max(\theta_{1,i}, \theta_{2,i}) - min(\theta_{1,i}, \theta_{2,i})$.

In linear crossover, three offspring chromosomes $\theta'_1 = (\theta'_{1,1}, \theta'_{1,2}$ $..., \theta'_{1,d})$, $\theta'_2 = (\theta'_{2,1}, \theta'_{2,2}..., \theta'_{2,d})$ and $\theta'_3 = (\theta'_{3,1}, \theta'_{3,2}..., \theta'_{3,d})$ are generated first, where

$$\theta'_{1,i} = 0.5\theta_{1,i} + 0.5\theta_{2,i} \tag{2.4}$$

$$\theta'_{2,i} = 1.5\theta_{1,i} - 0.5\theta_{2,i} \tag{2.5}$$
$$\theta'_{3,i} = 1.5\theta_{2,i} - 0.5\theta_{1,i} \tag{2.6}$$

where $i = 1, 2..., d$. The parent chromosomes are then replaced by the two strongest offspring chromosomes among the three ones.

In discrete crossover, one offspring $\theta' = (\theta'_1, \theta'_2..., \theta'_d)$ is generated such that $\theta'_i = 0.5[\theta_{1,i}(1 + c_i) + \theta_{2,i}(1 - c_i)]$, $i = 1, 2..., d$, and $c_i \in \{-1, 1\}$ is a random integer.

In Wright's heuristic crossover, suppose θ_1 is stronger than θ_2, one offspring $\theta' = (\theta'_1, \theta'_2..., \theta'_d)$ is generated such that $\theta'_i = r(\theta_{1,i} - \theta_{2,i}) + \theta_{1,i}$, where r is a random number within the range $[0, 1]$, $i = 1, 2..., d$.

Mutation

In BCGAs, each bit of an individual is mutated with a fixed or a varying probability. When the value of a selected bit equals zero, its value is changed to one. Otherwise, the value of the selected bit is set to zero.

Common mutation operators in RCGAs include random mutation, non-uniform mutation and so on [35]. Suppose the chormosome $\theta = (\theta_1, \theta_2..., \theta_d)$, where $\theta_i \in [\theta_{min}, \theta_{max}]$, $i = 1, 2..., d$, is selected to undergo mutation, the effect of different mutation operators is described below.

In random mutation, an offspring $\theta' = (\theta'_1, \theta'_2..., \theta'_d)$ is generated such that θ'_i is a uniform random number within the range $[\theta_{min}, \theta_{max}]$.

The following outlines the steps to create an offspring $\theta' = (\theta'_1, \theta'_2..., \theta'_d)$ using non-uniform mutation:

1. Generate a random integer $c \in \{-1, 1\}$.
2. IF $c = 1$, THEN set $\theta'_i = \theta_i + (\theta_{max} - \theta_i)(1 - r^{(1-\frac{\tau}{G})^b})$
   ELSE set $\theta'_i = \theta_i - (\theta_i - \theta_{min})(1 - r^{(1-\frac{\tau}{G})^b})$.

where τ is the current generation number, G is the maximum number of generations and b is a user-defined constant.

2.2.2 Genetic Programming

Genetic programming (GP) was introduced by Koza [30], [31], [32]. GP usually manipulates a population of computer programs while GAs usually operate on a population of binary strings of a fixed length [56]. There are two main types of GP, namely, tree-based and linear GP. Tree-based GP is the more common type of GP. A computer program is usually represented in the form of a parse tree in tree-based GP. In other words, a population of parse trees is usually maintained in most GP systems. A parse tree can modify its size and shape dynamically. However, many GP algorithms restrict the depth of a parse tree.

A non-leaf node of a parse tree is called a primitive function. A terminal node of a parse tree is called a terminal. The sets of primitive functions and terminals are called function and terminal sets respectively. The function set consists of arithmetic operators and functions [56]. The terminal set includes constants and independent variables. The function set has the closure property, which means each primitive function accepts any terminal or any output produced by any function in the function set as an input parameter. It is necessary to define the function and the terminal sets in order to solve a problem using GP. Figure 2.4 shows an example parse tree. In this figure, the function and the terminal sets are $\{+, *\}$ and $\{X, Y, 2\}$ respectively.

In a traditional GP algorithm, an initial population of parse trees is generated randomly. To construct a parse tree, one of the elements in the function set is selected as the label for the root node of the parse tree. When a node labeled with a primitive function is created, two or more nodes are generated as

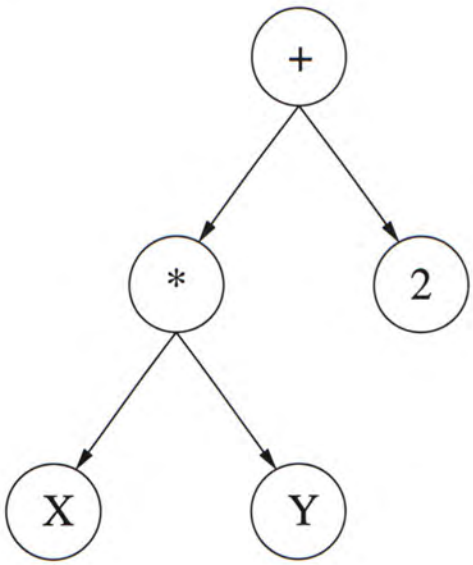

Figure 2.4: An Example Parse Tree

the children of the node, depending on the arity of the function. Each child node is labeled with either a primitive function or a terminal. The above process is repeated recursively until a child node labeled with a terminal is created. The following outlines the algorithm of a traditional GP system:

1. Set $\tau = 0$, where τ is the current generation number.
2. Initialize a population P_τ of parse trees.
3. Evaluate the fitness value of each individual in the population P_τ.
4. While all of the termination criteria are not satisfied,
   (a) Generate a new population $P_{\tau+1}$ of parse trees by selection, crossover and mutation.
   (b) Evaluate the fitness value of each parse tree in the population $P_{\tau+1}$.
   (c) Set $\tau = \tau + 1$.

5. Return the best parse tree as the computer program to a problem.

In the crossover operation, a pair of parent parse trees is randomly selected first. In each parental tree, one of its nodes is chosen as a crossover point. Offspring trees are produced by exchanging the subtrees rooted at the selected crossover points. Figure 2.5 illustrates a crossover operation between a pair of parse trees. The subtrees involved in the crossover operation are marked with dotted lines.

In the mutation operation, one of the parse trees is chosen first. One of the nodes of the parental tree is selected as the mutation point. The subtree rooted at the mutation point is replaced by a randomly generated subtree or leaf node. Figure 2.6 illustrates a mutation operation in a GP system.

2.2.3 Evolution Strategies

In evolution strategies (ES), a population of real-valued vectors is maintained. ES was originally used to solve real-valued function optimization problems [45], [50]. In a function optimization problem, the d-dimensional vector which maximizes or minimizes an objective function is determined.

Usually, each individual in ES is an ordered pair $X = (\mathbf{x}, \sigma)$, where $\mathbf{x} = (x_1, x_2..., x_d)$ and $\sigma = (\sigma_1, \sigma_2..., \sigma_d)$ are d-dimensional vectors. The vector $\mathbf{x}$ represents a point in the search space while the vector σ is the standard deviation vector for the mutation operator.

$(1+1)$-ES is the simplest and the earliest ES model. One offspring is generated by mutation in each geneation. The stronger individual between the parent and the offsping is preserved in the next generation. The following outlines the algorithm of $(1+1)$-ES:

1. Initialize the ordered pair $X = (\mathbf{x}, \sigma)$, where $\mathbf{x} = (x_1, x_2 ...,$

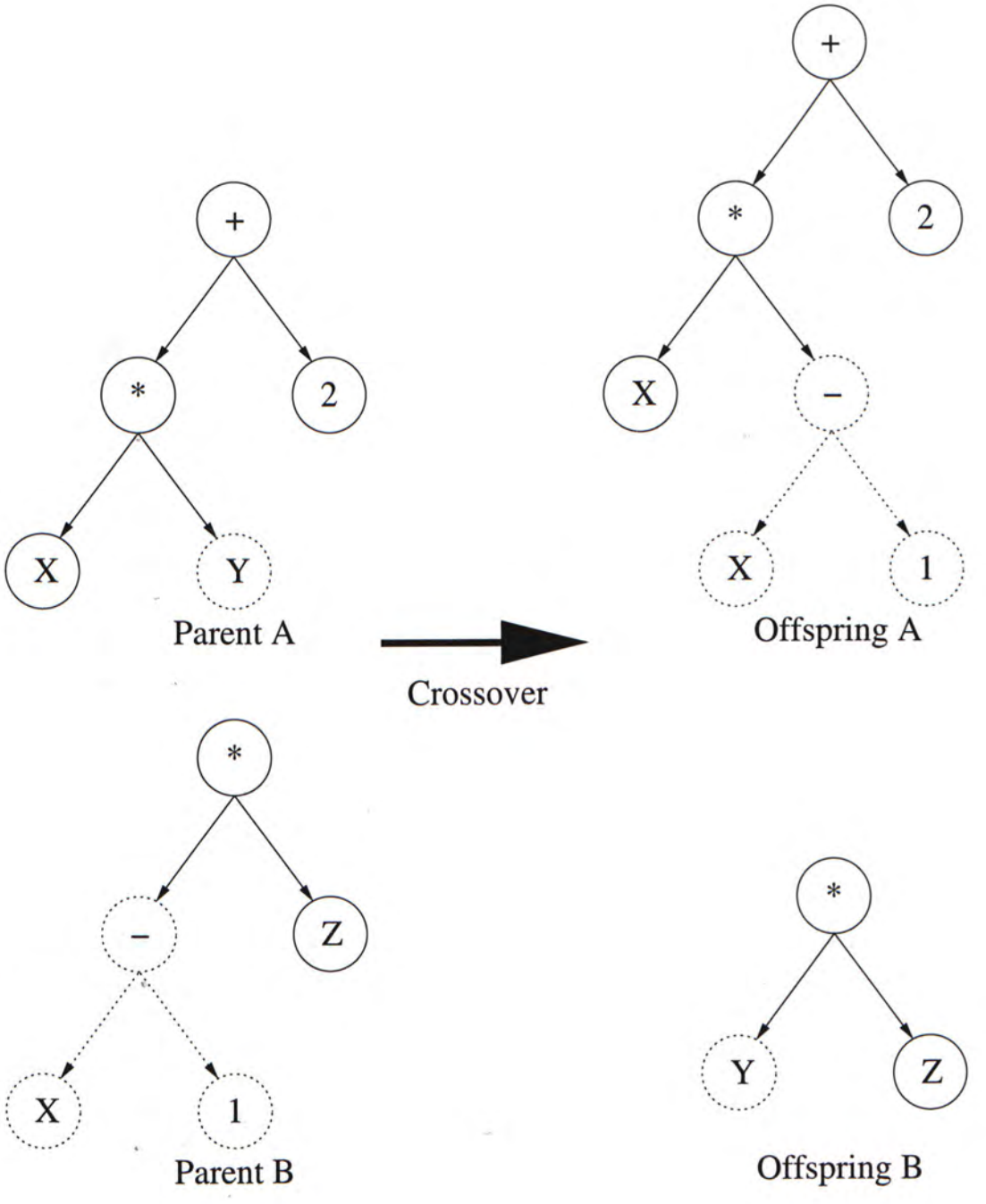

Figure 2.5: An Example that a Pair of Parse Trees Undergoes Crossover

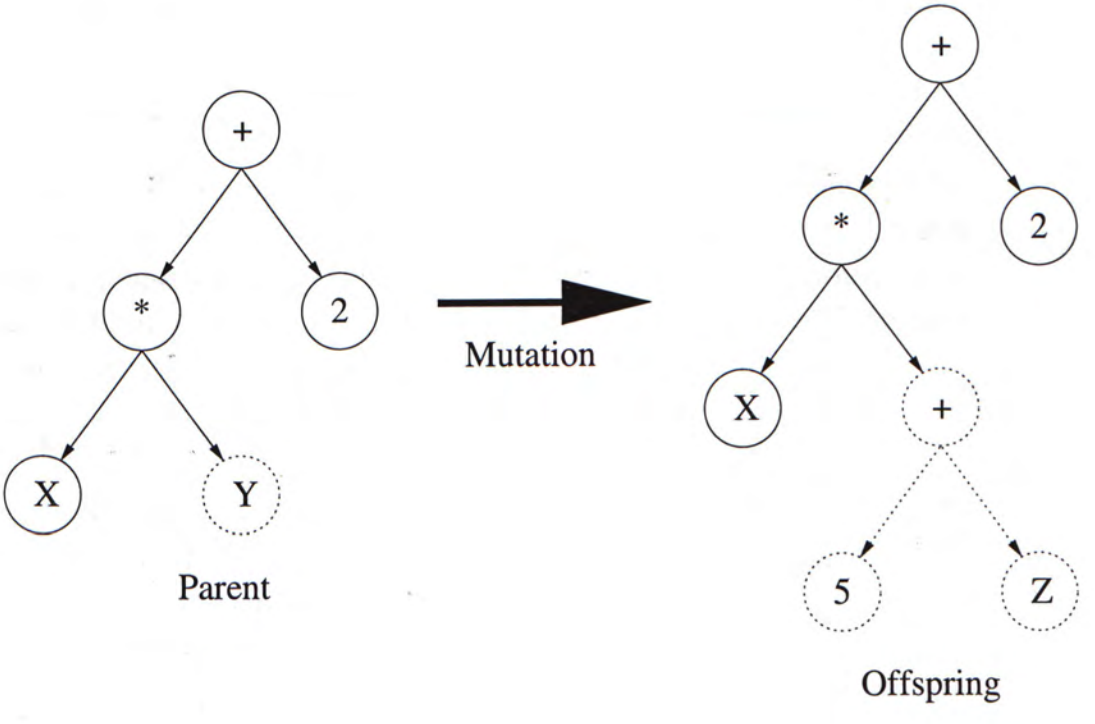

Figure 2.6: An Example that a Parse Tree Undergoes Mutation

$x_d)$ and $\sigma = (\sigma_1, \sigma_2 ..., \sigma_d)$ are d-dimensional vectors. The vector $\mathbf{x}$ represents a point in the search space while the vector σ is the standard deviation vector for the mutation operator.

2. While all of the termination criteria are not satisfied,
   (a) Generate the offspring $X' = (\mathbf{x}', \sigma')$ =Mutation(X).
   (b) If the offspring X' is better than the parent X, then set $X = X'$.
3. Return the vector $\mathbf{x}$ as the solution to a problem.

In $(\mu + 1)$-ES, one offspring is generated from μ parents by recombination and mutation in each generation. $(1 + 1)$-ES is a special case of $(\mu + 1)$-ES. The weakest individual among the μ parents and the offspring is abandoned. The following outlines the algorithm of $(\mu + 1)$-ES:

1. Set $\tau = 0$, where τ is the current generation number.
2. Initialize a population of μ ordered pairs $P_\tau = \{X_1, X_2 ..., X_\mu\}$, where $X_i = (\mathbf{x_i}, \sigma_\mathbf{i})$, $\mathbf{x_i} = (x_{i,1}, x_{i,2} ..., x_{i,d})$ and $\sigma_\mathbf{i} = (\sigma_{i,1}, \sigma_{i,2} ..., \sigma_{i,d})$ are d-dimensional vectors. The vector $\mathbf{x_i}$ represents a point in the search space while the vector $\sigma_\mathbf{i}$ is the standard deviation vector for the mutation operator.
3. While all of the termination criteria are not satisfied,
   (a) Generate the offspring $X' = (\mathbf{x}', \sigma') =$ Recombination(P_τ).
   (b) Generate the offspring $X'' = (\mathbf{x}'', \sigma'')$ = Mutation(X').
   (c) Let X_{worst} be the weakest individual among the μ parents and the offspring X''.
   (d) Set $P_{\tau+1} = (P_\tau \cup \{X''\}) \setminus \{X_{worst}\}$.
   (e) Set $\tau = \tau + 1$.

4. Return the best individual in the population P_τ as the solution to a problem.

In $(\mu+\lambda)$-ES, λ descendants are generated from μ parents by recombination and mutation in each generation. The μ strongest individuals among the μ parents and the λ offspring are preserved in the next generation. The following outlines the algorithm of $(\mu + \lambda)$-ES:

1. Set $\tau = 0$, where τ is the current generation number.
2. Initialize a population of μ ordered pairs $P_\tau = \{X_1, X_2 ..., X_\mu\}$, where $X_i = (\mathbf{x_i}, \sigma_\mathbf{i})$, $\mathbf{x_i} = (x_{i,1}, x_{i,2}..., x_{i,d})$ and $\sigma_\mathbf{i} = (\sigma_{i,1}, \sigma_{i,2}..., \sigma_{i,d})$ are d-dimensional vectors. The vector $\mathbf{x_i}$ represents a point in the search space while the vector $\sigma_\mathbf{i}$ is the standard deviation vector for the mutation operator.
3. Set $P' = P_\tau$.
4. While all of the termination criteria are not satisfied,
   (a) FOR $i = 1$ TO λ DO
      i. Generate the offspring $X_i' = (\mathbf{x_i'}, \sigma_\mathbf{i}') =$ Recombination(P_τ).
      ii. Generate the offspring $X_i'' = (\mathbf{x_i''}, \sigma_\mathbf{i}'') =$ Mutation(X_i').
      iii. Set $P' = P' \cup \{X_i''\}$.
   (b) Select the μ strongest individuals in the population P' to form the population $P_{\tau+1}$.
   (c) Set $\tau = \tau + 1$.
5. Return the best individual in the population P_τ as the solution to a problem.

In (μ, λ)-ES, λ descendants are generated from μ parents by recombination and mutation in each generation. The μ strongest

individuals among the λ descendants are preserved in the next generation. The following outlines the algorithm of (μ,λ)-ES:

1. Set $\tau = 0$, where τ is the current generation number.
2. Initialize a population of μ ordered pairs $P_\tau = \{X_1, X_2 \ldots, X_\mu\}$, where $X_i = (\mathbf{x_i}, \sigma_\mathbf{i})$, where $\mathbf{x_i} = (x_{i,1}, x_{i,2} \ldots, x_{i,d})$ and $\sigma_\mathbf{i} = (\sigma_{i,1}, \sigma_{i,2} \ldots, \sigma_{i,d})$ are d-dimensional vectors. The vector $\mathbf{x_i}$ represents a point in the search space while the vector $\sigma_\mathbf{i}$ is the standard deviation vector for the mutation operator.
3. While all of the termination criteria are not satisfied,
   (a) FOR $i = 1$ TO λ DO
      i. Generate the offspring $X'_i = (\mathbf{x'_i}, \sigma'_\mathbf{i}) =$ Recombination(P_τ).
      ii. Generate the offspring $X''_i = (\mathbf{x''_i}, \sigma''_\mathbf{i}) =$ Mutation(X'_i).
   (b) Set $P' = \bigcup_{i=1}^{\lambda} \{X''_i\}$.
   (c) Select the μ strongest individual in the population P' to form the population $P_{\tau+1}$.
   (d) Set $\tau = \tau + 1$.
4. Return the best individual in the population P_τ as the solution to a problem.

There are several recombination operators proposed in the literature [50]. In discrete recombination, two individuals are selected first. d uniformly distributed random numbers $u_1, u_2 \ldots, u_d \in [0, 1]$ are then generated. Suppose $X_a = (\mathbf{x'_a}, \sigma'_\mathbf{a})$ and $X_b = (\mathbf{x'_b}, \sigma'_\mathbf{b})$, are selected as parents, an offspring $X = (\mathbf{x'}, \sigma')$ is generated such that:

$$x'_i = \begin{cases} x_{a,i} & \text{if } u_i > 0.5, \\ x_{b,i} & \text{otherwise,} \end{cases} \tag{2.7}$$

$$\sigma_i' = \begin{cases} \sigma_{a,i} & \text{if } u_i > 0.5, \\ \sigma_{b,i} & \text{otherwise,} \end{cases} \tag{2.8}$$

where $i = 1, 2..., d$.

In intermediate recombination, two individuals are selected first. Suppose $X_a = (\mathbf{x}_\mathbf{a}', \sigma_\mathbf{a}')$ and $X_b = (\mathbf{x}_\mathbf{b}', \sigma_\mathbf{b}')$ are selected as parents, an offspring $X = (\mathbf{x}', \sigma')$ is generated such that:

$$x_i' = 0.5(x_{a,i} + x_{b,i}) \tag{2.9}$$

$$\sigma_i' = 0.5(\sigma_{a,i} + \sigma_{b,i}) \tag{2.10}$$

The following outlines the steps to create an offspring $X' = (\mathbf{x}', \sigma')$ from the individual $X = (\mathbf{x}, \sigma)$ by the mutation operator:

1. Set $\nu = N(0, 1)$.
2. Set $\sigma_i' = \sigma_i exp(\tau_1 N(0, 1) + \tau_2 \nu)$, where τ_1 and τ_2 are user-defined constants.
3. Set $x_i' = x_i + \sigma_i' N(0, 1)$.

2.2.4 Evolutionary Programming

Evolutionary Programming (EP) is a probabilistic optimization strategy inspired by the concepts of Darwinian evolution [15], [12]. The behavioural relationship between parents and their offspring is emphasized [56]. New offspring are produced by mutation only. No recombination is applied to generate offspring. The following outlines the steps of an EP algorithm:

1. Set $\tau = 0$, where τ is the current generation number.
2. Initialize a population P_τ of individuals.
3. Evaluate the fitness value of each individual in the poupla-tion P_τ.
4. While all of the termination criteria are not satisified,

(a) Generate one or more offsping for each individual in the population P_τ by mutation.

(b) Evaluate the fitness value of each offspring.

(c) Select the individuals for the population $P_{\tau+1}$ using a stochastic tournament selection.

(d) Set $\tau = \tau + 1$.

5. Return the best individual in the population P_τ.

2.3 Applications of Evolutionary Algorithms to Induction of Classification Rules

EAs are applied to evolve a set of classification rules because the search space can be very large. In this section, several EA-based rule induction algorithms such as SCION [33], [34], GABIL [29] and LOgic grammar based GENetic PROgramming (LOGENPRO) [55] are briefly discussed.

2.3.1 SCION

Leung et al. introduced a rule-based expert system called SCION. In SCION, GAs are applied to evolve a set of classification rules. The antecedent part of a classification rule is a conjunctive set of inequalities involving one continuous attribute, while the consequent part of the rule represents the class label. When a sample satisfies the antecedent part of a rule, the sample is classified according to the class label associated with the rule.

A population of chromosomes is partitioned into C subpopulations, where C is the number of possible classes. There is an associated class label in each subpopulation. A chromosome classifies a sample into the class associated with its subpopulation if the corresponding hyperrectangle contains the sample. Each chromosome is a sequence of duples, where the i^{th} duple

represents the lower and the upper bounds of the i^{th} input attribute.

Token competition is applied to remove redundant rules. Redundant rules may be produced because the individuals tend to contain similar sets of training samples. Under token competition, the diversity of the chromosomes in each subpopulation is maintained. The chromosomes in each subpopulation are sorted according to their fitness values first. For each training sample, its only one token is assigned to the strongest chromosome which is capable of classifying the sample correctly. Stronger chromosomes can obtain more tokens while weaker ones may fail to obtain any token. Chromosomes which fail to obtain any token are eliminated.

Rule migration was introduced in the SCION system because a weak chromosome for a subpopulation may be a strong chromosome for another subpopulation. In the i^{th} subpopulation, when the fitness value of a weak chromosome for the k^{th} subpopulation is greater than the average fitness value of the chromosomes in the k^{th} subpopulation, where $k \neq i$, it is migrated to the k^{th} subpopulation.

2.3.2 GABIL

GABIL was introduced by De Jong et al. in 1993. A variable-length binary string is used to represent classification rules in disjunctive normal form (DNF). The antecedent part of each rule is a conjunction of one or more conditions, each of which involves one of the input attributes. The consequenct part of a rule is the class label. The fitness value of an individual is the square of the percentage of correctly classified training samples.

Each individual represents a set of classification rules. The number of bits to represent a single classification rule is fixed. When an individual undergoes a genetic operation, extra bits

may be added to the individual if necessary.

2.3.3 LOGENPRO

A data mining system, called LOgic grammar based GENetic PROgramming (LOGENPRO), was introduced by Wong et al. LOGENPRO is capable of evolving a set of classification rules using GP.

Each individual is a derivation tree, which is used to represent a classification rule. But the search space can be very large and a grammar has to be designed to control the creation of derivation trees. All the individuals should conform to a specified grammar to accelerate the search for a set of classification rules. When a new derivation tree is generated by genetic operators such as crossover and mutation, it is necessary to check whether the produced tree still obeys a specified grammar. Users are allowed to specify the structure of a rule using a user-defined grammar.

Fitness sharing [19] and token competition are employed to maintain the diversity of a population. In many cases, more than one individual is chosen as the set of classification rules.

2.4 Applications of Evolutionary Algorithms to Construction of Decision Trees

Several decision tree algorithms employing EAs were proposed in the literature, including binary-tree genetic algorithm (BTGA) [9], OC1-GA [8], OC1-ES [8], GATree [42] and so on. The following subsections describe these algorithms in brief.

2.4.1 Binary Tree Genetic Algorithm

In Binary Tree-Genetic Algorithm (BTGA), the decision function at each non-leaf node is a linear combination of all input

attributes. GAs are applied to find the optimal linear decision function at each non-leaf node. A binary string is used to represent an individual. The fitness value of a chromosome depends on the impurity reduction after partitioning a set of training samples into two disjoint subsets. The Gini index is chosen to evaluate the impurity of a set of training samples. A rank based selection is applied to select chromosomes to be replicated to the mating pool. The chromosomes in the mating pool undergo two-point crossover and mutation. The process is repeated recursively until the number of training samples arriving at a node is less than a positive integer n_0 or the impurity of a set of training samples is less than a threshold g_0. It means prepruning is employed to control the growth of a decision tree. Figure 2.7 shows the algorithm of the procedure `createBTGA()`, which outlines the steps to construct a linear decision tree using BTGA. The procedure `createBTGA()` accepts the training set as the parameter.

2.4.2 OC1-GA

OC1-GA was extended from the OC1 algorithm proposed by Murthy et al. [38]. At each non-terminal node of an oblique decision tree, the optimal linear decision function is found using GAs. OC1-GA is a RCGA because it uses a real-valued vector to represent a chromosoome. A population of chromosomes is initialized such that one-tenth of the population is the best axis-parallel hyperplane at each non-leaf node of an oblique decision tree. In OC1-GA, pairwise tournament selection is used to choose chromosomes to be copied to the mating pool. Uniform crossover is applied to the chromosomes in the mating pool. Postpruning is used to control the size of an oblique decision tree.

PROCEDURE `createBTGA`

INPUT A set of training samples S_h

OUTPUT A new node N_h

1. IF $|S_h|$ is less than a positive integer n_0 or the impurity of S_h is less than a threshold g_0, THEN the node N_h is declared as a leaf node and go to step 6.
2. Find the optimal linear decision function $\mathbf{b^T x} > \gamma'$, where $\mathbf{b} = (b_1, b_2, ..., b_d)^T$ is a d-dimensional column vector and γ' is a real constant, using a BCGA such that the impurity reduction after partitioning the set S_h into two disjoint subsets is maximized.
3. IF the impurity reduction is less than g_0, THEN the node N_h is declared as a leaf node and go to step 6.
4. Define $R'_h = \{\mathbf{x} \in S_h | \mathbf{b}^T\mathbf{x} > \gamma'\}$ and $L'_h = S_h \setminus R'_h$.
5. Invoke `createBTGA(`R'_h`)` and `createBTGA(`L'_h`)`, and go to step 7.
6. Determine the class label associated with the node N_h.
7. Return the node N_h.

Figure 2.7: The Algorithm of the Procedure `createBTGA()`

2.4.3 OC1-ES

OC1-ES was also extended from the OC1 algorithm proposed by Murthy et al. [38]. At each non-terminal node of an oblique decision tree, the optimal linear decision function is determined using a (1+1)-ES with self-adaptive mutations. The initial linear decision function is equivalent to the best axis-parallel hyperplane in the attribute space. There is a mutation coefficient for each coefficient of the linear decision function to be optimized. All mutation coefficients are initially set to 1. In OC1-ES, one offspring is generated from the parent by mutation. The size of an oblique decision tree is controlled by postpruning.

2.4.4 GATree

GAs are applied to construct univariate decision trees. There are two child nodes at each non-leaf node of the decision trees evolved by GATree [42]. To evolve the optimal decision tree, a population of minimal binary decision trees, which have the root node and two leaf nodes, is initialized first. When two parents are selected to undergo crossover, one of the nodes of each subtree is selected as a crossover point. The subtrees rooted at the selected crossover points are then swapped. When an individual undergoes mutation, one of its nodes is selected first. If the selected node is a non-leaf node, the value of the corresponding decision function is changed to a random value. Otherwise, its class label is randomly changed to one of the possible classes. The fitness value f_i of the i^{th} individual is given by:

$$f_i = n_{C,i}^2 \times \frac{k}{n_{T,i}^2 + k} \tag{2.11}$$

where $n_{C,i}$ is the number of training samples correctly classified by the i^{th} individual, $n_{T,i}$ is its tree size and k is a user-defined constant. To reduce the computational time, a modified version

of limited error fitness (LEF) [18] is employed. When the number of misclassifications of an individual is greater than an error limit, all the remaining training samples will not be evaluated.

2.4.5 Induction of Linear Decision Trees using Strong Typing GP

In [5] and [4], an oblique decision tree is evolved using a strong typing GP. In a strong typing GP, the data type for each element in the terminal set is specified. Moreover, each function in the function set specifies the data types for all arguments and the data type of the output returned by the function. When an invalid individual is generated by crossover or mutation, it is modified so the the restrictions on the data types are satisfied. Tournament selection is used to select chromosomes to be replicated to the mating pool.

There are two possible ways to avoid the problem of code bloat. The fitness value of an individual is a weighted sum of the number of correctly classified training samples and its tree size in number of nodes.

In the second method, Pareto scoring with fitness sharing [19] is applied with two dimensions, including the number of misclassifications and the number of nodes of an individual. The advantage of this method over the first one is that there is no need to adjust the weights in order to produce better oblique decision trees.

To reduce the computational time, limited error fitness (LEF) [18] is employed. When the number of misclassifications of an individual is greater than an error limit, all the remaining training samples will not be evaluated. The error limit may be changed when oblique decision trees are being evolved. The error limit increases when the number of misclassifications of the best individual is greater than the error limit and decreases if the number

of misclassifications of the worst individual is lower than the error limit.

2.5 Spatial Data Structures and its Applications

2.5.1 Spatial Data Structures

k-D trees [3] and quadtrees are hierarchical data structures based on the principle of recursive decomposition of space [49]. Applications of these data structures include computer graphics, image processing, geographic information system (GIS), database management system, data mining and so on.

A k-D tree is a binary tree. There are two child nodes at each non-leaf node of a k-D tree. At each non-leaf node of a k-D tree, one of the attributes is chosen to divide a space into two subspaces. The choice of the attribute for partitioning a space at a non-leaf node depends on its depth. Each leaf node of a k-D tree may represent a region, a pixel or a record depending on its application. In my research work, each leaf node represents the smallest hyperrectangle containing a small subset of training samples. Figure 2.8 shows an example k-D tree.

A quadtree recursively subdivides a two-dimensional space into four quadrants. There are at most four child nodes at each non-leaf node of a quadtree. At each non-leaf node of a quadtree, two attributes are applied to divide a two-dimensional space into four quadrants. Each leaf node of a quadtree may represent an area or a pixel depending on its application. Figure 2.9 shows an example quadtree.

An octree is similar to a quadtree, except that a three-dimensional space is partitioned into eight octants recursively. There are at most eight child nodes at each non-leaf node of an octree. At each non-leaf node of an octree, a three-dimensional space is di-

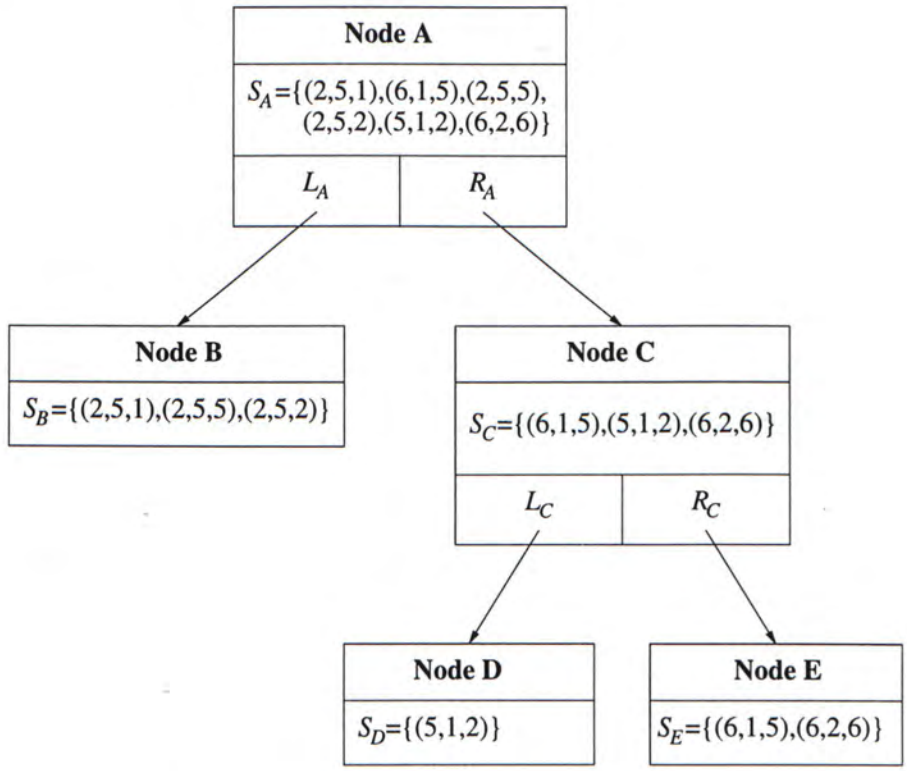

Figure 2.8: An Example k-D Tree

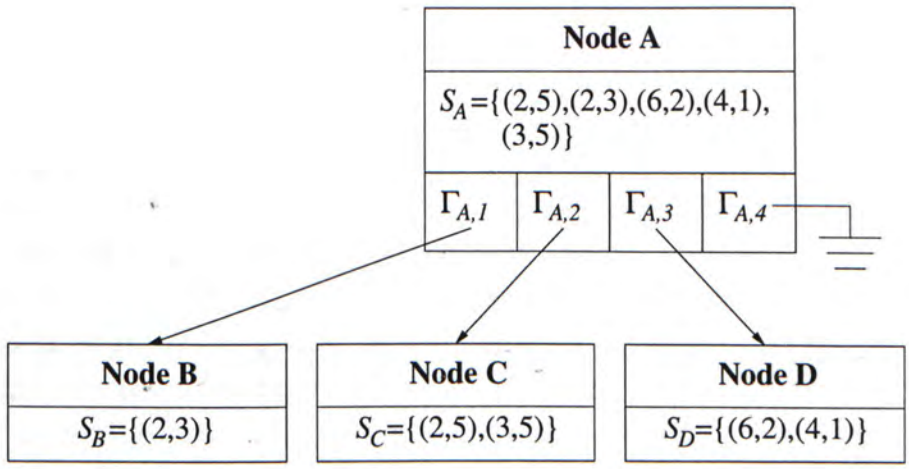

Figure 2.9: An Example Quadtree

vided into eight octants using three attributes. Each leaf node of an octree may represent an voxel in a three-dimensional image.

A generalized quadtree is a generalization of a quadtree in a higher dimensional space. There are at most 2^d children at each non-leaf node of a generalized quadtree, where d is the dimensionality of the space to be represented. All the attributes are applied to divide a d-dimensional space at each non-leaf node of a genearlized quadtree. Each leaf node may represent a record having d attributes in a database. In this thesis, each leaf node represents the smallest hyperrectangle containing a small subset of training samples.

2.5.2 Applications of Spatial Data Structures

Spatial data structures can be applied in various fields of computer science, including computer graphics, image processing, database management system and so on. Moreover, they can also be applied in geographic information systems (GIS).

In computer graphics, curvilinear data such as polygons can be represented using a quadtree. A possible way to represent a polygon is described as follows. When a region contains more than one line segment, it is divided into four quadrants recursively until a rectangle containing a single straight line is obtained. At each leaf node of a quadtree, the information about the straight line passing thourgh the rectangle such as direction, intercept and intensity is stored [52].

An octree can be used to represent a three dimensional object. An object is decomposed recursively until a single vertex or a single edge is obtained. A leaf node of an octree represents a single vertex or a single edge of an object.

In image procesing, a quadtree can be built to represent an image, which is a two-dimensional array of pixels. A leaf node correponds to a square array of one or more homogeneous pixels

in an image. In addition, the color of a pixel in an image can be determined by considering the quadtree constructed from the image. To find the color of a pixel, the root node of a quadtree is considered first. Then the descendants containing the pixel are considered successively until the leaf node containing the pixel is found. Moreover, basic image opeartions such as dithering [28] and windowing can be performed using the quadtree constructed from the image to be processed. In dithering, each pixel of a grayscaled image is converted to either black or white while maintaining as much similarity to the original image as possible. Windowing is the process of extracting a rectangular array of pixels from an image.

When a quadtree is constructed from an image, the image can be viewed at different levels of resolution. This is particularly useful when an image is transmitted through a communication channel with lower bandwitdth. In this case, a low resolution image can be viewed first and the more detailed one can be shown later.

In data mining, a quadtree may be constructed to improve the efficiency of a nearest neighbor classifier. To determine the class label of an unseen sample using a nearest neigbor classifier, say one-nearest neighbor classifier, the class label of the nearest training sample with respect to the unseen sample is determined. It is necessary to calculate the distance between the unseen sample and each of the training samples. When a quadtree is constructed from a set of training samples, only a subset of the training samples are needed to be considered.

In GIS, a powerline, a cityline map or a roadline map can be represented by a quadtree [48]. A powerline specifies the path of the main powerline within a region. A cityline map shows the border of a city. A roadway network within a region can be represented using a roadline map.

□ **End of chapter.**

Chapter 3

Induction of Classification Rules using Genetic Algorithms

3.1 Introduction

In this chapter, a novel rule-learning algorithm, called genetic algorithm-based convex polytope rule learning system (GA-based CPRLS), is proposed by extending a rule-based expert system called SCION [33], [34]. In the GA-based CPRLS, genetic algorithms (GAs) are applied to evolve a set of rules for classifications. The antecedent part of each rule is a conjunctive set of logical expressions. The possible logical expressions include linear inequalities with one or more continuous attributes and nominal attribute-value pairs. When a sample satisfies the antecedent part of a rule, it is classified according to the class label associated with the rule.

Although the antecedent part of a classification rule is a conjunctive set of logical expressions in both SCION and the GA-based CPRLS, there is a major difference between the GA-based CPRLS and SCION. The possible logical expressions include linear inequalities involving one or more numeric attributes and nominal attribute-value pairs in the GA-based CPRLS. In SCION, each logical expression is an inequality with one numeric attribute only. It means SCION cannot handle nominal attributes.

In other words, the GA-based CPRLS is a generalization of SCION.

3.2 Rule Learning using Genetic Algorithms

Before learning a set of rules using GAs, a set of samples is partitioned into the training and the testing sets. The training set is applied to evolve a set of rules for classifications. A good set of rules should minimize the number of misclassifications on the testing set, rather than the training set.

A population of chromosomes is initialized and maintained in the GA-based CPRLS. Each chromosome represents a conjunctive set of logical expressions, which may include linear inequalities with several continuous attributes and nominal attribute-value pairs. The population is partitioned into C subpopulations, where C is the number of possible classes. Each subpopulation is associated with a distinct class label. In each subpopulation, each chromosome classifies a sample as the same class if the sample satisfies the antecedent part of the corresponding rule.

After a population of chromosomes is initialized, the fitness value of each chromosome is evaluated. The higher the fitness value of a chromosome is, the better the corresponding rule is. Token competition is applied to remove redundant chromosomes in each subpopulation. Weaker chromosomes are also removed if necessary. Rule migration allows a weak chromosome for a particular class to become a chromosome in another subpopulation provided that it is a strong chromosome for the associated class. Common operators of GAs including crossover and mutation are applied to the survived chromosomes in each subpopulation. In each generation, the total number of correctly classified training samples is evaluated in order to preserve the best-so-far set of rules in the next generation. The above processes are repeated

until the maximum number of generations G is reached. Finally, a set of rules for classifications is constructed from the best set of chromosomes. The following outlines the steps of learning rules in the GA-based CPRLS.

- Initialize a population of chromosomes P.
- FOR $\tau = 1$ TO G DO
  - FOR $i = 1$ TO C DO
    1. Evaluate the fitness value of each chromosome in the subpopulation P_i using the fitness function in (3.9).
    2. Sort the chromosomes in the subpopulation P_i in the descending order of their fitness values.
    3. Apply token competition to the subpopulation P_i.
    4. Remove all chromosomes in the subpopulation P_i which fail to obtain any tokens.
    5. Remove weaker chromosomes if the number of chromosomes in the subpopulation P_i is greater than the maximum number of parent chromosomes Q_P.
  - Migrate rules among the subpopulations $P_1, P_2..., P_C$.
  - Apply crossover and mutation to the subpopulations $P_1, P_2..., P_C$.
  - Calculate the total number of training samples correctly classified by the population of chromosomes.

3.2.1 Population Initialization

A population of chromosomes are initialized before evolving a set of rules using GAs. The population is divided into C subpopulations $P_1, P_2..., P_C$ where C is the number of possible classes.

Each chromosome in the subpopulation P_i, $i = 1, 2..., C$, classifies a sample as class i if the sample satisfies the antecedent part of the corresponding rule.

Let $\Theta_{i,j} = (\Omega_{i,j}, \alpha_{\mathbf{i,j}})$, $i \in \{1, 2..., C\}$, $j \in \{1, 2..., L\}$, be a duple denoting the j^{th} chromosome in the subpopulation P_i, where L is the number of chromosomes in each subpopulation. A population of $C \times L$ chromosomes is maintained.

In this chapter, $\Omega_{i,j}$ is defined as the set of linear inequalities for the chromosome $\Theta_{i,j}$. Let $\omega_{\mathbf{i,j,k}} = (w_{i,j,k,1}, w_{i,j,k,2}..., w_{i,j,k,d+1})$, $i \in \{1, 2..., C\}$, $j \in \{1, 2..., L\}$, $k \in \{1, 2..., H\}$, be the $(d+1)$-dimensional vector specifying the coefficients of the k^{th} inequality in the set $\Omega_{i,j}$, where d is the number of continuous attributes and H is the number of inequalities. The inequality represented by the vector $\omega_{\mathbf{i,j,k}}$ is given by:

$$\sum_{m=1}^{d} w_{i,j,k,m} x_m > w_{i,j,k,d+1}. \tag{3.1}$$

Suppose $\mathbf{y} = (y_1, y_2..., y_d)$ and $\mathbf{z} = (z_1, z_2..., z_d)$, where y_i and z_i, $i = 1, 2..., d$, are respectively the minimum and the maximum values of the i^{th} numeric attribute in the training set. The inequality specified by the vector $\omega_{\mathbf{i,j,k}}$, $i \in \{1, 2..., C\}$, $j \in \{1, 2..., L\}$, satisfies the following conditions:

$$-1 \leq w_{i,j,k,m} \leq 1 \quad , m = 1, 2..., d, \tag{3.2}$$

$$d_1 \leq w_{i,j,k,d+1} \leq d_2 \tag{3.3}$$

where

$$d_1 = \sum_{m=1}^{d} \frac{|w_{i,j,k,m}|(y_m + z_m) + w_{i,j,k,m}(y_m - z_m)}{2} \tag{3.4}$$

$$d_2 = \sum_{m=1}^{d} \frac{|w_{i,j,k,m}|(y_m + z_m) + w_{i,j,k,m}(z_m - y_m)}{2} \tag{3.5}$$

$$\sum_{m=1}^{d} w_{i,j,k,m}^2 = 1 \tag{3.6}$$

On the other hand, $\alpha_{i,j} = (\alpha_{i,j,1}, \alpha_{i,j,2}..., \alpha_{i,j,d'})$, $i \in \{1, 2..., C\}$, $j \in \{1, 2..., L\}$, is a d'-dimensional integer-valued vector representing the nominal attribute-value pairs for the chromosome $\Omega_{i,j}$, where $\alpha_{i,j,k} \in \{0, 1, 2, ...h_k\}$, $k \in \{1, 2, ..., d'\}$, represents the value of the k^{th} nominal attribute, h_k is the number of possible values of the k^{th} nominal attribute and d' is the number of nominal attributes. The equivalent logical expression for the k^{th} nominal attribute is given by:

$$x_k \in \begin{cases} \{\alpha_{i,j,k}\} & \text{if } \alpha_{i,j,k} \in \{1, 2, ..., h_k\} \\ \{1, 2, ..., h_k\} & \text{if } \alpha_{i,j,k} = 0 \end{cases} \tag{3.7}$$

If a sample $\mathbf{x} = (x_1, x_2..., x_d, x'_1, x'_2..., x'_{d'})$, where x_i, $i \in \{1, 2..., d\}$, is a continuous attribute and x'_j, $j \in \{1, 2..., d'\}$, is a nominal attribute, satisfies the logical expression represented by the chromosome $\Theta_{i,j}$, or equivalently,

$$[\bigwedge_{k=1}^{H} (\sum_{m=1}^{d} w_{i,j,k,m} x_m > w_{i,j,k,d+1})] \wedge [\bigwedge_{k=1}^{d'} ((\alpha_{i,j,k} = 0) \vee (x'_k = \alpha_{i,j,k}))] \tag{3.8}$$

the sample is classified as class i.

3.2.2 Fitness Evaluation of Chromosomes

The fitness value of each chromosome is evaluated. The higher the fitness value of a chromosome is, the stronger the corresponding rule is. Given $S_{i,j}$, $i \in \{1, 2..., C\}$, $j \in \{1, 2..., L\}$, is the set of training samples satisfying the condition specified by the chromosome $\Theta_{i,j}$ and $S'_{i,j}$ is the set of training samples of class i in the set $S_{i,j}$. The fitness value of the chromosome $\Theta_{i,j}$, $i \in \{1, 2..., C\}$, $j \in \{1, 2..., L\}$, is given by:

$$f_{i,j} = \frac{|S'_{i,j}|}{|S_{i,j}| + 1} \tag{3.9}$$

The numerator of the above equation equals the number of training samples correctly classified by the chromosome $\Theta_{i,j}$. One is added to the denominator in order to avoid a divide by zero error when all the training samples cannot be classified by the chromosome.

3.2.3 Token Competition

In each subpopulation, the chromosomes tend to cover a similar set of training samples as the rule learning proceeds. This reduces the diversity of the chromosomes in each subpopulation.

A similar problem is also addressed in [33], [34]. Token competition can be employed to remove redundant chromosomes. In the GA-based CPRLS, token competition is applied independently to each subpopulation. In the subpopulation P_i, $i = 1, 2..., C$, the chromosomes are sorted in the descending order of their fitness values first. For each training sample, one token is assigned to the strongest chromosome which is capable of classifying it correctly. Once a token is assigned by a training sample, other chromosomes cannot obtain any tokens for that sample although they can correctly classify it. The stronger chromosomes are capable of acquiring more tokens. In each subpopulation, it is more difficult for weaker chromosomes similar to the stronger ones to receive tokens. Chromosomes which fail to obtain any tokens are then eliminated. A more concise set of stronger rules can be produced under token competition. Moreover, redundant rules can be eliminated and the diversity of the chromosomes in each subpopulation is increased.

Given the training set $S = \{\mathbf{s_1}, \mathbf{s_2}...\mathbf{s_n}\}$ has n training samples, where $\mathbf{s_i} = (x_{i,1}, x_{i,2}..., x_{i,d}, x'_{i,1}, x'_{i,2}..., x'_{i,d'}, c_i)$, $i = 1, 2..., n$, is the i^{th} sample in the training set, $x_{i,1}, x_{i,2}..., x_{i,d}$ are numeric

attributes, $x'_{i,1}, x'_{i,2}..., x'_{i,d'}$ are nominal attributes and c_i is the class label of the sample $\mathbf{s_i}$, the following outlines the steps of token competition:

- FOR $i = 1$ TO C DO
  - Sort the chromosomes $\Theta_{i,1}, \Theta_{i,2}..., \Theta_{i,L}$ in the subpopulation P_i according to the descending order of their fitness values.
  - FOR $j = 1$ TO L DO
    1. Set $t_{i,j} = 0$, where $t_{i,j}$ is the number of tokens acquired by the chromosome $\Theta_{i,j}$.
    2. Let $\Theta_{i,(j)} = \{\Omega_{i,(j)}, \alpha_{\mathbf{i,(j)}}\}$, $j = 1, 2..., L$ be the j^{th} strongest chromosome in the subpopulation P_i.
  - FOR $j = 1$ TO $|S|$ DO
    FOR $k = 1$ TO L DO
    IF $\bigwedge_{m=1}^{H}(\sum_{p=1}^{d} w_{i,(k),m,p}x_{j,p} > w_{i,(k),m,d+1})]$, THEN
    IF $[\bigwedge_{m=1}^{d'}((\alpha_{i,(k),m} = 0) \vee (x'_{j,m} = \alpha_{i,(k),m}))]$, THEN
    1. Set $t_{i,(k)} = t_{i,(k)} + 1$.
    2. Quit the innermost FOR loop.

3.2.4 Chromosome Elimination

If a chromosome fails to acquire any tokens under token competition, it will be eliminated. When the number of chromosomes in each subpopulation is still greater than Q_P, weaker chromosomes will be eliminated and imprisoned. In order to include the effect of token competition, another fitness function should be used to evaluate the strength of a chromosome. The modified fitness value of the chromosome $\Theta_{i,j}$, $i \in \{1, 2..., C\}$, $j \in \{1, 2..., L\}$, is given by:

$$f'_{i,j} = \frac{t_{i,j}}{|S_{i,j}| + 1} \tag{3.10}$$

Note that the numerator of the above equation equals the number of tokens acquired by the chromosome, rather than the number of training samples correctly classified. The denominator of the above equation depends on the number of training samples satisfying the condition represented by the chromosome $\Theta_{i,j}$.

The survived chromosomes in each subpopulation become parent chromosomes. Parent chromosomes are allowed to produce their offspring using crossover and mutation. It is hoped that better chromosomes can be produced by these genetic operators. The imprisoned chromosomes in a subpopulation may be migrated to another subpopulations, depending on its strength for other classes.

In each subpopulation, the probability of a parent chromosome selected for crossover or mutation depends on the rank of its fitness value calculated by (3.10). Chromosomes which fail to obtain any tokens are never selected. When a parent chromosome is in rank i, $i = 1, 2..., Q_P$, the probability of the chromosome chosen for crossover or mutation is:

$$(\frac{1}{2})^{min(i,Q_P-1)} \tag{3.11}$$

3.2.5 Rule Migration

A weak chromosome for a particular class may be a strong chromosome for another classes. In the subpopulation P_i, $i = 1, 2..., C$, the average fitness value $\overline{f_i}$ of the survived chromosomes is calculated first. The fitness value of each imprisoned chromosome in the subpopulation P_i, $i = 1, 2..., C$, is computed for all classes except class i. If the fitness value of an imprisoned chromosome in the subpopulation P_i is greater than $\overline{f_j}$ for class j, $j \in \{1, 2..., C\} \setminus \{i\}$, it will be migrated to the subpopulation P_j, provided that the number of migrated chromosomes for the subpopulation P_j is less than the migration quota Q_M. The following outlines the steps of rule migration.

- FOR $i = 1$ TO C DO

  1. Evaluate the average fitness value $\overline{f_i}$ of the survived chromosomes in the subpopulation P_i.
  2. Set $m_i = 0$, where m_i is the number of migrated chromosomes for the subpopulation P_i.

- FOR $i = 1$ TO C DO

  1. Set I as the number of imprisoned chromosomes in the subpopulation P_i.
  2. Let $\overline{\Theta_{i,1}}, \overline{\Theta_{i,2}}, ...\overline{\Theta_{i,I}}$ be the imprisoned chromosomes in the subpopulation P_i.
  3. FOR $j = 1$ TO I DO
     - FOR $k = 1$ TO C DO
       IF $k \neq i$, THEN
       (a) Evaluate the fitness value $f_{i,j}^{(k)}$ of the chromosome $\overline{\Theta_{i,j}}$ for class k.
       (b) IF $(f_{i,j}^{(k)} > \overline{f_k}) \wedge (m_k < Q_M)$, THEN
           i. The chromosome $\overline{\Theta_{i,j}}$ is migrated to the subpopulation P_k.
           ii. Set $m_k = m_k + 1$.

3.2.6 Crossover

In each subpopulation, a pair of chromosomes are selected from the survived ones after chromosome elimination. The migrated chromosomes are not selected for crossover and mutation. Suppose two parent chromosomes $\Theta_{i,u} = (\Omega_{i,u}, \alpha_{\mathbf{i,u}})$ and $\Theta_{i,v} = (\Omega_{i,v}, \alpha_{\mathbf{i,v}})$ are selected to undergo crossover. The elements of $\Omega_{i,u}$ and $\Omega_{i,v}$ are exchanged using two-point crossover. On the other hand, the elements of $\alpha_{\mathbf{i,u}}$ and $\alpha_{\mathbf{i,v}}$ are swapped using one-point crossover. The following outlines the steps when two

parent chromosomes $\Theta_{i,u}$ and $\Theta_{i,v}$ are selected to exchange their genes.

1. Let $\mathbf{p_u} = (p_{u,1}, p_{u,2}..., p_{u,H(d+1)})$, where $p_{u,k} = w_{i,u,\lfloor \frac{k-1}{d+1} \rfloor + 1,(k-1) \bmod (d+1)+1}$, $k = 1, 2..., H(d+1)$.
2. Let $\mathbf{p_v} = (p_{v,1}, p_{v,2}..., p_{v,H(d+1)})$, where $p_{v,k} = w_{i,v,\lfloor \frac{k-1}{d+1} \rfloor + 1,(k-1) \bmod (d+1)+1}$, $k = 1, 2..., H(d+1)$.
3. Generate two random integers x and y such that $1 \leq x < y \leq H(d+1) - 1$.
4. Let $\mathbf{q_u} = (q_{u,1}, q_{u,2}..., q_{u,H(d+1)})$ and $\mathbf{q_v} = (q_{v,1}, q_{v,2}..., q_{v,H(d+1)})$, where

$$q_{u,k} = \begin{cases} p_{u,k} & \text{if } k = 1, 2..., x, \\ p_{v,k} & \text{if } k = x+1, x+2..., y, \\ p_{u,k} & \text{if } k = y+1, y+2..., H(d+1), \end{cases} \tag{3.12}$$

and

$$q_{v,k} = \begin{cases} p_{v,k} & \text{if } k = 1, 2..., x, \\ p_{u,k} & \text{if } k = x+1, x+2..., y, \\ p_{v,k} & \text{if } k = y+1, y+2..., H(d+1), \end{cases} \tag{3.13}$$

5. Generate a random integer x' such that $1 \leq x' \leq d' - 1$.
6. Let $\beta_{\mathbf{i,u}} = (\beta_{i,u,1}, \beta_{i,u,2}..., \beta_{i,u,d'})$ and $\beta_{\mathbf{i,v}} = (\beta_{i,v,1}, \beta_{i,v,2}..., \beta_{i,v,d'})$, where

$$\beta_{i,u,k} = \begin{cases} \alpha_{i,u,k} & \text{if } k = 1, 2..., x', \\ \alpha_{i,v,k} & \text{if } k = x'+1, x'+2..., d', \end{cases} \tag{3.14}$$

and

$$\beta_{i,v,k} = \begin{cases} \alpha_{i,v,k} & \text{if } k = 1, 2..., x', \\ \alpha_{i,u,k} & \text{if } k = x'+1, x'+2..., d', \end{cases} \tag{3.15}$$

7. Set the offspring chromosomes $\Theta'_{i,u} = (\Omega'_{i,u}, \beta_{\mathbf{i,u}})$ and $\Theta'_{i,v} = (\Omega'_{i,v}, \beta_{\mathbf{i,v}})$ such that
$\Omega'_{i,u} = \{\omega'_{\mathbf{i,u,1}}, \omega'_{\mathbf{i,u,2}} \cdots, \omega'_{\mathbf{i,u,H}}\}$ and
$\Omega'_{i,v} = \{\omega'_{\mathbf{i,v,1}}, \omega'_{\mathbf{i,v,2}} \cdots, \omega'_{\mathbf{i,v,H}}\}$, where
$\omega'_{\mathbf{i,u,k}} = (q_{u,(k-1)(d+1)+1}, q_{u,(k-1)(d+1)+2} \cdots, q_{u,k(d+1)})$ and
$\omega'_{\mathbf{i,v,k}} = (q_{v,(k-1)(d+1)+1}, q_{v,(k-1)(d+1)+2} \cdots, q_{v,k(d+1)})$, $k = 1, 2..., H$.

Suppose Q_R is the maximum number of chromosomes produced by rule migration and crossover. The above processes are repeated until the total number of chromosomes produced by rule migration and the crossover operator equals $2\lfloor \frac{Q_R}{2} \rfloor$.

3.2.7 Mutation

In each subpopulation, one of the parent chromosomes is selected and replicated first. Suppose the parent chromosome $\Theta_{i,j} = (\Omega_{i,j}, \alpha_{\mathbf{i,j}})$, $i \in \{1, 2..., C\}$, $j \in \{1, 2..., Q_P\}$, is selected to undergo mutation, each element of $\Omega_{i,j}$ and $\alpha_{\mathbf{i,j}}$ is modified by uniform mutation with a probability p_m.

When the value of $w_{i,j,k,m}$, $k \in \{1, 2..., H\}$, $m \in \{1, 2..., d\}$, is selected to mutate, it is replaced by a random real number within the range $[-1, 1]$. When the value of $w_{i,j,k,d+1}$ is selected to mutate, it is replaced by a random real number within the range $[d_1, d_2]$, where the values of d_1 and d_2 are calculated using (3.4) and (3.5) respectively. Moreover, the values of $w_{i,j,k,1}$, $w_{i,j,k,2}$ ..., $w_{i,j,k,d+1}$ are normalized so that the condition (3.6) is satisfied. When the value of $\alpha_{i,j,k}$, $k \in \{1, 2..., d'\}$, is selected to mutate, it is replaced by a random integer in the set $\{0, 1..., h_k\}$, where h_k is the number of possible values of the k^{th} nominal attribute.

The above processes are repeated until each subpopulation fills up with chromosomes.

3.2.8 Calculating the Number of Correctly Classified Training Samples in a Rule Set

A population of chromosomes are maintained and evolved in the GA-based CPRLS. It is necessary to calculate the number of correctly classified training samples for the population of chromosomes because a population having the set of best chromosomes does not necessarily have the best set of chromosomes for classifications. In each subpopulation, only the Q_P strongest chromosomes are used to calculate the number of correctly classified samples and to construct the set of rules for classifications.

It is possible that a pair of chromosomes from subpopulations P_i and P_j, $i \neq j$, classify the same sample because the sample satisfies the conditions specified by both of the chromosomes, even though they are not identical. This causes inconsistency of the set of classification rules derived from the population. To solve the problem of inconsistency in the GA-based CPRLS, the sample is classified as unknown.

It is possible that a sample does not satisfy the condition specified by each chromosome. In this case, the sample is also classified as unknown because there is no suitable rule to classify the sample.

3.3 Performance Evaluation

The performance of the GA-based CPRLS is evaluated in terms of validation accuracy and execution time in this section. Four sets of experiments were performed. In the first set of experiments, the performance of the GA-based CPRLS is compared with that of various data mining algorithms. The second set of experiments compares the performance of the GA-based CPRLS and that of Random Search-based Convex Polytope Rule Learning System (RS-based CPRLS). The third set of experiments

investigate the effects of token competition in the GA-based CPRLS. In the last set of experiments, the effects of rule migration are studied.

All the experiments were executed on a dual Intel Xeon 2.2GHz machine.

3.3.1 Performance Comparison of the GA-based CPRLS and Various Supervised Classification Algorithms

In this subsection, the performance of the GA-based CPRLS is compared with that of various data mining algorithms, including C4.5 [44], OC1 [38], NDT [26], OC1-GA, OC1-ES [8], BTGA [9] and SCION. C4.5 is a univariate decision tree algorithm [44]. OC1 constructs oblique decision trees [38], [7]. A quadratic decision tree can be constructed by NDT [26]. In OC1-GA and BTGA, GAs are employed to search for the optimal hyperplane at each non-terminal node of oblique decision trees [8], [9]. In OC1-ES, a (1+1) evolution strategy with self-adaptive mutations is applied to find the optimal hyperplane at each non-leaf node of oblique decision trees [8]. Six artificial datasets are chosen to compare the performance of the GA-based CPRLS with that of the others.

The first dataset, called ADS1, is an artificial dataset with 1000 samples. ADS1 is a two-class problem. Two straight lines are used to separate the samples into two classes. Each sample is a two-dimensional vector (x_1, x_2), where $x_1 \in [0, 1000]$ and $x_2 \in [0, 2000]$. A sample is labeled as class 1 if one of the following conditions is satisfied:

$$2x_1 + x_2 \geq 2000 \wedge 2x_1 \leq x_2 \tag{3.16}$$

$$2x_1 + x_2 < 2000 \wedge 2x_1 > x_2 \tag{3.17}$$

Otherwise, the sample is labeled as class 2. Figure 3.1 shows the dataset ADS1.

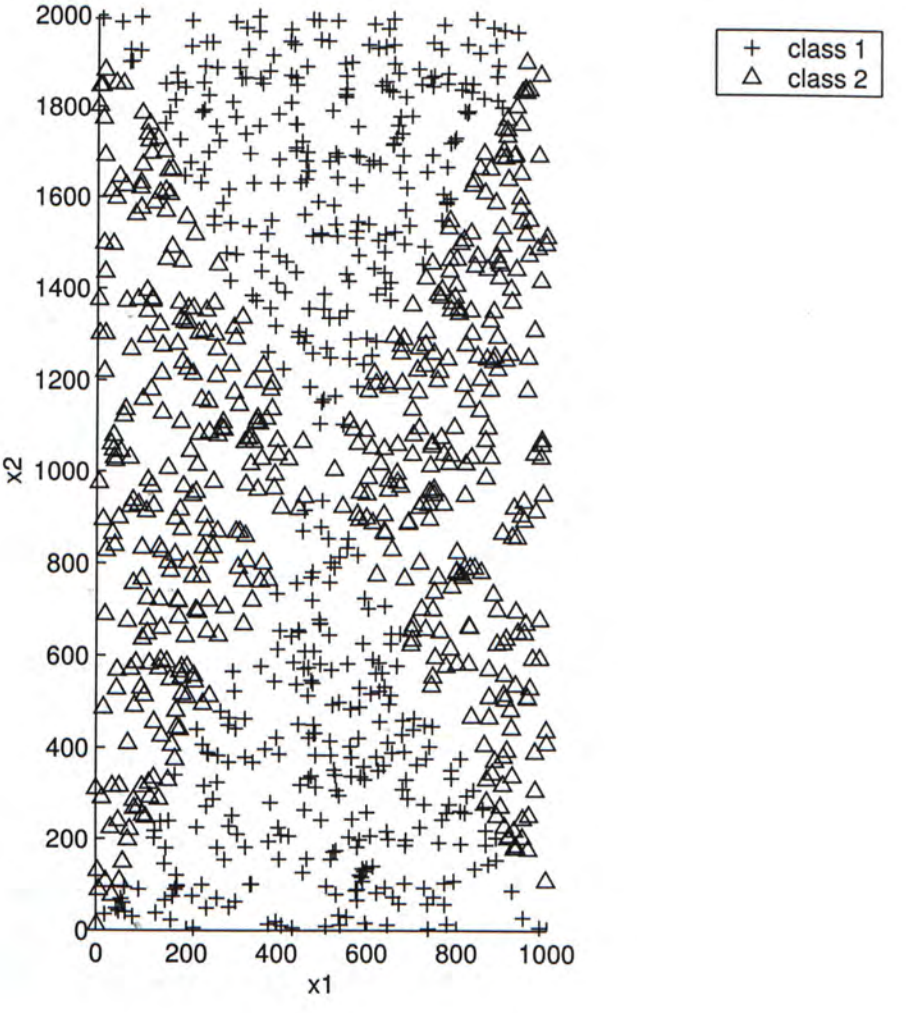

Figure 3.1: The Dataset ADS1

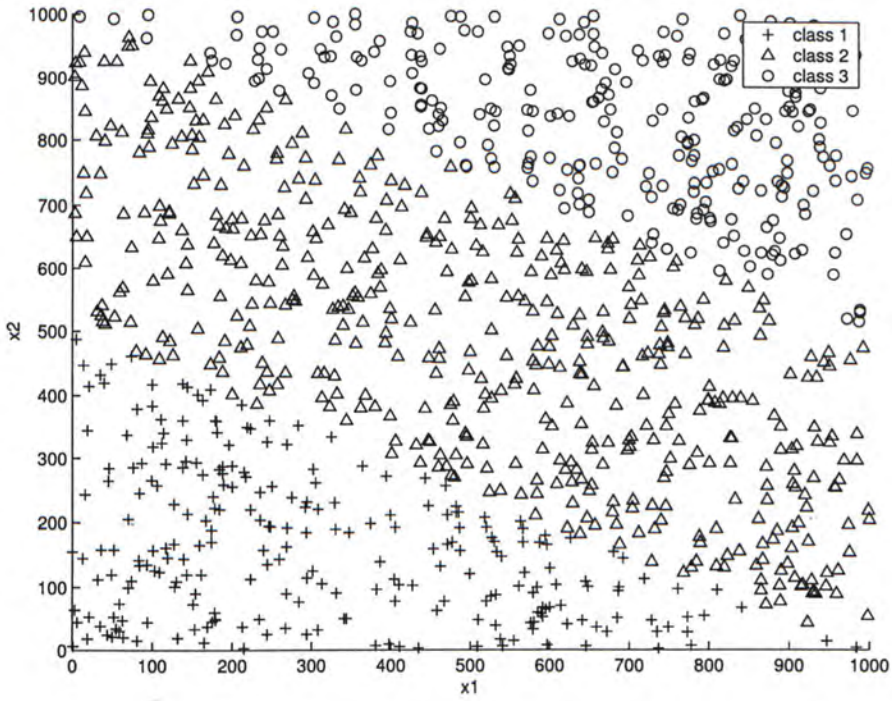

Figure 3.2: The Dataset ADS2

The second dataset, called ADS2, is also an artificial dataset with 1000 samples. Two parallel straight lines are used to separate the samples into three classes. Each sample is a two-dimensional vector (x_1, x_2), where $x_1, x_2 \in [0, 1000]$. A sample is labeled as class 1 if (3.18) is satisfied. If (3.18) is violated but (3.19) is satisfied, the sample is labeled as class 2. If neither (3.18) nor (3.19) is satisfied, the sample is classified as class 3. Figure 3.2 shows the dataset ADS2.

$$x_1 + 2x_2 < 1000 \tag{3.18}$$

$$x_1 + 2x_2 < 2000 \tag{3.19}$$

The third dataset, called ADS3, is an artificial dataset with 1000 samples. ADS3 is a two-class problem. Each sample is a two-dimensional vector (x_1, x_2), where $x_1 \in [0, 5000]$ and $x_2 \in$

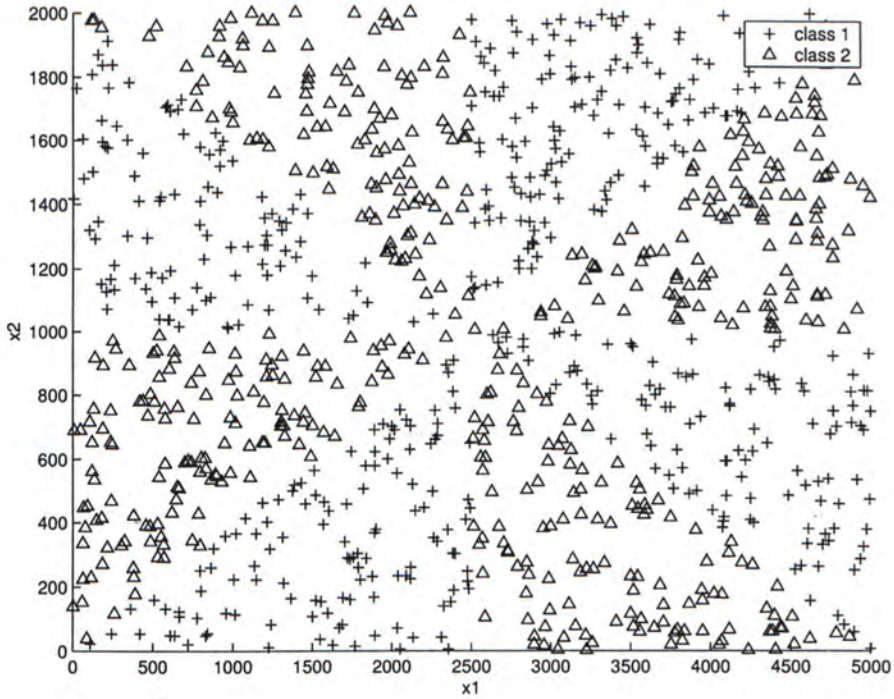

Figure 3.3: The Dataset ADS3

$[0, 2000]$. A sample is labeled as class 1 if one of the following conditions is satisfied:

$$5x_2 > 2x_1 \wedge x_1 > 2500 \tag{3.20}$$

$$2x_1 + 5x_2 \leq 10000 \wedge x_2 > 1000 \tag{3.21}$$

$$5x_2 \leq 2x_1 \wedge x_1 \leq 2500 \tag{3.22}$$

$$2x_1 + 5x_2 > 10000 \wedge x_2 \leq 1000 \tag{3.23}$$

Otherwise, the sample is labeled as class 2. Figure 3.3 shows the dataset ADS3.

The fourth dataset, called ADS4, is an artificial dataset with 1000 samples. ADS4 is a four-class problem. Each sample is a two-dimensional vector (x_1, x_2), where $x_1 \in [0, 1000]$ and $x_2 \in [0, 2000]$. The samples are labeled according to the following rules

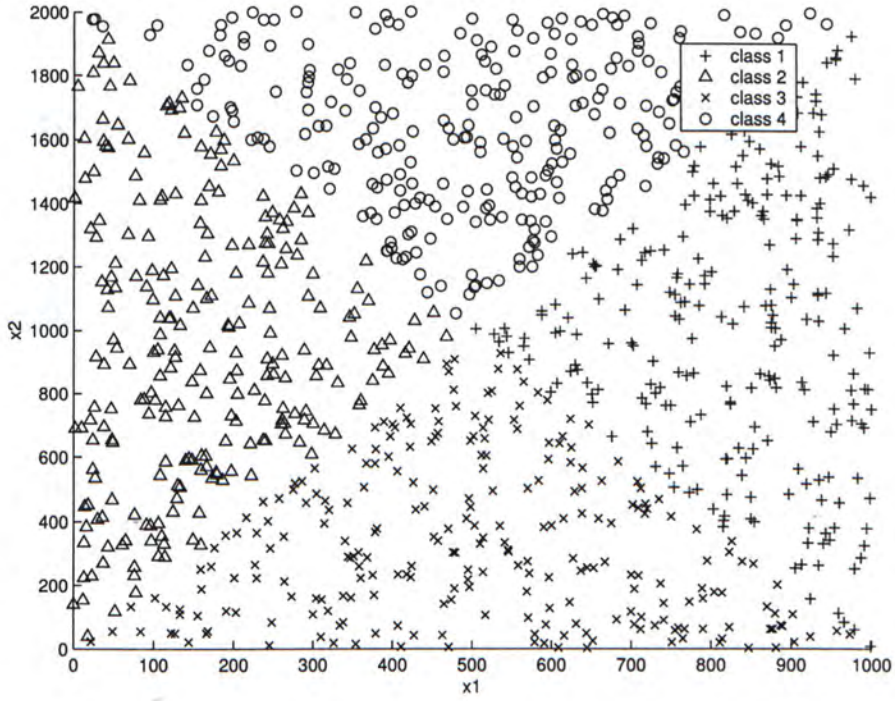

Figure 3.4: The Dataset ADS4

- IF $2x_1 + x_2 > 2000 \wedge x_2 \leq 2x_1$, THEN $class = 1$.
- IF $2x_1 + x_2 \leq 2000 \wedge x_2 > 2x_1$, THEN $class = 2$.
- IF $2x_1 + x_2 \leq 2000 \wedge x_2 \leq 2x_1$, THEN $class = 3$.
- IF $2x_1 + x_2 > 2000 \wedge x_2 > 2x_1$, THEN $class = 4$.

Figure 3.4 shows the dataset ADS4.

The fifth dataset, called ADS5, is an artificial dataset with 1000 samples. ADS5 is a four-class problem. Each sample has two numeric attributes x_1, x_2 and one nominal attribute x_3, where $x_1 \in [0, 1000]$, $x_2 \in [0, 2000]$ and $x_3 \in \{red, green\}$. The samples are labeled according to the following rules:

- IF $2x_1 + x_2 > 2000 \wedge x_2 \leq 2x_1 \wedge x_3 = red$, THEN $class = 1$.

- IF $2x_1 + x_2 > 2000 \wedge x_2 \leq 2x_1 \wedge x_3 = green$, THEN $class = 2$.
- IF $2x_1 + x_2 \leq 2000 \wedge x_2 \leq 2x_1 \wedge x_3 = red$, THEN $class = 3$.
- IF $2x_1 + x_2 \leq 2000 \wedge x_2 \leq 2x_1 \wedge x_3 = green$, THEN $class = 4$.
- IF $2x_1 + x_2 \leq 2000 \wedge x_2 > 2x_1 \wedge x_3 = red$, THEN $class = 1$.
- IF $2x_1 + x_2 \leq 2000 \wedge x_2 > 2x_1 \wedge x_3 = green$, THEN $class = 2$.
- IF $2x_1 + x_2 > 2000 \wedge x_2 > 2x_1 \wedge x_3 = red$, THEN $class = 3$.
- IF $2x_1 + x_2 > 2000 \wedge x_2 > 2x_1 \wedge x_3 = green$, THEN $class = 4$.

The sixth dataset, called ADS6, is an artificial dataset with 1000 samples. ADS6 is a four-class problem. Each sample has two numeric attributes x_1, x_2 and one nominal attribute x_3, where $x_1 \in [0, 1000]$, $x_2 \in [0, 2000]$ and $x_3 \in \{apple, orange, banana, grape\}$. The samples are labeled according to the following rules:

- IF $2x_1 + x_2 > 2000 \wedge x_2 \leq 2x_1 \wedge x_3 = apple$, THEN $class = 1$.
- IF $2x_1 + x_2 > 2000 \wedge x_2 \leq 2x_1 \wedge x_3 = orange$, THEN $class = 2$.
- IF $2x_1 + x_2 > 2000 \wedge x_2 \leq 2x_1 \wedge x_3 = banana$, THEN $class = 3$.
- IF $2x_1 + x_2 > 2000 \wedge x_2 \leq 2x_1 \wedge x_3 = grape$, THEN $class = 4$.
- IF $2x_1 + x_2 \leq 2000 \wedge x_2 \leq 2x_1 \wedge x_3 = apple$, THEN $class = 3$.
- IF $2x_1 + x_2 \leq 2000 \wedge x_2 \leq 2x_1 \wedge x_3 = orange$, THEN $class = 4$.

- IF $2x_1 + x_2 \leq 2000 \wedge x_2 \leq 2x_1 \wedge x_3 = banana$, THEN $class = 1$.
- IF $2x_1 + x_2 \leq 2000 \wedge x_2 \leq 2x_1 \wedge x_3 = grape$, THEN $class = 2$.
- IF $2x_1 + x_2 \leq 2000 \wedge x_2 > 2x_1 \wedge x_3 = apple$, THEN $class = 1$.
- IF $2x_1 + x_2 \leq 2000 \wedge x_2 > 2x_1 \wedge x_3 = orange$, THEN $class = 2$.
- IF $2x_1 + x_2 \leq 2000 \wedge x_2 > 2x_1 \wedge x_3 = banana$, THEN $class = 3$.
- IF $2x_1 + x_2 \leq 2000 \wedge x_2 > 2x_1 \wedge x_3 = grape$, THEN $class = 4$.
- IF $2x_1 + x_2 > 2000 \wedge x_2 > 2x_1 \wedge x_3 = apple$, THEN $class = 3$.
- IF $2x_1 + x_2 > 2000 \wedge x_2 > 2x_1 \wedge x_3 = orange$, THEN $class = 4$.
- IF $2x_1 + x_2 > 2000 \wedge x_2 > 2x_1 \wedge x_3 = banana$, THEN $class = 1$.
- IF $2x_1 + x_2 > 2000 \wedge x_2 > 2x_1 \wedge x_3 = grape$, THEN $class = 2$.

A (1+1) evolution strategy with self-adaptive mutations is applied in the OC1-ES algorithm [8]. Table 3.1 shows the number of generations for the OC1-ES algorithm on ADS1, ADS2, ADS3 and ADS4. Note that OC1-ES cannot handle the datasets ADS5 and ADS6 because they have one nominal attribute.

The implementation of the OC1-GA algorithm in this set of experiments is different from that in [8]. No mutation is applied for all the experiments reported in [8], while non-uniform mutation [35] is applied for all the experiments in this subsection. The purpose of adding mutation to the OC1-GA algorithm is to

Dataset	Number of Generations
ADS1	100,000
ADS2	200,000
ADS3	110,000
ADS4	98,000

Table 3.1: Number of Generations for OC1-ES on ADS1, ADS2, ADS3 and ADS4

Parameters	ADS1	ADS2	ADS3	ADS4
Population Size	100			
Number of Generations	2000	1000	25000	22000
Crossover Probability	0.7	0.8	0.8	0.9
Mutation Probability	0.1	0.05	0.1	0.1

Table 3.2: Parameters of OC1-GA on ADS1, ADS2, ADS3 and ADS4

improve the quality of the decision trees constructed by it. Table 3.2 shows the parameters of the OC1-GA algorithm so that its validation accuracy is maximized on ADS1, ADS2, ADS3 and ADS4. Note that OC1-GA cannot handle datasets with nominal attributes, including ADS5 and ADS6.

Prepruning is employed in BTGA [9]. If the number of training samples at a node is less than a positive integer n_0 or the impurity reduction is less than a threshold g_0, no child node is created. Table 3.3 shows the parameters of the BTGA algorithm and its validation accuracy is maximized on ADS1, ADS2, ADS3 and ADS4. Note that BTGA cannot construct a tree classifier for datasets with nominal attributes.

Tables 3.4 and 3.5 show the parameters of SCION and that of the GA-based CPRLS respectively. Standard parameter settings are applied in C4.5, OC1 and NDT. C4.5 can handle both nominal and continuous attributes, while OC1 and NDT can handle continuous attributes only.

Parameters	ADS1	ADS2	ADS3	ADS4
Population Size	100			
Number of Generations	5000	1000	63000	55000
Crossover Probability	0.7	0.9	0.9	0.9
Mutation Probability	0.05	0.05	0.2	0.1
n_0	15	100	10	100
g_0	0.01	0.2	0.1	0.2

Table 3.3: Parameters of BTGA on ADS1, ADS2, ADS3 and ADS4

Parameters	ADS1	ADS2	ADS3	ADS4
Number of Chromosomes L	100			
Number of Generations G	10000		125000	110000
Parent Quota Q_P	30			
Crossover Quota Q_C	40			
Migration Quota Q_M	5	5	5	6
Mutation Probability	0.2	0.1	0.1	0.15

Table 3.4: Parameters of SCION on ADS1, ADS2, ADS3 and ADS4

Parameters	ADS1	ADS2	ADS3	ADS4	ADS5	ADS6
Number of Chromosomes L	10		20	10		12
Number of Hyperplanes H	3					
Number of Generations G	10000		100000		10000	100000
Parent Quota Q_P	3		6	3		4
Crossover Quota Q_C	4		8	4		
Migration Quota Q_M	2	1	3	2		
Mutation Probability	0.2	0.1	0.2	0.1		

Table 3.5: Parameters of GA-based CPRLS on ADS1, ADS2, ADS3, ADS4, ADS5 and ADS6

Table 3.6 shows the average and the standard deviation of the validation accuracy of various supervised classification algorithms when 10-fold cross-validation is applied over 10 runs.

According to the one-sided t-tests, the GA-based CPRLS outperforms the others on ADS1 in terms of validation accuracy at 95% confidence interval. The performance of the GA-based CPRLS is better than that of OC1, NDT, OC1-GA, OC1-ES and BTGA because impurity reduction is used to determine the decision function at each non-leaf node of a decision tree. Although the concept of impurity reduction makes these decision tree algorithms to work well in many cases, but it is not the case on ADS1.

On the other hand, OC1-GA, OC1-ES and BTGA outperform the others (including the GA-based CPRLS) on ADS2 in terms of validation accuracy at 95% confidence interval using the one-sided t-tests. Less parameters are required to specify a classifier using OC1-GA, OC1-ES and BTGA when compared with the GA-based CPRLS. For OC1-GA, OC1-ES and BTGA, the decision functions at non-leaf nodes are sufficient to model the class boundaries of ADS2.

BTGA and the GA-based CPRLS outperforms the others on ADS3 in terms of validation accuracy at 95% confidence interval, according to the one-sided t-tests. On the other hand, BTGA outperforms the others (including the GA-based CPRLS) on ADS4 in terms of validation accuracy at 95% confidence interval using the one-sided t-tests.

The GA-based CPRLS outperforms C4.5 on ADS1, ADS2, ADS3, ADS4, ADS5 and ADS6 in terms of validation accuracy at 95 % confidence interval. Although C4.5 is capable of handling both numeric and nominal attributes, the GA-based CPRLS produces a better classifier on datasets with non-axis parallel boundaries.

The GA-based CPRLS outperforms SCION on ADS1, ADS2,

Algorithm	ADS1	ADS2	ADS3	ADS4	ADS5	ADS6
C4.5	94.1 ± 0.5	95.2 ± 0.3	93.6 ± 0.6	93.3 ± 0.4	92.0 ± 0.5	88.6 ± 0.5
OC1	96.5 ± 1.2	99.1 ± 0.4	93.2 ± 0.8	98.4 ± 0.3	Cannot be determined	
NDT	96.5 ± 0.6	98.4 ± 0.3	93.1 ± 0.5	98.3 ± 0.3	Cannot be determined	
OC1-GA	95.5 ± 0.5	99.7 ± 0.2	93.0 ± 0.8	95.8 ± 0.5	Cannot be determined	
OC1-ES	95.8 ± 0.4	99.6 ± 0.3	94.8 ± 0.8	98.6 ± 0.6	Cannot be determined	
BTGA	97.7 ± 0.4	99.7 ± 0.1	96.7 ± 0.4	99.5 ± 0.2	Cannot be determined	
SCION	93.2 ± 0.4	94.5 ± 0.3	92.1 ± 0.3	92.5 ± 0.8	Cannot be determined	
GA-based CPRLS	98.8 ± 0.3	99.3 ± 0.2	97.0 ± 0.6	99.1 ± 0.2	98.1 ± 0.6	96.3 ± 0.5

Table 3.6: Average and Standard Deviation of Validation Accuracy (%) of Various Supervised Classification Algorithms on ADS1, ADS2, ADS3, ADS4, ADS5 and ADS6 based on 10 Independent Runs

ADS3 and ADS4 because the antecedent part of a classification rule may include linear inequalities involving several numeric attributes in the GA-based CPRLS. In SCION, the antecedent part of a rule is restricted to a conjunctive set of linear inequalities involving one continuous attribute only. It is more difficult for SCION to produce a better set of rules on datasets with non-axis parallel class boundaries.

Table 3.7 shows the average and the standard deviation of the execution time of various supervised classification algorithms on ADS1, ADS2 and ADS3 when 10-fold cross-validation is applied over 10 runs. Table 3.8 shows the average and the standard deviation of the execution time of various supervised classification algorithms on ADS4, ADS5 and ADS6 when 10-fold cross-validation is applied over 10 runs. The execution time of the GA-based CPRLS on ADS1 is longer than that of C4.5, OC1 and NDT. A large number of generations are required to construct a better set of rules for classifications using the GA-based CPRLS. The execution times of OC1-GA, OC1-ES and BTGA on ADS1, ADS3 and ADS4 are longer than that of the GA-based

Algorithm	ADS1	ADS2	ADS3
C4.5	< 1	< 1	< 1
OC1	20.8 ± 2.3	9.8 ± 1.9	12.4 ± 0.7
NDT	24.7 ± 2.1	16.6 ± 1.1	45.6 ± 1.8
OC1-GA	752.5 ± 11.0	125.2 ± 1.3	8841.2 ± 112.5
OC1-ES	811.2 ± 39.8	123.0 ± 2.4	8853.8 ± 189.3
BTGA	708.3 ± 6.1	44.8 ± 1.9	8765.2 ± 85.9
SCION	711.5 ± 12.8	955.8 ± 11.2	8812.5 ± 95.8
GA-based CPRLS	672.0 ± 12.4	933.7 ± 14.1	8605.2 ± 102.5

Table 3.7: Average and Standard Deviation of Execution Time (in Seconds) of Various Supervised Classification Algorithms on ADS1, ADS2 and ADS3 based on 10 Independent Runs

CPRLS to investigate whether the GA-based CPRLS is capable of constructing a better classifier in a shorter period of time.

On the other hand, the execution time of the GA-based CPRLS on ADS1, ADS2, ADS3 and ADS4 is shorter than that of SCION to investigate whether the GA-based CPRLS is capable of producing a better set of rules than SCION in a shorter period of time.

3.3.2 Performance Comparison of the GA-based CPRLS and RS-based CPRLS

In this part, the performance of the GA-based CPRLS is compared with that of the RS-based CPRLS (Random Search-based Convex Polytope Rule Learning System). In the RS-based CPRLS, $100,000$ candidate rules are randomly generated for each class. This value equals the total number of chromosomes generated in the GA-based CPRLS for the experiments in the previous subsection. Each candidate rule has a conjunctive set of H' linear inequalities.

Table 3.5 shows the parameters of the GA-based CPRLS for

Algorithm	ADS4	ADS5	ADS6
C4.5	< 1	< 1	< 1
OC1	7.3 ± 1.1	Cannot be determined	
NDT	28.1 ± 1.4	Cannot be determined	
OC1-GA	7915.2 ± 170.2	Cannot be determined	
OC1-ES	7912.3 ± 169.3	Cannot be determined	
BTGA	7902.0 ± 78.3	Cannot be determined	
SCION	7851.3 ± 152.4	Cannot be determined	
GA-based CPRLS	7681.6 ± 121.4	626.3 ± 14.0	4904.6 ± 60.4

Table 3.8: Average and Standard Deviation of Execution Time (in Seconds) of Various Supervised Classification Algorithms on ADS4, ADS5 and ADS6 based on 10 Independent Runs

the experiments in this subsection. Note that $H' = H$ on each dataset to evaluate the performance of the RS-based CPRLS.

Table 3.9 reports the average and the standard deviation of the classification accuracy of the GA-based CPRLS and the RS-based CPRLS when 10-fold cross-validation is applied over 10 runs. According to the one-sided t-tests, the GA-based CPRLS outperforms the RS-based CPRLS on ADS1, ADS2, ADS3, ADS4, ADS5 and ADS6 at 95% confidence interval. The GA-based CPRLS is usually more capable of finding a better set of rules for classifications than the RS-based CPRLS because there is a fitness value for each candidate rule to guide the search for better ones in the GA-based CPRLS. Moreover, crossover and mutation improve the search for better rules because more computational effort is allocated to potentially more promising regions of the search space.

3.3.3 Effects of Token Competition

In order to investigate the effect of token competition in the GA-based CPRLS, the validation accuracy of the best set of rules on

Dataset	GA-based CPRLS	RS-based CPRLS
ADS1	98.8 ± 0.3	93.2 ± 0.5
ADS2	99.3 ± 0.2	95.3 ± 0.3
ADS3	97.0 ± 0.6	92.8 ± 0.7
ADS4	99.1 ± 0.2	95.5 ± 0.4
ADS5	98.1 ± 0.6	93.1 ± 0.3
ADS6	96.3 ± 0.5	92.6 ± 0.4

Table 3.9: Average and Standard Deviation of Validation Accuracy (%) of the GA-based CPRLS and RS-based CPRLS on ADS1, ADS2, ADS3, ADS4, ADS5 and ADS6 based on 10 Independent Runs

ADS1 is evaluated as the number of generations increases from 100 to 10000. Table 3.5 shows the values of the other parameters of the GA-based CPRLS. In each subpopulation, only the Q_P strongest rules are considered.

Table 3.10 reports the average and the standard deviation of the validation accuracy of the best set of rules on ADS1 as the number of generations increases from 100 to 10000.

From Table 3.10, the validation accuracy of the GA-based CPRLS with token competition is much higher than that without token competition. The GA-based CPRLS with token competition is capable of constructing a better set of rules than that without token competition. Token competition is capable of removing redundant chromosomes in the GA-based CPRLS. This increases the diversity of the chromosomes in a subpopulation because a less similar set of parent chromosomes is constructed, producing a less similar set of offspring chromosomes.

3.3.4 Effects of Rule Migration

In this subsection, the effect of rule migration on the performance of the GA-based CPRLS is investigated. This can be achieved by adjusting the migration quota Q_M. Note that no

Number of Generations	Token Competition	Without Token Competition
100	61.7 ± 2.2	41.3 ± 2.2
200	70.3 ± 3.7	42.9 ± 1.9
300	75.5 ± 2.5	43.5 ± 2.1
400	78.6 ± 2.1	43.9 ± 3.0
500	81.0 ± 2.4	44.3 ± 3.4
600	83.2 ± 2.1	44.8 ± 3.2
700	84.3 ± 2.2	44.7 ± 2.5
800	85.6 ± 2.5	45.0 ± 2.7
900	86.8 ± 1.9	45.2 ± 2.9
1000	87.4 ± 2.5	45.4 ± 3.2
2000	94.4 ± 1.7	45.8 ± 3.1
3000	95.9 ± 1.1	46.6 ± 2.5
4000	96.8 ± 0.8	47.1 ± 2.6
5000	97.8 ± 0.3	47.7 ± 2.1
6000	98.1 ± 0.2	47.9 ± 2.5
7000	98.3 ± 0.3	48.6 ± 2.3
8000	98.5 ± 0.5	49.2 ± 2.5
9000	98.7 ± 0.3	49.6 ± 3.1
10000	98.8 ± 0.3	50.5 ± 3.5

Table 3.10: Average and Standard Deviation of Validation Accuracy (%) of the GA-based CPRLS with and without Token Competition on ADS1 based on 10 Independent Runs

Migration Quota Q_M	Validation Accuracy (%)
0	96.5 ± 1.0
1	97.6 ± 0.6
2	98.8 ± 0.3
3	98.2 ± 0.5
4	97.2 ± 0.8
5	95.2 ± 1.2

Table 3.11: Average and Standard Deviation of Validation Accuracy (%) of the GA-based CPRLS versus Migration Quota Q_M on ADS1 based on 10 Independent Runs

rule migration occurs when $Q_M = 0$. The experiments in this subsection investigates the effect of token competition as the migration quota Q_M varies. Table 3.5 shows the values of the other parameters of the GA-based CPRLS.

Table 3.11 reports the average and the standard deviation of the validation accuracy of the best set of rules on ADS1 as the migration quota Q_M increases from 0 to 5.

From Table 3.11, the validation accuracy of the GA-based CPRLS increases for $0 \leq Q_M < 2$ but decreases for $2 \leq Q_M \leq 5$. The GA-based CPRLS performs better with a suitable value of Q_M because a weak chromosome for a particular class may be a good chromosome for another classes. When a subpopulation accepts a good chromosome from another subpopulation where the chromosome is regarded as a weak one for its original subpopulation, the quality of the best offspring chromosome is greater than the average fitness of the parent chromosomes. However, common genetic operators including crossover and mutation do not guarantee that such a chromosome can be produced.

On the other hand, too large a value of Q_M does not improve the performance of the GA-based CPRLS because it is possible that no chromosomes can be reproduced by crossover because

too many chromosomes are migrated from another subpopulations.

3.4 Chapter Summary

In this chapter, a novel rule-learning system called GA-based CPRLS has been proposed by extending SCION. The antecedent part of a classification rule is a conjunctive set of logical expressions, which may include linear inequalities with several numeric attributes and nominal attribute-value pairs.

The algorithm to evolve a set of rules using the GA-based CPRLS has been discussed. The processes of token competition and rule migration have also been described. Moreover, the performance of the GA-based CPRLS has been compared with that of various supervised classification algorithms.

Token competition and rule migration improve the performance of the GA-based CPRLS in terms of validation accuracy. The GA-based CPRLS provides an alternative algorithm to induce a set of classification rules. Experiments show that the GA-based CPRLS provides a better set of rules than SCION on datasets with non-axis parallel class boundaries.

□ **End of chapter.**

Chapter 4

Genetic Algorithm-based Quadratic Decision Trees

4.1 Introduction

In this chapter, a novel multivariate decision tree algorithm, called Genetic Algorithm-based Quadratic Decision Tree (GA-based QDT) [40], is proposed. At each non-leaf node of a GA-based QDT, the decision criterion is of the form:

$$\mathbf{x}^{\mathbf{T}} A\mathbf{x} + \mathbf{b}^{\mathbf{T}}\mathbf{x} > \gamma \tag{4.1}$$

where $A = (a_{j,k})$ is a symmetric matrix of order d, $\mathbf{b} = (b_1, b_2..., b_d)^T$ is a d-dimensional column vector, and γ is a real constant. The decision criterion in (4.1) is equivalent to a quadratic hypersurface in a d-dimensional attribute space. GAs are employed to find the optimal quadratic hypersurface to partition a set of training samples into two disjoint subsets. At each leaf node of a GA-based QDT, there is a class label to classify an input sample arriving at that node. Figure 4.1 shows an example GA-based QDT. In this example, an input sample $\mathbf{x} = (6, 0.5)^T$ is classified as class 2 because $6^2 + 0.5 \leq 38$ and $6 - 2 \times 0.5^2 > 5$.

Although the proposed algorithm extends the original work of Chai *et al.* [9], there are some major differences between Binary Tree Genetic Algorithm (BTGA) and the GA-based QDT.

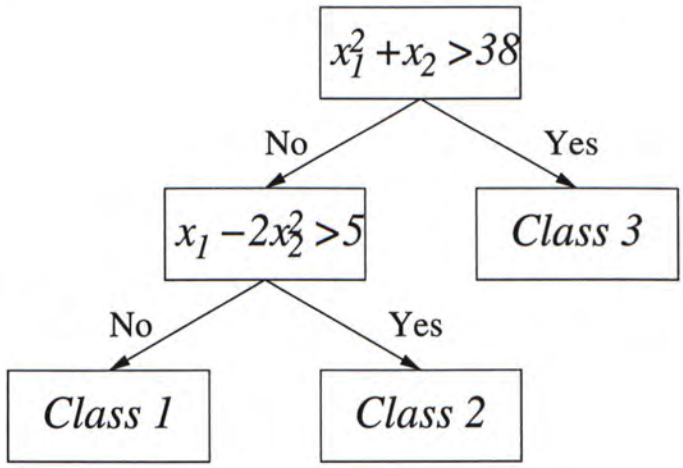

Figure 4.1: An Example GA-based QDT

Firstly, the decision criterion at each non-leaf node of BTGA is equivalent to a hyperplane while that of the GA-based QDT is equivalent to a quadratic hypersurface in the attribute space. Secondly, the coefficients of a hyperplane are encoded by a finite binary string in BTGA. In the GA-based QDT, the coefficients of a quadratic hypersurface are encoded by a vector of real numbers. The number of coefficients required to represent a quadratic hypersurface is $(d+1)(d+2)/2$ (including the constant term γ in (4.1)), therefore the search space is continuous and high-dimensional. It is not suitable to encode a solution to a high-dimensional problem as a finite binary string. Thirdly, BTGA uses linear normalization technique [11] to assign the fitness value of each chromosome, while an absolute fitness value is assigned to each chromosome in the GA-based QDT. Experimental results show that linear normalization technique is less capable of finding better quadratic hypersurfaces.

4.2 Construction of Quadratic Decision Trees

Before constructing a decision tree, a set of input samples is divided into two disjoint subsets, called training set and testing set. The training set is applied to construct a decision tree. A constructed decision tree should minimize the number of misclassifications on the testing set, instead of the training set.

In order to construct a quadratic decision tree using the training set, the root node is created first. Descendants of the root node may be created if necessary. When a new node N_h is created, the GA-based QDT searches for the optimal quadratic hypersurface if:

- the impurity of the set S_h of training samples arriving at the node N_h is not less than a threshold g_0; and
- $|S_h|$ is not less than a positive integer n_0.

Otherwise, the node N_h is declared as a leaf node and the associated class label is the class with the maximum number of training samples arriving at that node.

In this chapter, the impurity of a set of samples is measured by the Gini-index. The impurity of S_h is defined as:

$$g_1 = 1 - \sum_{i=1}^{C} (\frac{|S_{h,i}|}{|S_h|})^2 \tag{4.2}$$

where C is the number of classes and $S_{h,i}$, $i = 1, 2..., C$, is the set of training samples of class i arriving at node N_h. Suppose R_h is the set of training samples arriving at node N_h such that (4.1) is satisfied and $L_h = S_h \setminus R_h$, the weighted average impurity of the subsets L_h and R_h is defined as:

$$g_2 = \frac{|R_h|}{|S_h|}(1 - \sum_{i=1}^{C} (\frac{|R_{h,i}|}{|R_h|})^2) + \frac{|L_h|}{|S_h|}(1 - \sum_{i=1}^{C} (\frac{|L_{h,i}|}{|L_h|})^2) \tag{4.3}$$

where $R_{h,i}$, $i = 1, 2..., C$, is the set of training samples of class i arriving at node N_h such that (4.1) is satisfied and $L_{h,i} = S_{h,i} \setminus R_{h,i}$, $i = 1, 2..., C$. The impurity reduction after partitioning the set S_h into two disjoint subsets L_h and R_h is defined as:

$$g' = g_1 - g_2. \tag{4.4}$$

Note that $g' \in [0, 1]$ for all cases.

When a new node N_h is created, the GA-based QDT searches for the optimal quadratic hypersurface, maximizing the impurity reduction after splitting the set S_h into two disjoint subsets. The algorithm to find the optimal quadratic hypersurface is described in the next subsection. If the impurity reduction after partitioning the set S_h into two disjoint subsets is less than the threshold g_0, the node N_h is declared as a leaf node and the associated class label is the class with the maximum number of training samples arriving at that node. Otherwise, the optimal quadratic hypersurface is applied to partition the set S_h into two disjoint subsets. A child node is created for each subset.

Figure 4.2 shows the algorithm of the procedure `createQDT()`. The procedure `createQDT()` outlines the steps to create a new node and its descendants of a GA-based QDT. To construct a GA-based QDT, the procedure `createQDT()` accepts the training set as the parameter.

4.3 Evolving the Optimal Quadratic Hypersurface using Genetic Algorithms

At each non-leaf node of a GA-based QDT, there is an associated quadratic hypersurface for partitioning the training samples arriving at that node into two disjoint subsets. In this section, D is defined as the number of terms required to represent a quadratic hypersurface when each sample has d input attributes. From

PROCEDURE `createQDT`

INPUT A set of training samples S_h

OUTPUT A new node N_h

1. IF $|S_h|$ is less than a positive integer n_0 or the impurity of S_h is less than a threshold g_0, THEN the node N_h is declared as a leaf node and go to step 6.
2. Find the optimal quadratic hypersurface $\mathbf{x}^{\mathbf{T}} A'\mathbf{x} + \mathbf{b'}^{\mathbf{T}}\mathbf{x} > \gamma'$, where $A' = (a'_{j,k})$ is a symmetric matrix of order d, $\mathbf{b'} = (b'_1, b'_2, ..., b'_d)^T$ is a d-dimensional column vector and γ' is a real constant, using GAs such that the impurity reduction after dividing the set S_h into two disjoint subsets is maximized.
3. IF the impurity reduction is less than g_0, THEN the node N_h is declared as a leaf node and go to step 6.
4. Define $R'_h = \{\mathbf{x} \in S_h | \mathbf{x}^T A'\mathbf{x} + \mathbf{b'}^T\mathbf{x} > \gamma'\}$ and $L'_h = S_h \setminus R'_h$.
5. Invoke `createQDT(`R'_h`)` and `createQDT(`L'_h`)`, and go to step 7.
6. Determine the class label associated with the node N_h.
7. Return the node N_h.

Figure 4.2: The Algorithm of the Procedure `createQDT()`

(4.1), the relation between D and d is given by:

$$D = \frac{d(d+1)}{2} + d + 1 = \frac{(d+1)(d+2)}{2}, \tag{4.5}$$

since $\frac{d(d+1)}{2}$ terms are required to represent a symmetric matrix of order d. For example, 10 terms are required to represent a quadratic hypersurface in a 3-dimensional attribute space. When a new node N_h is created, a GA is applied to evolve the optimal quadratic hypersurface if the impurity of the set S_h of training samples arriving at that node is not less than a threshold g_0 and $|S_h|$ is not less than a positive integer n_0. The following outlines the steps to evolve the optimal quadratic hypersurface using a GA:

1. Initialize a population of chromosomes $P = \{\theta_1, \theta_2 ..., \theta_L\}$, where L is the population size.
2. Evaluate the fitness values of all chromosomes in P.
3. Let $\theta_{\mathbf{best}}$ be the best chromosome in the population P, T be the number of generations, τ be the current generation number.
4. FOR $\tau = 1$ TO T
   (a) Select L chromosomes from P (with replacement) using the roulette wheel selection method. The selected chromosomes are replicated to the mating pool M.
   (b) Set M = Crossover(M).
   (c) Set M = Mutation(M).
   (d) Evaluate the fitness values of all chromosomes in M.
   (e) Let $\theta_{\mathbf{worst}}$ be the worst chromosome in M.
   (f) Set $P = M \setminus \{\theta_{\mathbf{worst}}\} \cup \{\theta_{\mathbf{best}}\}$.
   (g) Let $\theta_{\mathbf{best}}$ be the best chromosome in P.

5. The chromosome $\theta_{\mathbf{best}}$ is chosen to divide the set S_h of training samples arriving at node N_h into two disjoint subsets, provided that the fitness value of the chromosome is not less than the threshold g_0.

An elitist strategy is employed to ensure the best chromosome in the current generation is preserved in the next generation.

4.3.1 Population Initialization

Given a population of L chromosomes $P = \{\theta_1, \theta_2..., \theta_L\}$. The chromosome θ_i, $i = 1, 2..., L$, represents the following quadratic hypersurface:

$$\mathbf{x}^T A_i \mathbf{x} + \mathbf{b}_i^T \mathbf{x} > \gamma_i, \tag{4.6}$$

where $A_i = (a_{i,j,k})$, $i = 1, 2..., L$, is a symmetric matrix of order d, $\mathbf{b_i} = (b_{i,1}, b_{i,2}..., b_{i,d})^T$, $i = 1, 2..., L$, is a d-dimensional column vector and γ_i, $i = 1, 2..., L$, is a real constant. The chromosome $\theta_i = (w_{i,1}, w_{i,2}..., w_{i,D})$, $i = 1, 2..., L$, is encoded such that:

$$w_{i,j} = \begin{cases} a_{i,j,j} & \text{if } j = 1, 2..., d, \\ 2a_{i,1,j-d+1} & \text{if } j = d+1, d+2..., 2d-1, \\ 2a_{i,2,j-(2d-1)+2} & \text{if } j = 2d, 2d+1..., 3d-3, \\ \vdots & \\ 2a_{i,d-1,d} & \text{if } j = d(d+1)/2, \\ b_{i,j-d(d+1)/2} & \text{if } j = d(d+1)/2+1..., d(d+3)/2, \\ \gamma_i & \text{if } j = D. \end{cases} \tag{4.7}$$

The value of $w_{i,j}$, $i = 1, 2..., L$, $j = 1, 2..., D-1$, is initialized with a uniform random number within the range [-1, 1]. The following outlines the steps to initialize the value of $w_{i,D}$, $i = 1, 2..., L$:

1. Two training samples $\mathbf{x} = (x_1, x_2..., x_d)^T$ and $\mathbf{y} = (y_1, y_2..., y_d)^T$ belonging to different classes are randomly chosen.

2. Set $\mathbf{z} = r\mathbf{x} + (1-r)\mathbf{y}$ where $r \in [0,1]$ is a uniform random number.

3. Set $w_{i,D} = \mathbf{z}^{\mathrm{T}} A_i \mathbf{z} + \mathbf{b}_{\mathbf{i}}^{\mathrm{T}} \mathbf{z}$.

Each chromosome represents a candidate quadratic hypersurface which passes through a randomly generated point on the line segment joining a pair of randomly selected training samples of two different classes. Moreover, the values of $w_{i,1}$, $w_{i,2}$ ..., $w_{i,D}$, $i = 1, 2..., L$, are normalized such that the following condition is satisfied:

$$\sum_{i=1}^{D} w_{i,j}^2 = 1. \tag{4.8}$$

Each chromosome represents a candidate quadratic hypersurface in a d-dimensional attribute space. It is hoped that the quality of each chromosome is improved by selection, crossover and mutation.

4.3.2 Fitness Evaluation

The fitness value f_i of the chromosome $\theta_{\mathbf{i}}$, $i = 1, 2..., L$, equals the impurity reduction when the corresponding quadratic hypersurface is applied to divide a set of training samples into two disjoint subsets. The impurity reduction is evaluated using (4.2), (4.3) and (4.4).

4.3.3 Selection

After the fitness value of each chromosome is evaluated, L chromosomes are selected (with replacement) using the roulette wheel selection method. The selected chromosomes are replicated to the mating pool. The chromosome $\theta_{\mathbf{i}}$, $i = 1, 2..., L$, is selected with a probability p_i, where

$$p_i = \frac{f_i}{\sum_{i=1}^{L} f_i}, \tag{4.9}$$

where f_i, $i = 1, 2..., L$, is the fitness value of the chromosome $\theta_{\mathbf{i}}$. Chromosomes with higher fitness values are more likely to be replicated to the mating pool because they are more likely to generate offspring of higher quality.

4.3.4 Crossover

$L/2$ pairs of chromosomes are selected from the mating pool without replacement. After a pair of parent chromosomes are selected, they undergo crossover with a fixed probability p_c. Let $M = \{\theta'_1, \theta'_2..., \theta'_{\mathbf{L}}\}$ be the mating pool. The following describes the steps of the crossover applied to the chromosomes in the mating pool M. A new population M' of offspring chromosomes is generated.

1. Set $M' = \phi$.
2. FOR $i = 1$ TO $L/2$
   (a) A pair of chromosomes $\theta_{\mathbf{u}}$ and $\theta_{\mathbf{v}}$ are selected from the mating pool M.
   (b) Set $M = M \setminus \{\theta_{\mathbf{u}}, \theta_{\mathbf{v}}\}$.
   (c) Generate a uniform random number $r \in [0, 1]$.
   (d) IF $r < p_c$, THEN generate two uniform random numbers $r_1, r_2 \in [-0.5, 1.5]$;
   ELSE set $r_1 = r_2 = 0$.
   (e) IF $\theta_{\mathbf{u}}\theta_{\mathbf{v}}^{\mathbf{T}} < 0$, THEN set $\theta_{\mathbf{v}} = -\theta_{\mathbf{v}}$.
   (f) Generate two offspring chromosomes $\theta''_{\mathbf{2i-1}}$ and $\theta''_{\mathbf{2i}}$ such that:

$$\theta''_{\mathbf{2i-1}} = r_1\theta_{\mathbf{u}} + (1 - r_1)\theta_{\mathbf{v}} \tag{4.10}$$

$$\theta''_{\mathbf{2i}} = r_2\theta_{\mathbf{v}} + (1 - r_2)\theta_{\mathbf{u}} \tag{4.11}$$

   (g) Set $M' = M' \cup \{\theta''_{2i-1}, \theta''_{2i}\}$.

4.3.5 Mutation

After the crossover operator is applied to the chromosomes in the mating pool, the offspring chromosomes undergo mutation. When the chromosome $\theta_i = (w_{i,1}, w_{i,2}..., w_{i,D}), i = 1, 2..., L$, representing the quadratic hypersurface in (4.6) undergoes mutation, the value of $w_{i,j}$, $j = 1, 2..., D$, is modified with a fixed probability p_m. When the value of $w_{i,j}$, $j = 1, 2..., D-1$, is mutated, it is set to zero or modified by non-uniform mutation [35]. Suppose τ is the current generation number and T is the number of generations. The following describes the steps of the mutation applied to the chromosome θ_i, $i = 1, 2..., L$:

1. FOR $j = 1$ TO $D - 1$
   (a) Generate a uniform random number $r_1 \in [0, 1]$.
   (b) IF $r_1 < p_m$, THEN
      i. Generate a uniform random number $r_2 \in [0, 1]$.
      ii. IF $w_{i,j} \neq 0$ and $r_2 < p_m$, THEN set $w_{i,j} = 0$; ELSE
         A. Generate a random integer $k \in \{-1, 1\}$.
         B. Generate a uniform random number $r_3 \in [0, 1]$.
         C. Set $w_{i,j} = w_{i,j} + (k - w_{i,j})(1 - r_3^{(1-\frac{\tau}{T})})$.
      iii. IF $w_{i,j} = 0$ and $r_2 < p_m$, THEN
         A. Generate a random integer $k \in \{-1, 1\}$.
         B. Generate a uniform random number $r_3 \in [0, 1]$.
         C. Set $w_{i,j} = w_{i,j} + (k - w_{i,j})(1 - r_3^{(1-\frac{\tau}{T})})$.
2. Generate a uniform random number $r_1 \in [0, 1]$.
3. IF $r_1 < p_m$, THEN
   (a) Let $y_{h,j}$, $j = 1, 2..., d$, be the minimum value of the j^{th} input attribute in the set S_h of training samples arriving at node N_h.

(b) Let $z_{h,j}$, $j = 1, 2..., d$, be the maximum value of the j^{th} input attribute in the set S_h of training samples arriving at node N_h.

(c) Set $m_1 = \frac{1}{2}\sum_{j=1}^{D-1} w_{i,j}[\lambda_{h,j}(1 + sgn(w_{i,j})) + \rho_{h,j}(1 - sgn(w_{i,j}))]$ and $m_2 = \frac{1}{2}\sum_{j=1}^{D-1} w_{i,j}[\lambda_{h,j}(1 - sgn(w_{i,j})) + \rho_{h,j}(1 + sgn(w_{i,j}))]$, where

$$\lambda_{h,j} = \begin{cases} y_{h,j}^2 & \text{if } j = 1, 2..., d, \\ y_{h,1}y_{h,j-d+1} & \text{if } j = d+1, d+2..., 2d-1, \\ y_{h,2}y_{h,j-(2d-1)+2} & \text{if } j = 2d, 2d+1..., 3d-3, \\ \vdots & \\ y_{h,d-1}y_{h,d} & \text{if } j = d(d+1)/2, \\ y_{h,j-d(d+1)/2} & \text{if } j = d(d+1)/2+1..., d(d+3)/2, \end{cases}$$

and

$$\rho_{h,j} = \begin{cases} z_{h,j}^2 & \text{if } j = 1, 2..., d, \\ z_{h,1}z_{h,j-d+1} & \text{if } j = d+1, d+2..., 2d-1, \\ z_{h,2}z_{h,j-(2d-1)+2} & \text{if } j = 2d, 2d+1..., 3d-3, \\ \vdots & \\ z_{h,d-1}z_{h,d} & \text{if } j = d(d+1)/2, \\ z_{h,j-d(d+1)/2} & \text{if } j = d(d+1)/2+1..., d(d+3)/2. \end{cases}$$

(d) Generate a random integer $k \in \{0, 1\}$.

(e) Generate a uniform random number $r_3 \in [0, 1]$.

(f) IF $k = 0$,
THEN set $w_{i,D} = w_{i,D} + (m_2 - w_{i,D})(1 - r_3^{(1-\frac{\tau}{T})})$;
ELSE set $w_{i,D} = w_{i,D} - (w_{i,D} - m_1)(1 - r_3^{(1-\frac{\tau}{T})})$.

4.4 Performance Evaluation

In this section, the performance of the GA-based QDT is evaluated in terms of validation accuracy, number of nodes and exe-

cution time. Four sets of experiments were performed. The first set of experiments compares the performance of the GA-based QDT with that of various supervised classification algorithms. The second set of experiments compares the performance of the GA-based QDT and that of Random Search-based Quadratic Decision Tree (RS-based QDT). In the third set of experiments, the effect of changing one of the parameters of the GA-based QDT is investigated. In the last set of experiments, the effect of adding noise in a dataset on the performance of the GA-based QDT is studied. All the experiments were executed on a dual Intel Xeon 2.2GHz machine.

4.4.1 Performance Comparison of the GA-based QDT and Various Supervised Classification Algorithms

In this subsection, the performance of the GA-based QDT is compared with that of various supervised classification algorithms, including C4.5, OC1, NDT, OC1-GA, OC1-ES and BTGA. Two artificial and two public domain datasets from the UCI machine learning repository are chosen to compare the performance of the GA-based QDT with that of the others.

The first dataset, called ADS7, is an artificial dataset with 100 samples. ADS7 is a two-class problem. A straight line is used to separate the samples into two classes. Each sample is a two-dimensional vector (x_1, x_2), where $x_1, x_2 \in [0, 1000]$. A sample is labeled as class 1 if $0.8x_1 - 0.6x_2 > 150$. Otherwise, the sample is labeled as class 2. Figure 4.3 shows the dataset ADS7.

The second dataset, called ADS8, is an artificial dataset with 1000 samples. ADS8 is a three-class problem. Each sample is a two-dimensional vector (x_1, x_2), where $x_1, x_2 \in [0, 1000]$. If a sample satisfies (4.12), it is labeled as class 1. If a sample violates (4.12) but satisfies (4.13), it is labeled as class 2. If a

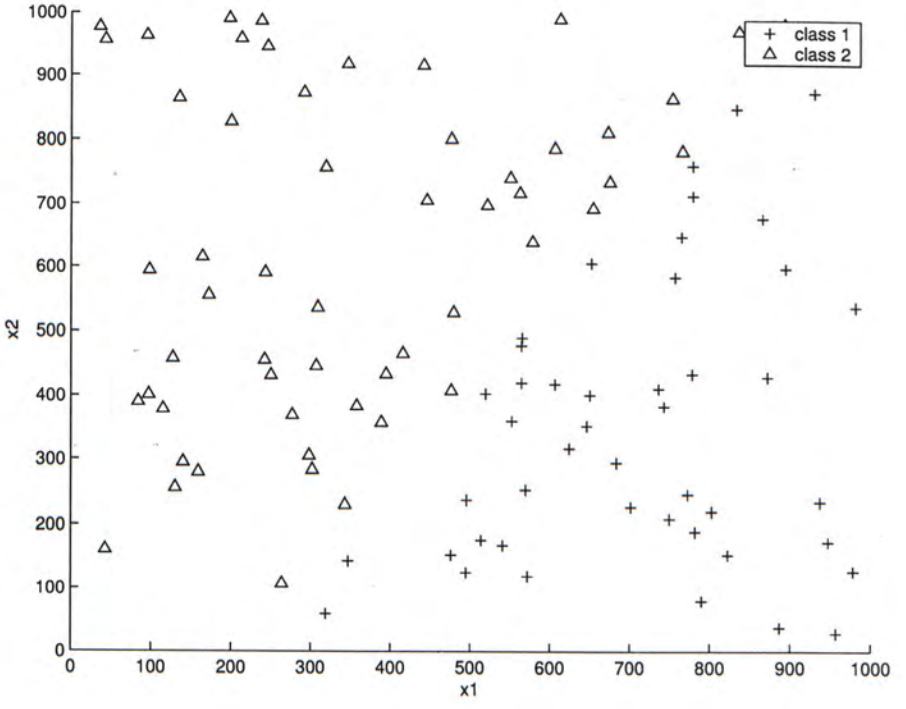

Figure 4.3: The Dataset ADS7

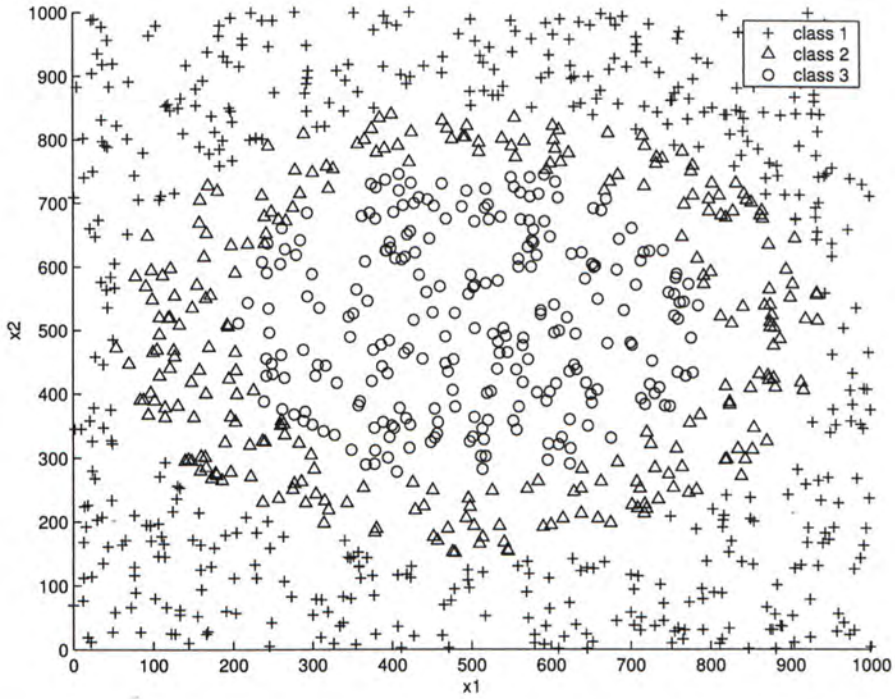

Figure 4.4: The Dataset ADS8

sample violates both (4.12) and (4.13), it is labeled as class 3. Figure 4.4 shows the dataset ADS8.

$$\frac{(x_1 - 500)^2}{202500} + \frac{(x_2 - 500)^2}{122500} > 1.0 \tag{4.12}$$

$$\frac{(x_1 - 500)^2}{90000} + \frac{(x_2 - 510)^2}{62500} > 1.0 \tag{4.13}$$

The third dataset, called ECOLI, is a public domain dataset from the UCI machine learning repository. Each sample has 7 numeric input attributes. This dataset has 336 samples and 8 classes.

The fourth dataset, called BALANCE, is also a public domain dataset from the UCI machine learning repository. Each sample has 4 numeric input attributes. This dataset has 625 samples

Dataset	Number of Generations
ADS7	300,000
ADS8	60,000
ECOLI	300,000
BALANCE	150,000

Table 4.1: Number of Generations for OC1-ES on ADS7, ADS8, ECOLI and BALANCE

Parameters	ADS7	ADS8	ECOLI	BALANCE
Population Size	100			
Number of Generations	2500	1000	4000	2000
Crossover Probability	1.0	0.8	0.9	0.9
Mutation Probability	0.25	0.3	0.3	0.15

Table 4.2: Parameters of OC1-GA on ADS7, ADS8, ECOLI and BALANCE

and 3 classes.

The number of generations for the OC1-ES algorithm is modified on each dataset so that the execution time of the OC1-ES algorithm is longer than that of the GA-based QDT. Table 4.1 shows the number of generations for the OC1-ES algorithm on each dataset.

The implementation of the OC1-GA algorithm in this set of experiments is identical to that described in Section 3.3.1. The number of generations for the OC1-GA algorithm is modified on each dataset so that the execution time of the OC1-GA algorithm is longer than that of the GA-based QDT. Table 4.2 shows the parameters of the OC1-GA algorithm so that its validation accuracy is maximized on each dataset.

If the number of training samples at a node is less than a positive integer n_0 or the impurity reduction is less than a threshold g_0, no child node is created in BTGA. The number of generations for the BTGA algorithm is modified on each dataset so that the

Parameters	ADS7	ADS8	ECOLI	BALANCE
Population Size	100			
Number of Generations	2500	1000	4000	2000
Crossover Probability	1.0	0.8	0.8	0.8
Mutation Probability	0.25	0.15	0.15	0.1
n_0	50	10	15	10
g_0	0.4	0.01	0.1	0.1

Table 4.3: Parameters of BTGA on ADS7, ADS8, ECOLI and BALANCE

Parameters	ADS7	ADS8	ECOLI	BALANCE
Population Size	100			
Number of Generations	1000			
Crossover Probability	0.9	1.0	0.9	0.9
Mutation Probability	0.15	0.15	0.1	0.1
n_0	50	100	20	30
g_0	0.3	0.3	0.15	0.1

Table 4.4: Parameters of the GA-based QDT on ADS7, ADS8, ECOLI and BALANCE

execution time of the BTGA algorithm is longer than that of the GA-based QDT. Table 4.3 shows the parameters of the BTGA algorithm so that its validation accuracy is maximized on each dataset.

Table 4.4 shows the parameters of the GA-based QDT so as to maximize its validation accuracy. The value of n_0 specifies the minimum number of training samples and that of g_0 specifies the minimum impurity reduction. Standard parameter settings are applied in various decision tree algorithms including C4.5, OC1 and NDT.

Table 4.5 shows the average and the standard deviation of the validation accuracy of various supervised classification algorithms when 10-fold cross-validation is applied over 10 runs.

According to the one-sided t-tests, the GA-based QDT outperforms the others on ADS8, ECOLI and BALANCE in terms of validation accuracy at 95% confidence interval. The decision criterion at each non-leaf node of a GA-based QDT usually provides a better approximation to non-linear class boundaries when compared with that of univariate and oblique decision tree algorithms. Although both NDT and GA-based QDT construct quadratic decision trees, the GA-based QDT outperforms NDT on all of the datasets. When a new node is created, GA-based QDT is more capable of finding a better quadratic hypersurface than NDT because GA-based QDT is more capable of escaping from local optima.

On the other hand, BTGA outperforms the others (including the GA-based QDT) on ADS7 in terms of validation accuracy at 95% confidence interval using the one-sided t-tests. Suppose each input sample has d numeric input attributes. The number of parameters required to specify a hyperplane is $O(d)$, while the number of terms required to specify a quadratic hypersurface is $O(d^2)$. Since the class boundary of ADS7 is linear, a hyperplane can be used to divide the samples into two classes completely. It is much faster to find the optimal hyperplane than the optimal quadratic hypersurface using GAs when the training samples to be partitioned are linearly separable. Moreover, a quadratic hypersurface tends to overfit the training samples when they are linearly separable.

Univariate decision tree algorithms should outperform the others (including the GA-based QDT) on datasets whose class boundaries are axis-parallel hyperplanes. Again, a quadratic hypersurface tends to overfit the training samples in such datasets. It is more suitable to use univariate decision tree algorithms to construct decision trees for such datasets than the GA-based QDT.

Table 4.6 shows the average and the standard deviation of the

Algorithm	ADS7	ADS8	ECOLI	BALANCE
C4.5	87.4 ± 2.0	93.6 ± 0.6	81.6 ± 1.2	77.6 ± 0.7
OC1	94.0 ± 1.6	93.2 ± 0.8	80.7 ± 1.8	91.1 ± 0.6
NDT	93.8 ± 1.5	93.1 ± 0.5	81.1 ± 1.8	91.8 ± 1.1
OC1-GA	93.4 ± 1.2	93.0 ± 0.5	83.6 ± 1.3	93.9 ± 1.2
OC1-ES	98.1 ± 1.3	94.8 ± 0.8	80.6 ± 2.0	90.7 ± 0.9
BTGA	99.4 ± 0.7	96.0 ± 0.3	83.6 ± 1.4	93.1 ± 1.0
GA-based QDT	98.3 ± 0.8	98.9 ± 0.3	84.9 ± 0.7	97.2 ± 0.5

Table 4.5: Average and Standard Deviation of Validation Accuracy (%) of Various Supervised Classification Algorithms on ADS7, ADS8, ECOLI and BALANCE based on 10 Independent Runs

tree size (in number of nodes) of various supervised classification algorithms when 10-fold cross-validation is applied over 10 runs. The time required to classify an input sample depends on the number of non-leaf nodes visited and the number of parameters required to specify the decision criterion at each non-leaf node of a decision tree. Although the GA-based QDT usually constructs decision trees with fewer nodes, the decision trees constructed by the GA-based QDT do not necessarily classify input samples more quickly than those constructed by the others.

Table 4.7 shows the average and the standard deviation of the execution time of various supervised classification algorithms when 10-fold cross-validation is applied over 10 runs. The execution time of the GA-based QDT is longer than that of C4.5, OC1 and NDT. The number of coefficients required to specify a quadratic hypersurface in a d-dimensional attribute space is $O(d^2)$, a sufficiently large number of generations is required to find an acceptable quadratic hypersurface using GAs. The number of generations for OC1-ES, OC1-GA and BTGA is modified so that the execution times of these algorithms are longer than that of the GA-based QDT (except for BTGA on ADS7). The

Algorithm	ADS7	ADS8	ECOLI	BALANCE
C4.5	5.7 ± 0.1	33.6 ± 0.9	18.6 ± 1.0	41.8 ± 1.1
OC1	3.1 ± 0.2	12.3 ± 3.1	4.6 ± 1.0	5.5 ± 1.2
NDT	3.2 ± 0.3	10.5 ± 2.3	17.9 ± 0.9	7.6 ± 1.5
OC1-GA	3.2 ± 0.4	16.0 ± 3.1	5.0 ± 0.9	6.1 ± 1.2
OC1-ES	3.0 ± 0.1	11.5 ± 2.1	4.9 ± 0.8	5.5 ± 1.5
BTGA	3.0 ± 0.0	25.9 ± 0.6	6.7 ± 0.3	7.0 ± 0.5
GA-based QDT	3.0 ± 0.0	5.0 ± 0.0	6.1 ± 0.1	5.1 ± 0.1

Table 4.6: Average and Standard Deviation of Tree Size (in Number of Nodes) of Various Supervised Classification Algorithms on ADS7, ADS8, ECOLI and BALANCE based on 10 Independent Runs

reason for this is to investigate whether the GA-based QDT constructs a better decision tree in a shorter period of time.

However, the execution time of BTGA on ADS7 is shorter than that of the GA-based QDT because BTGA can find a hyperplane such that the weighted average impurity of the training samples in ADS7 is zero in less than 1000 generations. In other words, the impurity reduction is maximized. In this case, BTGA finishes the construction of oblique decision trees for the dataset ADS7.

4.4.2 Performance Comparison of the GA-based QDT and RS-based QDT

In this part, the performance of the GA-based QDT is compared with that of Random Search-based Quadratic Decision Tree (RS-based QDT). At each non-leaf node of a RS-based QDT, $100,000$ candidate quadratic hypersurfaces are randomly generated using the same way as the chromosomes in the GA-based QDT are initialized. This number is equal to the number of chromosomes generated by the GA-based QDT when a new node is created. The impurity reduction of each candidate

Algorithm	ADS7	ADS8	ECOLI	BALANCE
C4.5	< 1	< 1	< 1	< 1
OC1	< 1	44.0 ± 0.8	73.7 ± 2.1	57.8 ± 0.6
NDT	1.7 ± 0.5	102.6 ± 4.5	142.0 ± 2.7	90.3 ± 2.6
OC1-GA	47.6 ± 4.6	417.9 ± 18.0	1015 ± 27	471.2 ± 30.0
OC1-ES	36.0 ± 3.9	264.0 ± 16.9	1021 ± 38	476.1 ± 12.9
BTGA	3.4 ± 2.0	717.3 ± 80.3	1254 ± 133	485.7 ± 67.3
GA-based QDT	31.5 ± 0.5	246.5 ± 21.6	959.9 ± 54.0	366.7 ± 5.1

Table 4.7: Average and Standard Deviation of Execution Time (in Seconds) of Various Supervised Classification Algorithms on ADS7, ADS8, ECOLI and BALANCE based on 10 Independent Runs

quadratic hypersurface is evaluated and the quadratic hypersurface with the maximum impurity reduction is chosen to partition a set of samples into two disjoint subsets if the minimum number of samples is not less than a positive integer n_0 and the impurity reduction is not less than a threshold g_0. In this case, a child node is created for each subset.

Table 4.4 shows the minimum number of samples n_0 and the minimum impurity reduction g_0 required to create child nodes on each dataset. Table 4.8 reports the average and the standard deviation of the validation accuracy of the GA-based QDT and the RS-based QDT when 10-fold cross-validation is applied over 10 runs. According to the one-sided t-tests, the GA-based QDT outperforms the RS-based QDT on ADS8, ECOLI and BALANCE at 95% confidence interval. The GA-based QDT is usually more capable of finding a better quadratic hypersurface than the RS-based QDT when a new node is created.

4.4.3 Effects of Changing Parameters of the GA-based QDT

There are several parameters in the GA-based QDT, including:

Dataset	GA-based QDT	RS-based QDT
ADS7	98.3 ± 0.8	98.3 ± 1.3
ADS8	98.9 ± 0.3	52.1 ± 3.8
ECOLI	84.9 ± 0.7	83.8 ± 0.3
BALANCE	97.2 ± 0.5	89.2 ± 1.0

Table 4.8: Average and Standard Deviation of Validation Accuracy (%) of the GA-based QDT and RS-based QDT on ADS7, ADS8, ECOLI and BALANCE based on 10 Independent Runs

- Crossover Probability (p_c);
- Mutation Probability (p_m);
- Number of Generations (T);
- Minimum Number of Training Samples (n_0); and
- Minimum Impurity Reduction (g_0).

The first three parameters are common parameters of GAs. The last two parameters specify the necessary and sufficient conditions of creating child nodes. Table 4.4 shows that the optimal values of these parameters (except for the number of generations) are different on each dataset. The BALANCE dataset is applied to illustrate the effect of changing one of these parameters.

Number of Generations (T)

In this part, the number of generations increases from 100 to 5000 while the values of the other parameters are shown in Table 4.4.

Table 4.9 reports the average and the standard deviation of the validation accuracy, the execution time and the tree size (in number of nodes) based on 10 independent runs when the number of generations T equals $100, 200..., 900, 1000, 2000..., 5000$.

The results reported in Table 4.9 are used to plot Figures 4.5, 4.6 and 4.7.

Although the validation accuracy tends to increase as the number of generations increases, the validation accuracy and the number of generations may not be related by an increasing function. For example, the validation accuracy for $T = 800$ is greater than that for $T = 900$. Although the quality of the quadratic hypersurface at each non-leaf node can be improved as the number of generations increases, producing better quadratic hypersurfaces during the training stage does not necessarily construct better decision trees.

On the other hand, the execution time is not directly proportional to the number of generations. The number of nodes varies as the number of generations varies. When a larger GA-based QDT is constructed, more candidate quadratic hypersurfaces are initialized, evaluated and evolved using GAs. At the root node, all the samples in the training set are considered when GAs are applied to find the optimal quadratic hypersurface. Nevertheless, a different subset of samples in the training set is considered to search for the optimal quadratic hypersurface at each non-root node of a GA-based QDT. As a result, the time required to find the optimal quadratic hypersurface is different for each non-leaf node.

Crossover Probability (p_c)

In this part, the crossover probability increases from 0.0 to 1.0 while the values of the other parameters are shown in Table 4.4.

Table 4.10 shows the average and the standard deviation of the validation accuracy, the execution time and the tree size (in number of leaf nodes) based on 10 independent runs when the crossover probability p_c equals $0.0, 0.1..., 1.0$. The results reported in Table 4.10 are used to plot Figures 4.8, 4.9 and 4.10. The maximum validation accuracy is attained when the

T	Validation Accuracy (%)	Execution Time (s)	Tree Size
100	95.06 ± 0.83	45.3 ± 5.6	5.93 ± 0.28
200	96.02 ± 0.56	90.1 ± 9.9	5.81 ± 0.17
300	96.28 ± 0.68	122.0 ± 2.3	5.67 ± 0.14
400	96.25 ± 0.85	159.8 ± 3.1	5.57 ± 0.19
500	96.34 ± 0.66	199.8 ± 4.0	5.46 ± 0.19
600	97.06 ± 0.75	233.8 ± 2.3	5.51 ± 0.17
700	96.78 ± 0.92	268.7 ± 5.5	5.55 ± 0.22
800	96.98 ± 0.44	310.5 ± 4.9	5.47 ± 0.22
900	96.73 ± 0.75	347.3 ± 3.6	5.39 ± 0.19
1000	97.23 ± 0.49	366.7 ± 5.1	5.13 ± 0.11
2000	97.13 ± 0.81	741.0 ± 7.1	5.16 ± 0.11
3000	97.08 ± 0.84	1110.5 ± 11.4	5.19 ± 0.19
4000	97.40 ± 0.61	1459.2 ± 6.1	5.17 ± 0.12
5000	97.50 ± 0.65	1885.2 ± 7.2	5.13 ± 0.09

Table 4.9: Average and Standard Deviation of Validation Accuracy (%), Execution Time (in Seconds) and Tree Size (in Number of Nodes) on BALANCE based on 10 Independent Runs as the Number of Generations T Varies

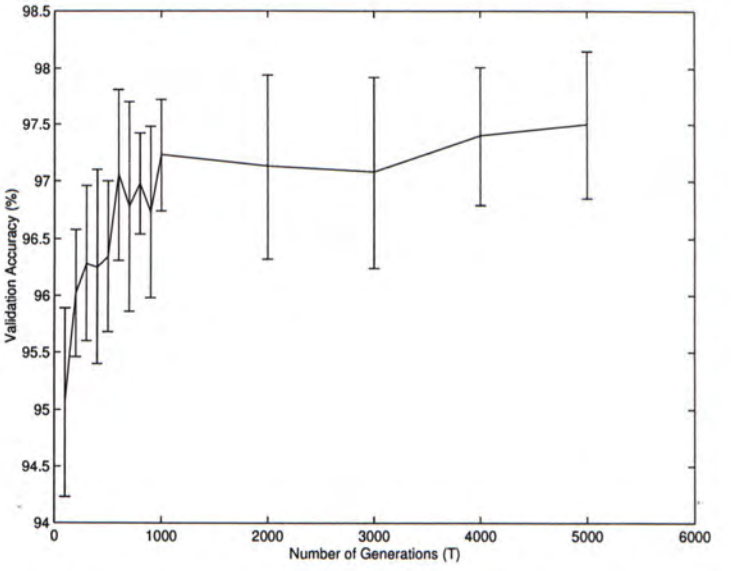

Figure 4.5: Validation Accuracy (%) of GA-based QDT on BALANCE versus Number of Generations T

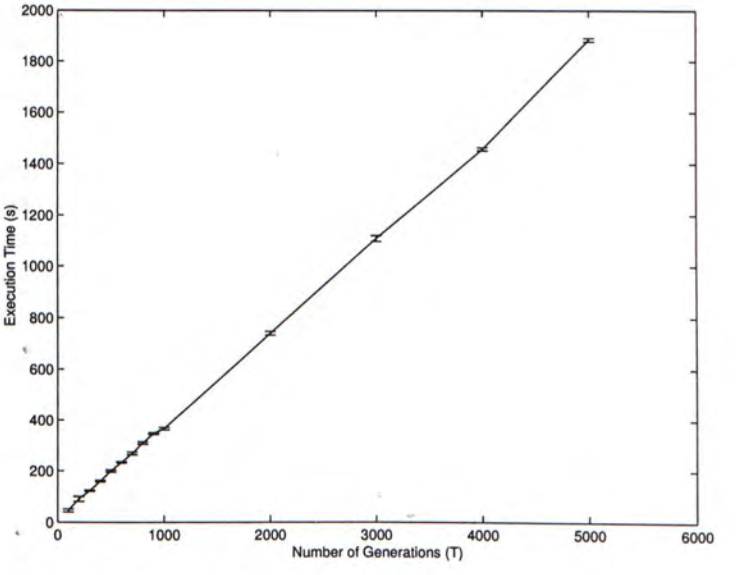

Figure 4.6: Execution Time (in Seconds) of GA-based QDT on BALANCE versus Number of Generations T

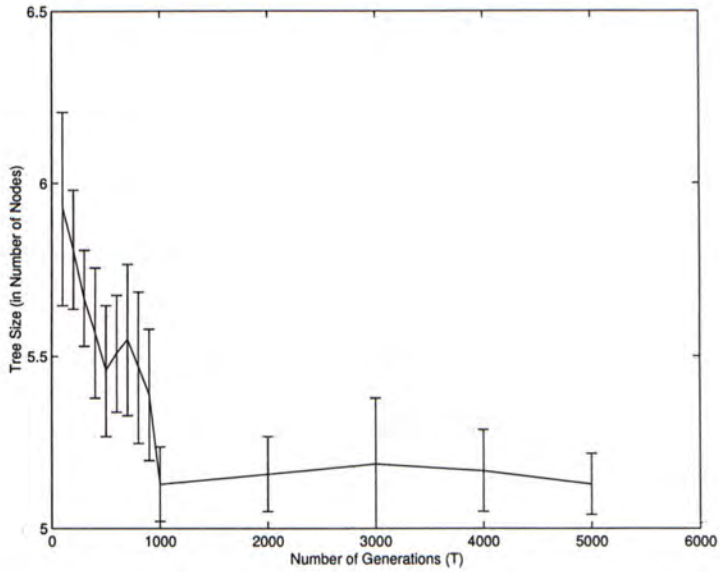

Figure 4.7: Tree Size (in Number of Nodes) of GA-based QDT on BALANCE versus Number of Generations T

crossover probability is 0.9. However, the confidence intervals of the validation accuracy for different crossover probabilities do overlap.

Mutation Probability (p_m)

In this part, the mutation probability increases from 0.0 to 1.0 while the values of the other parameters are shown in Table 4.4.

Table 4.11 reports the average and the standard deviation of the validation accuracy, the execution time and the tree size (in number of leaf nodes) based on 10 independent runs when the mutation probability p_m equals $0.0, 0.05, 0.1, ..., 0.25, 0.3, 0.4..., 1.0$. The results reported in Table 4.11 are used to plot Figures 4.11, 4.12 and 4.13.

The maximum validation accuracy is attained when the mutation probability is 0.05. The execution time tends to be longer as the mutation probability increases because non-uniform mu-

p_c	Validation Accuracy (%)	Execution Time (s)	Tree Size
0.0	96.45 ± 0.99	369.4 ± 6.6	3.30 ± 0.16
0.1	96.82 ± 0.98	364.1 ± 5.2	3.15 ± 0.12
0.2	96.80 ± 0.71	362.8 ± 5.0	3.14 ± 0.11
0.3	97.07 ± 0.67	367.8 ± 4.8	3.18 ± 0.12
0.4	96.80 ± 0.86	364.8 ± 4.0	3.18 ± 0.11
0.5	97.04 ± 1.03	367.8 ± 5.1	3.22 ± 0.10
0.6	97.20 ± 0.81	369.7 ± 6.3	3.21 ± 0.11
0.7	96.83 ± 0.86	367.5 ± 5.5	3.19 ± 0.13
0.8	97.07 ± 0.82	365.3 ± 5.7	3.11 ± 0.03
0.9	97.23 ± 0.49	366.7 ± 5.1	3.13 ± 0.11
1.0	96.50 ± 0.42	368.4 ± 8.1	3.14 ± 0.10

Table 4.10: Average and Standard Deviation of Validation Accuracy (%), Execution Time (in Seconds) and Tree Size (in Number of Leaf Nodes) on BALANCE based on 10 Independent Runs when the Crossover Probability p_c Varies

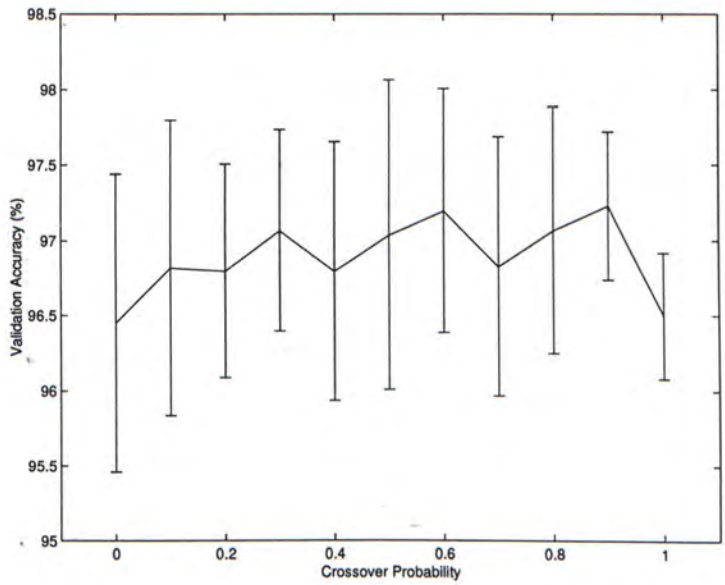

Figure 4.8: Validation Accuracy (%) of GA-based QDT on BALANCE versus Crossover Probability p_c

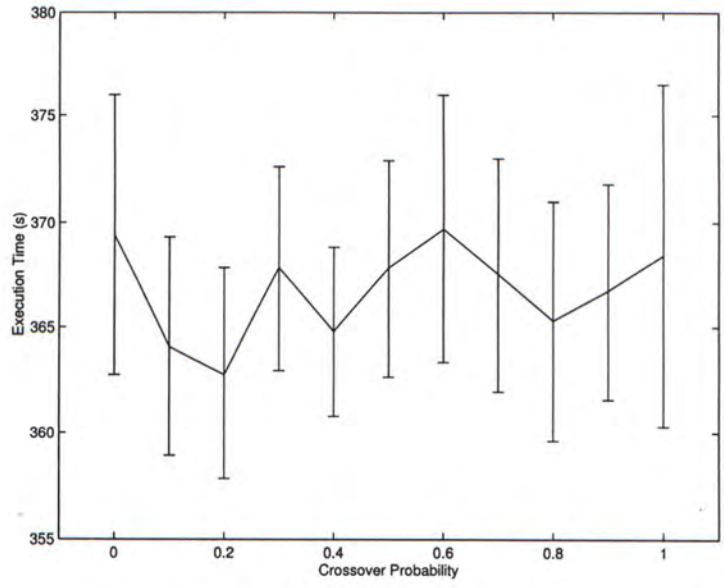

Figure 4.9: Execution Time (in Seconds) of GA-based QDT on BALANCE versus Crossover Probability p_c

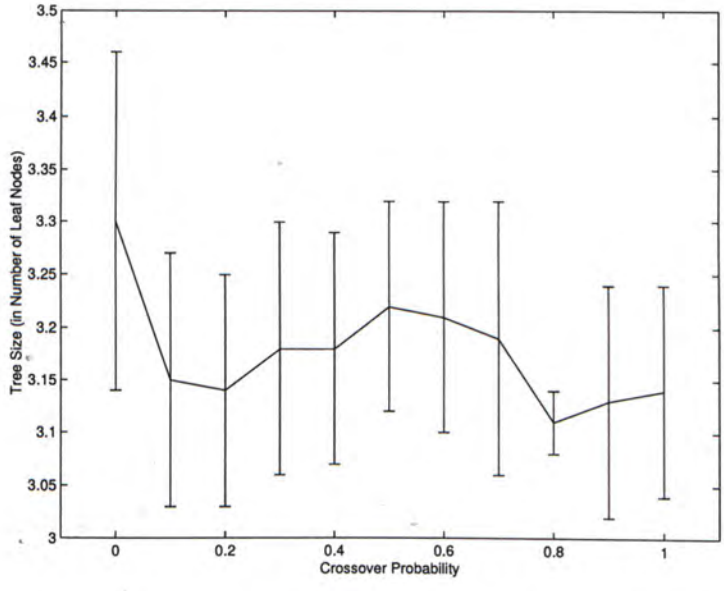

Figure 4.10: Tree Size (in Number of Leaf Nodes) of GA-based QDT on BALANCE versus Crossover Probability p_c

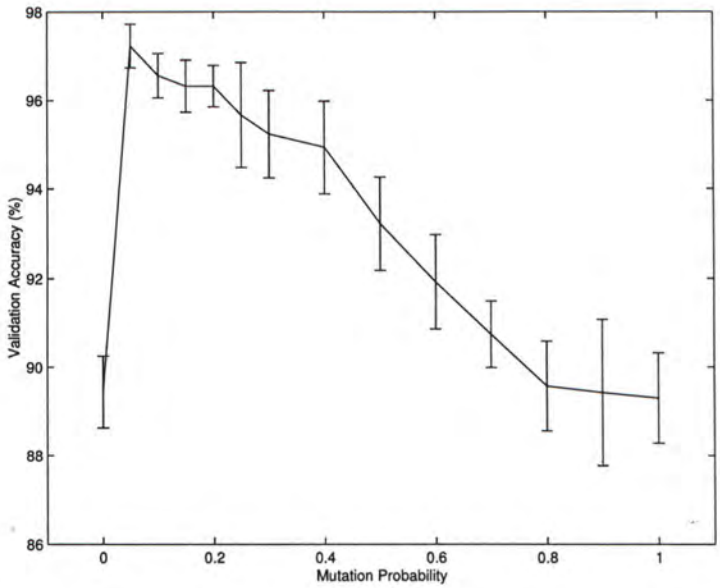

Figure 4.11: Validation Accuracy (%) of GA-based QDT on BALANCE versus Mutation Probability p_m

tation is applied to each chromosome in the GA-based QDT. The computation of $r_3^{(1-\frac{r}{T})}$ (see Section 4.3.5) is a bottleneck of non-uniform mutation. As the mutation probability increases, the expected number of mutations increases. Therefore, the execution time is longer.

The minimum number of leaf nodes is attained when the mutation probability is 0.05. When the mutation probability equals 0.05, GA-based QDT has its greatest capability of evolving the optimal quadratic decision function at each internal node on BALANCE. The impurity reduction after partioning a set of training samples for this value of mutation probability is much higher than that for other values of mutation probability. Therefore, less nodes are generated when the mutation probability is 0.05.

p_m	Validation Accuracy (%)	Execution Time (s)	Tree Size
0.0	89.44 ± 0.81	381.2 ± 5.4	4.23 ± 0.58
0.05	97.23 ± 0.49	366.7 ± 5.1	3.13 ± 0.11
0.1	96.56 ± 0.50	379.9 ± 8.4	5.54 ± 0.23
0.15	96.32 ± 0.59	386.3 ± 10.4	5.74 ± 0.23
0.2	96.32 ± 0.47	390.2 ± 10.3	5.90 ± 0.38
0.25	95.66 ± 1.19	413.8 ± 15.0	6.04 ± 0.48
0.3	95.23 ± 0.99	438.3 ± 17.7	6.12 ± 0.44
0.4	94.93 ± 1.05	439.8 ± 11.4	3.80 ± 0.30
0.5	93.22 ± 1.04	471.6 ± 20.3	3.97 ± 0.41
0.6	91.92 ± 1.06	497.7 ± 10.4	4.08 ± 0.33
0.7	90.74 ± 0.75	505.6 ± 12.7	4.28 ± 0.35
0.8	89.57 ± 1.01	514.0 ± 16.1	4.22 ± 0.55
0.9	89.42 ± 1.65	540.4 ± 15.7	4.54 ± 0.71
1.0	89.30 ± 1.02	565.8 ± 10.5	4.55 ± 0.73

Table 4.11: Average and Standard Deviation of Validation Accuracy (%), Execution Time (in Seconds) and Tree Size (in Number of Leaf Nodes) on BALANCE based on 10 Independent Runs when the Mutation Probability p_m Varies

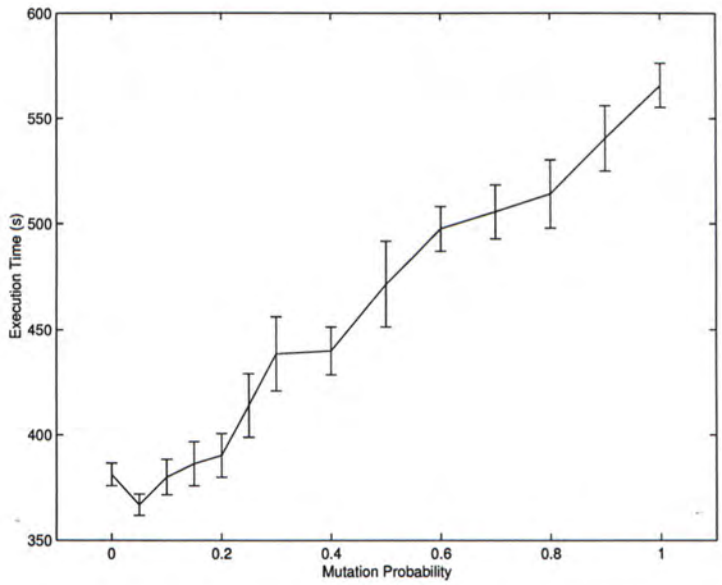

Figure 4.12: Execution Time (in Seconds) of GA-based QDT on BALANCE versus Mutation Probability p_m

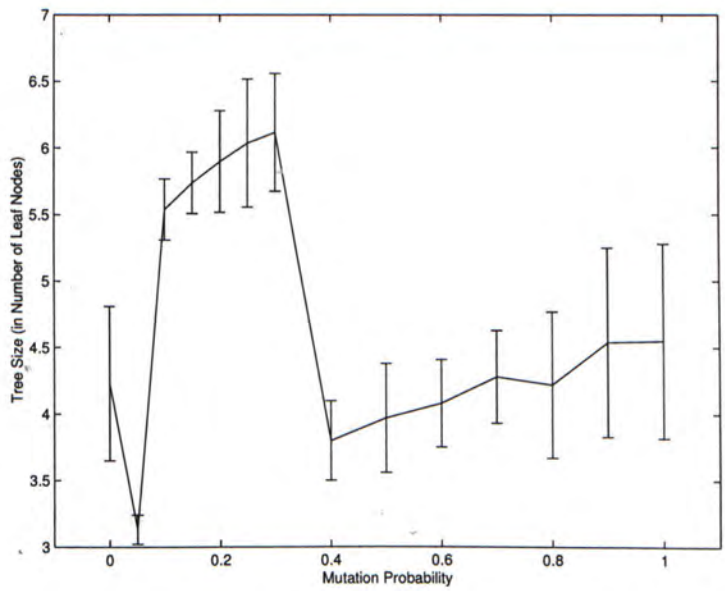

Figure 4.13: Tree Size (in Number of Leaf Nodes) of GA-based QDT on BALANCE versus Mutation Probability p_m

n_0	Validation Accuracy (%)	Execution Time (s)	Tree Size
10	97.20 ± 0.51	367.3 ± 6.1	5.17 ± 0.16
15	97.20 ± 0.51	367.6 ± 5.9	5.17 ± 0.16
20	97.23 ± 0.49	366.9 ± 4.8	5.13 ± 0.11
25	97.23 ± 0.49	366.8 ± 4.7	5.13 ± 0.11
30	97.23 ± 0.49	366.7 ± 5.1	5.13 ± 0.11
35	97.09 ± 0.56	367.1 ± 4.5	5.11 ± 0.10
40	97.09 ± 0.56	364.9 ± 3.3	5.11 ± 0.10
45	97.04 ± 0.54	366.2 ± 7.5	5.08 ± 0.09
50	97.10 ± 0.62	363.7 ± 5.3	5.00 ± 0.00

Table 4.12: Average and Standard Deviation of Validation Accuracy (%), Execution Time (in Seconds) and Tree Size (in Number of Nodes) on BALANCE based on 10 Independent Runs when the Minimum Number of Training Samples n_0 Varies

Minimum Number of Training Samples (n_0)

In this part, the minimum number of training samples increases from 10 to 50 while the values of the other parameters are shown in Table 4.4.

Table 4.12 shows the average and the standard deviation of the validation accuracy, the execution time and the tree size (in number of nodes) based on 10 independent runs when the minimum number of training samples n_0 equals $10, 15..., 50$. The results reported in Table 4.12 are used to plot Figures 4.14, 4.15 and 4.16. The maximum validation accuracy is attained when the minimum number of training samples is 20, 25 or 30. As expected, the number of nodes decreases as the minimum number of training samples increases.

Minimum Impurity Reduction (g_0)

In this part, the minimum impurity reduction is increased from 0.05 to 1.0 while the values of the other parameters are shown

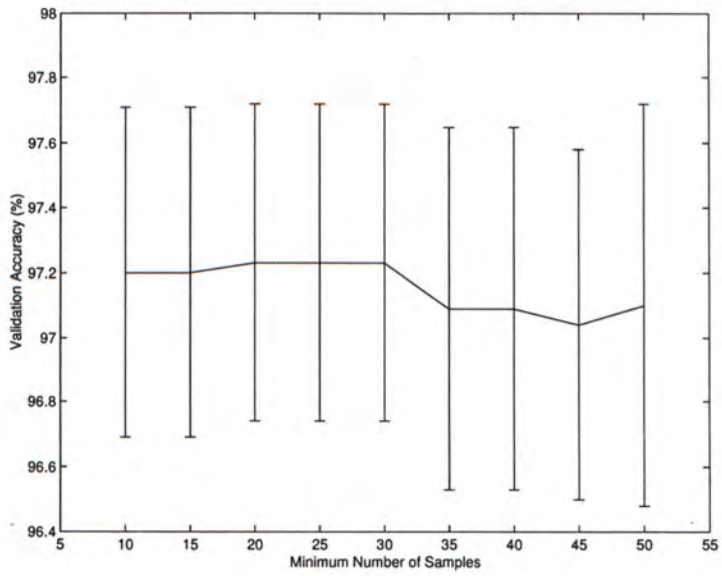

Figure 4.14: Validation Accuracy (%) of GA-based QDT on BALANCE versus Minimum Number of Samples n_0

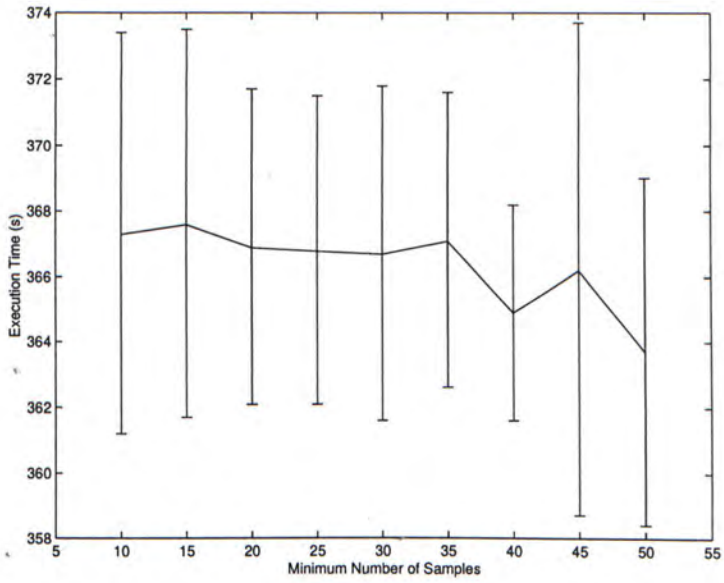

Figure 4.15: Execution Time (in Seconds) of GA-based QDT on BALANCE versus Minimum Number of Samples n_0

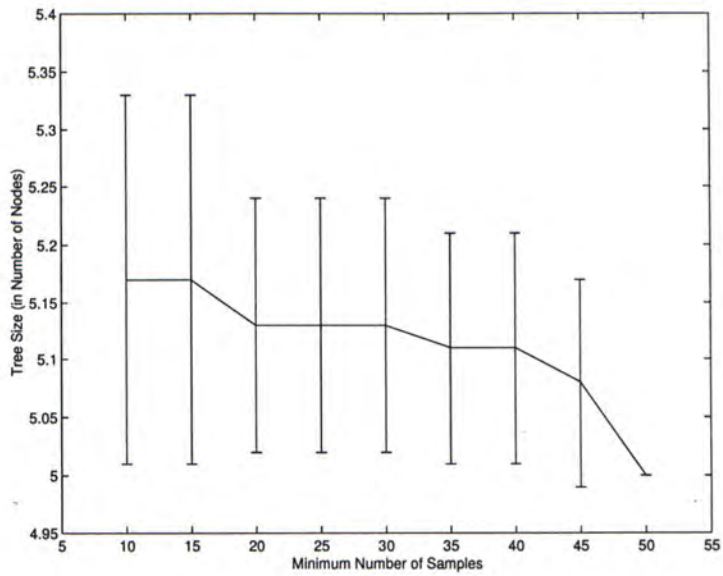

Figure 4.16: Tree Size (in Number of Nodes) of GA-based QDT on BALANCE versus Minimum Number of Samples n_0

in Table 4.4.

Table 4.13 reports the average and the standard deviation of the validation accuracy, the execution time and the tree size (in number of nodes) based on 10 independent runs when the minimum impurity reduction g_0 equals $0.05, 0.1..., 0.25, 0.3, 0.4..., 1$. The results reported in Table 4.13 are used to plot Figures 4.17, 4.18 and 4.19. The maximum validation accuracy is attained when the minimum impurity reduction is 0.1. As the minimum impurity reduction increases from 0.1 to 0.5, the validation accuracy decreases. But the validation accuracy remains stable as the minimum impurity reduction increases from 0.5 to 1.0.

As expected, the number of nodes of a GA-based QDT decreases as the minimum impurity reduction increases. When the impurity reduction equals 0.5, every constructed decision tree has the root node only because GAs fails to find a quadratic

g_0	Validation Accuracy (%)	Execution Time (s)	Tree Size
0.05	97.20 ± 0.40	402.6 ± 9.8	5.48 ± 0.15
0.1	97.23 ± 0.49	366.7 ± 5.1	5.13 ± 0.11
0.15	97.04 ± 0.52	362.5 ± 3.1	5.00 ± 0.05
0.2	96.67 ± 0.81	455.1 ± 6.2	4.92 ± 0.08
0.25	91.82 ± 0.47	290.5 ± 4.5	4.12 ± 0.10
0.3	91.26 ± 0.32	232.1 ± 0.7	3.00 ± 0.00
0.4	91.39 ± 0.42	218.2 ± 4.9	3.00 ± 0.00
0.5	41.30 ± 1.56	215.3 ± 3.8	1.00 ± 0.00
0.6	41.30 ± 1.56	0.0 ± 0.0	1.00 ± 0.00
0.7	41.30 ± 1.56	0.0 ± 0.0	1.00 ± 0.00
0.8	41.30 ± 1.56	0.0 ± 0.0	1.00 ± 0.00
0.9	41.30 ± 1.56	0.0 ± 0.0	1.00 ± 0.00
1.0	41.30 ± 1.56	0.0 ± 0.0	1.00 ± 0.00

Table 4.13: Average and Standard Deviation of Validation Accuracy (%), Execution Time (in Seconds) and Tree Size (in Number of Nodes) on BALANCE based on 10 Independent Runs when the Minimum Impurity Reduction g_0 Varies

hypersurface such that the impurity reduction is greater than or equal to 0.5.

When the minimum impurity reduction is greater than or equal to 0.6, the execution time is almost zero and every constructed decision tree has the root node only. The GA-based QDT algorithm does not attempt to find the optimal quadratic hypersurface because the impurity of every possible training set of the BALANCE dataset is less than the minimum impurity reduction. The impurity of the BALANCE dataset should lie between the range 0.5 and 0.6.

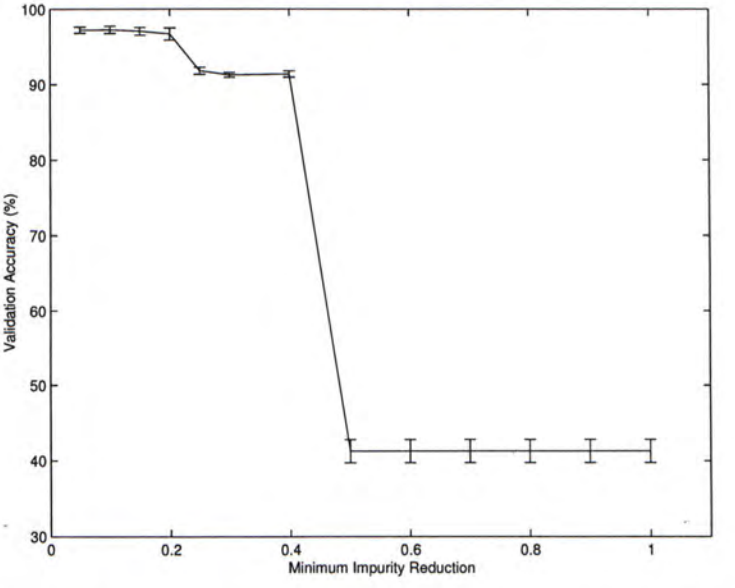

Figure 4.17: Validation Accuracy (%) of GA-based QDT on BALANCE versus Minimum Impurity Reduction g_0

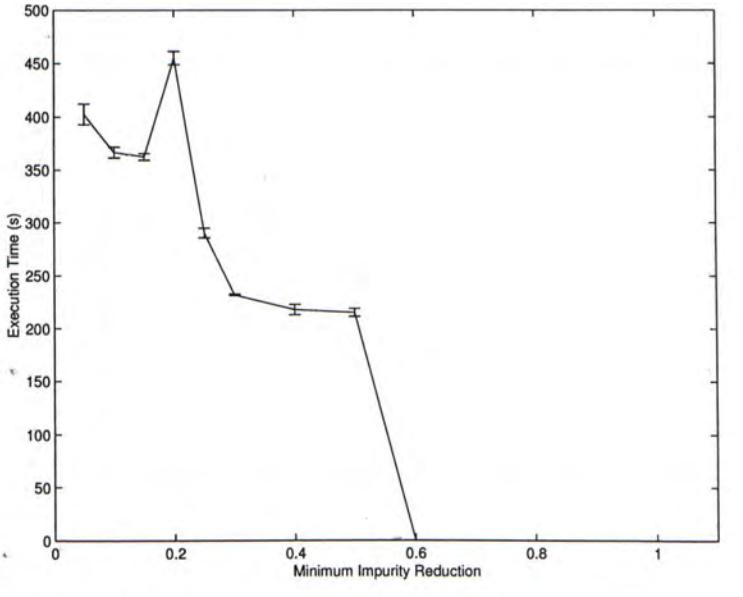

Figure 4.18: Execution Time (in Seconds) of GA-based QDT on BALANCE versus Minimum Impurity Reduction g_0

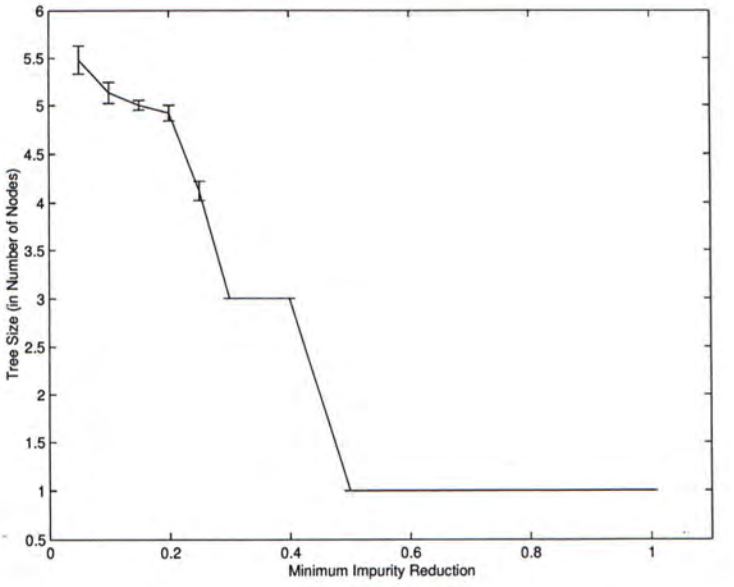

Figure 4.19: Tree Size (in Number of Nodes) of GA-based QDT on BALANCE versus Minimum Impurity Reduction g_0

4.5 Chapter Summary

In this chapter, a novel multivariate decision tree algorithm called GA-based QDT has been proposed by extending the BTGA algorithm. The algorithm to construct a GA-based QDT has been discussed. The performance of the GA-based QDT is compared with that of various supervised classification algorithms. When a new node of a quadratic decision tree is created, the performance of searching for the optimal quadratic hypersurface using random search has been compared to that using GAs. The effects of changing one of the parameters of the GA-based QDT have been studied.

Experiments show that the GA-based QDT provides a better approximation to non-linear class boundaries when compared with other univariate and linear decision tree algorithms. Moreover, GAs are more capable of searching for the optimal

quadratic decision function than random search at each internal node of a quadratic decision tree.

□ **End of chapter.**

Chapter 5

Induction of Linear and Quadratic Decision Trees using Spatial Data Structures

5.1 Introduction

In this chapter, two spatial data structures, including k-D trees and generalized quadtrees, are applied to speed up the construction of oblique and quadratic decision trees on datasets with sufficiently large number of training samples. When a new node of an oblique or a quadratic decision tree is created, either a k-D tree or a generalized quadtree is constructed using the training samples arriving at that node.

k-D trees are binary trees. There are two child nodes at each non-leaf node of a k-D tree. At each non-leaf node of a k-D tree, one of the input attributes is chosen to divide a set of training samples into two disjoint subsets.

There are at most 2^d child nodes at each non-leaf node of a generalized quadtree, where d is the number of input attributes of a sample. At each non-leaf node of a generalized quadtree, all the input attributes are applied to divide a set of training samples into at most 2^d disjoint subsets.

Both oblique and quadratic decision trees are usually con-

structed using a top-down approach. At each non-leaf node of an oblique decision tree, the optimal linear decision function is determined. The linear decision function at a non-leaf node is of the form:

$$w_1x_1 + w_2x_2 + \ldots + w_dx_d > w_0, \tag{5.1}$$

where $w_0, w_1 \ldots, w_d$ are the coefficients of a linear decision function. A linear decision function is equivalent to a hyperplane in a d-dimensional attribute space. On the other hand, the optimal quadratic decision function is determined at each non-terminal node of a quadratic decision tree. The quadratic decision function at a non-leaf node is specified in (4.1). At each leaf node of an oblique or a quadratic decision tree, there is a class label to classify a sample arriving at that node.

The optimality of a hyperplane or a quadratic hypersurface at a non-terminal node is determined using the impurity reduction after partitioning a set of training samples into two disjoint subsets. The impurity of a set of samples can be measured by the Gini-index, entropy and so on. To measure the impurity reduction after dividing a set of training samples into two disjoint subsets, it is required to find the number of training samples for each class such that either (4.1) or (5.1) is satisfied, depending on whether a quadratic or an oblique decision tree is being constructed. An intuitive way to determine the number of training samples satisfying (4.1) or (5.1) for each class is to consider each of these training samples. However, this approach is time consuming on large or high-dimensional datasets because the computational time required to evaluate the impurity reduction is $O(dn)$ for a hyperplane and $O(d^2n)$ for a quadratic hypersurface, where n is the number of training samples being considered.

Alternatively, either a k-D tree or a generalized quadtree is constructed before searching for the optimal linear or quadratic

decision function at each internal node of an oblique or a quadratic decision tree. At each node of a k-D tree or a generalized quadtree, there is the associated smallest hyperrectangle containing all the training samples arriving at that node. If a quadratic (or a linear) decision function does not intersect the hyperrectangle, all the training samples inside the hyperrectangle either satisfy or violate (4.1) (or (5.1)). In this case, there is no need to determine whether (4.1) or (5.1) is satisfied for each of these training samples. Figures 5.1 and 5.2 respectively illustrate a linear and a quadratic decision functions which do not intersect the smallest hyperrectangle containing a set of training samples. Otherwise, all of the descendants are considered until the decision function in (4.1) or (5.1) does not intersect the smallest hyperrectangle of a non-leaf node or a leaf node is being considered. If a leaf node is being processed and the decision function in (4.1) or (5.1) intersects the smallest hyperrectangle containing all the training samples arriving at that node, it is necessary to calculate the sign for each of these training samples. Figures 5.3 and 5.4 respectively illustrate a linear and a quadratic decision functions which intersect the smallest hyperrectangle containing a set of training samples.

5.2 Construction of k-D Trees

When a new node N_h of an oblique or a quadratic decision tree is created, either a k-D tree or a generalized quadtree is constructed and the optimal decision function is determined if the impurity of the set S_h of training samples arriving at that node is not less than a threshold g_0 and $|S_h|$ is greater than or equal to a positive integer n_0. In this section, the structure of a k-D tree is described first. The algorithm to construct a k-D tree is then discussed.

A k-D tree is a binary tree. There are two child nodes at each

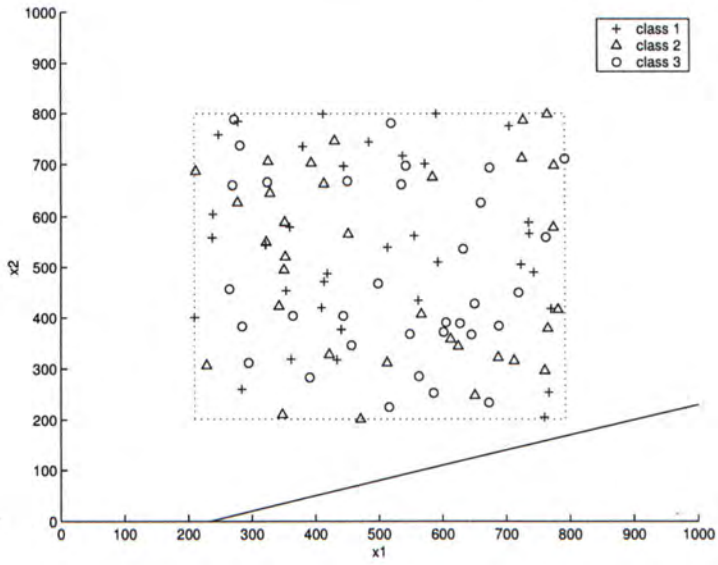

Figure 5.1: An Example that a Linear Decision Function Does Not Intersect the Smallest Rectangle Containing a Set of Training Samples

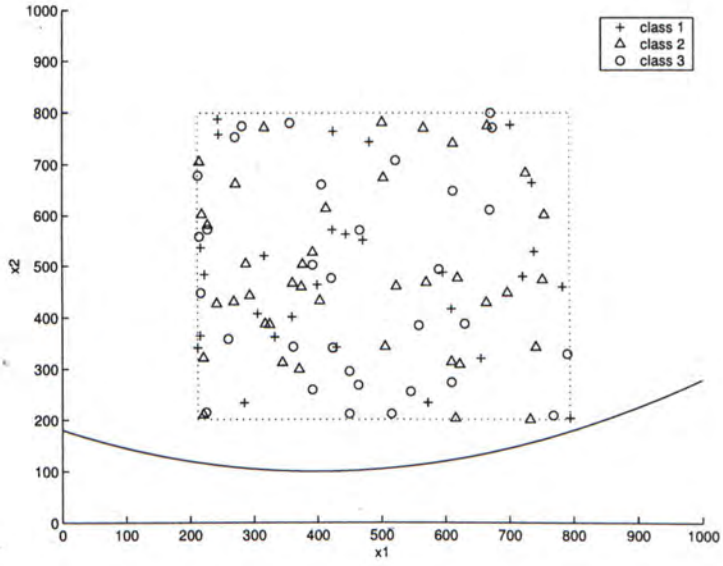

Figure 5.2: An Example that a Quadratic Decision Function Does Not Intersect the Smallest Rectangle Containing a Set of Training Samples

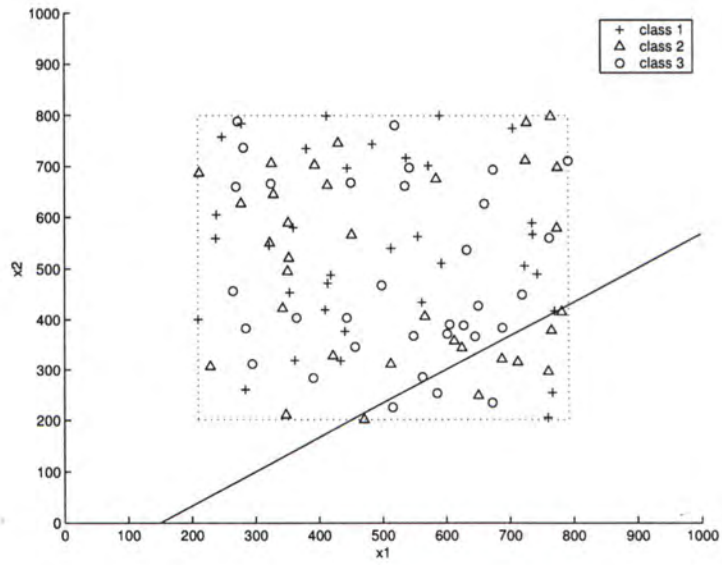

Figure 5.3: An Example that a Linear Decision Function Intersects the Smallest Rectangle Containing a Set of Training Samples

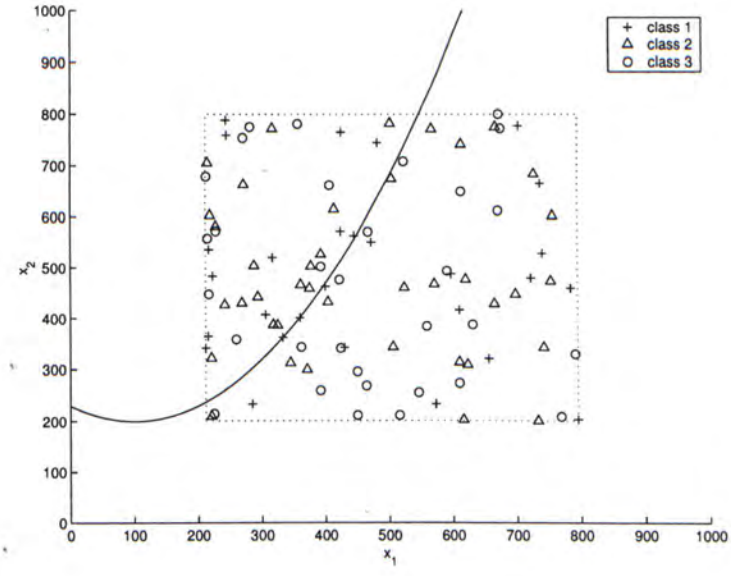

Figure 5.4: An Example that a Quadratic Decision Function Intersects the Smallest Rectangle Containing a Set of Training Samples

non-leaf node of a k-D tree. Suppose each sample has d input attributes, a non-leaf node N_Q of a k-D tree has the following attributes as shown in Figure 5.5:

1. A set S_Q of training samples arriving at the node N_Q of a k-D tree.
2. A set $S_{Q,i}$, $i = 1, 2..., C$, of training samples of class i arriving at the node N_Q, where C is the number of classes.
3. A d-dimensional vector $\mathbf{y_Q} = (y_{Q,1}, y_{Q,2}..., y_{Q,d})$, where $y_{Q,i}$, $i = 1, 2..., d$, is the minimum value of the i^{th} input attribute of the set S_Q of training samples.
4. A d-dimensional vector $\mathbf{z_Q} = (z_{Q,1}, z_{Q,2}..., z_{Q,d})$, where $z_{Q,i}$, $i = 1, 2..., d$, is the maximum value of the i^{th} input attribute of the set S_Q of training samples.
5. A positive integer $k_Q \in \{1, 2..., d\}$, denoting the k_Q^{th} input attribute is chosen to divide the set S_Q of training samples into two disjoint subsets.
6. A real number γ_Q representing the threshold for dividing the set S_Q of training samples into two disjoint subsets using the k_Q^{th} input attribute.
7. A pointer L_Q pointing to the left child node of the node N_Q.
8. A pointer R_Q pointing to the right child node of the node N_Q.

A leaf node of a k-D tree has the first four attributes only. Note that $[y_{Q,1}, z_{Q,1}] \times [y_{Q,2}, z_{Q,2}] ... \times [y_{Q,d}, z_{Q,d}]$ represents the smallest hyperrectangle containing the set S_Q of training samples arriving at the node N_Q of a k-D tree. Figure 5.5 shows an example k-D tree used in this chapter. To construct a k-D tree using the set

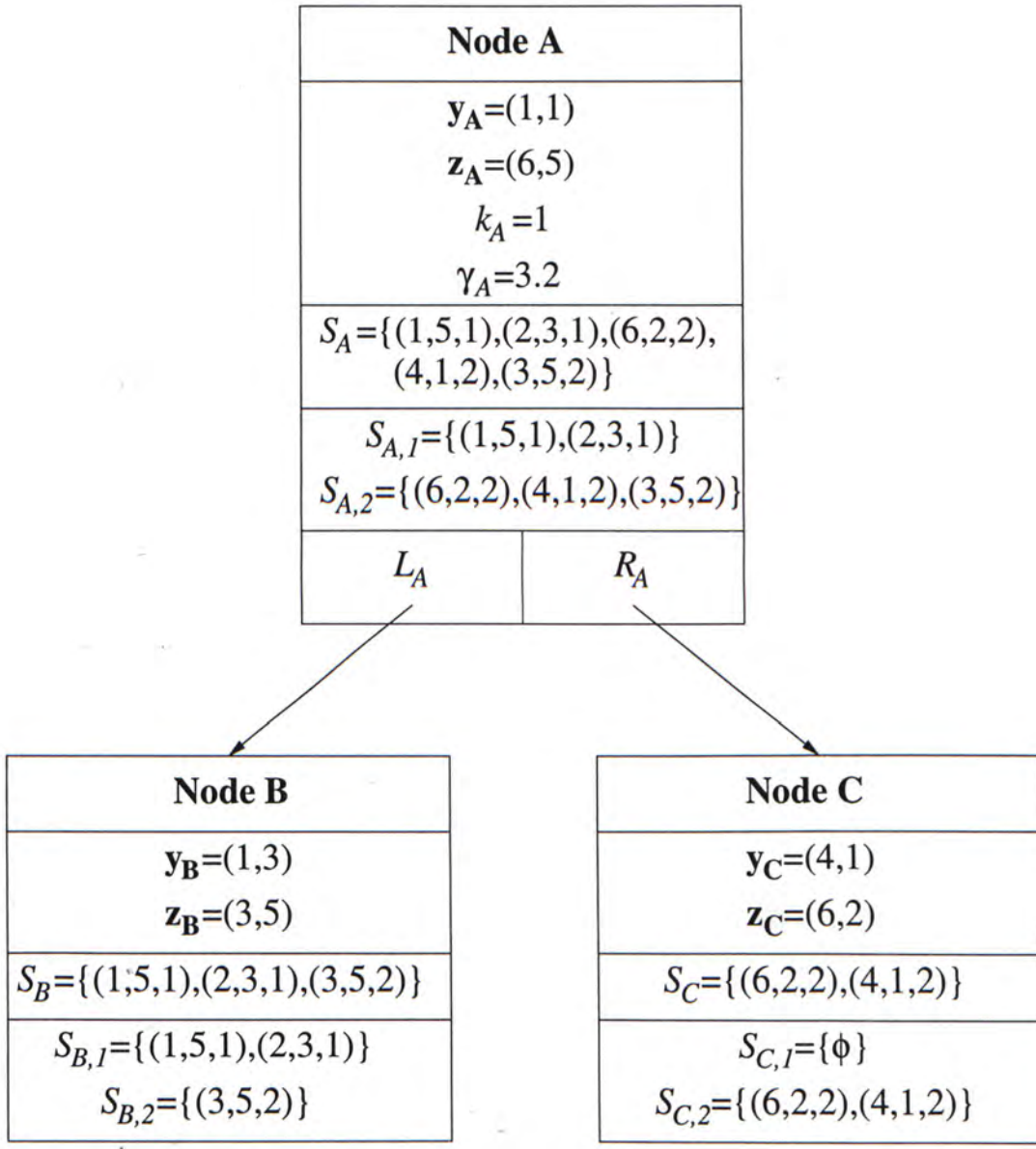

Figure 5.5: An Example k-D Tree

S_h of training samples arriving at node N_h of a decision tree, the root node of a k-D tree is created first. At node N_Q of a k-D tree, two new child nodes are created when the number of training samples arriving at that node is not less than a positive integer n_Q and there exists $k_Q \in \{1, 2..., d\}$ such that $y_{Q,k_Q} \neq z_{Q,k_Q}$.

When a new node N_Q of a k-D tree is created, the attributes are initialized using the set S_Q of training samples arriving at that node. If node N_Q is the root node of a k-D tree, the smallest positive integer $k_Q \in \{1, 2..., d\}$ such that $y_{Q,k_Q} \neq z_{Q,k_Q}$ is determined and the k_Q^{th} input attribute is chosen to partition the set S_Q of training samples into two disjoint subsets. If node N_Q is a non-root node of a k-D tree, the following outlines the steps to determine the value of $k_Q \in \{1, 2..., d\}$:

1. Determine the value of $k_P \in \{1, 2..., d\}$ such that the k_P^{th} input attribute is chosen to divide the set of training samples arriving at the parent of the node N_Q.

2. Set $k_Q = (k_P \bmod d) + 1$.

3. WHILE $y_{Q,k_Q} = z_{Q,k_Q}$ DO
   Set $k_Q = (k_Q \bmod d) + 1$.

Given the set $S_Q = \{\mathbf{x_Q^{(1)}}, \mathbf{x_Q^{(2)}}..., \mathbf{x_Q^{(|S_Q|)}}\}$ of training samples arriving at the node N_Q of a k-D tree, the threshold γ_Q for dividing the set S_Q into two disjoint subsets is given by:

$$\gamma_Q = \frac{1}{|S_Q|} \sum_{i=1}^{|S_Q|} x_{Q,k_Q}^{(i)}, \tag{5.2}$$

where $x_{Q,k_Q}^{(i)}$, $i = 1, 2..., |S_Q|$, is the value of the k_Q^{th} input attribute of the sample $\mathbf{x_Q^{(i)}}$.

Figure 5.6 shows the algorithm of the procedure `createkDTree()`, which outlines the steps of creating a new node of a k-D tree. To create the root node of a k-D tree, the procedure `createkDTree()`

accepts the set S_h of training samples arriving at node N_h of a decision tree as a parameter.

5.3 Construction of Generalized Quadtrees

In this section, the structure of a generalized quadtree is described first. After that, the algorithm to construct a generalized quadtree is discussed.

When a sample has d input attributes, there are at most 2^d child nodes at each non-leaf node of a generalized quadtree. A non-leaf node N_Q of a generalized quadtree has the following attributes as shown in Figure 5.7:

1. A set S_Q of training samples arriving at the node N_Q of a generalized quadtree.
2. A set $S_{Q,i}$, $i = 1, 2..., C$, of training samples of class i arriving at the node N_Q, where C is the number of classes.
3. A d-dimensional vector $\mathbf{y_Q} = (y_{Q,1}, y_{Q,2}..., y_{Q,d})$, where $y_{Q,i}$, $i = 1, 2..., d$, is the minimum value of the i^{th} input attribute of the set S_Q of training samples.
4. A d-dimensional vector $\mathbf{z_Q} = (z_{Q,1}, z_{Q,2}..., z_{Q,d})$, where $z_{Q,i}$, $i = 1, 2..., d$, is the maximum value of the i^{th} input attribute of the set S_Q of training samples.
5. A d-dimensional vector $\gamma_\mathbf{Q} = (\gamma_{Q,1}, \gamma_{Q,2}..., \gamma_{Q,d})$, where $\gamma_{Q,i}$, $i = 1, 2..., d$, is the threshold for the i^{th} input attribute to partition the set S_Q of training samples.
6. A set of pointers $\Gamma_{Q,1}, \Gamma_{Q,2}..., \Gamma_{Q,2^d}$, where $\Gamma_{Q,i}$, $i = 1, 2..., 2^d$, is the pointer pointing to the i^{th} child node of the node N_Q of a generalized quadtree.

A leaf node of a generalized quadtree has the first four attributes only. If node N_Q is a non-leaf node, it is possible that there

PROCEDURE `createkDTree`

INPUTS
- A set of training samples $S_Q = \{\mathbf{x_Q^{(1)}}, \mathbf{x_Q^{(2)}}..., \mathbf{x_Q^{(|S_Q|)}}\}$, where $\mathbf{x_Q^{(i)}} = (x_{Q,1}^{(i)}, x_{Q,2}^{(i)}..., x_{Q,d}^{(i)}, c_Q^{(i)})^T$, $i = 1, 2..., |S_Q|$, $x_{Q,1}^{(i)}, x_{Q,2}^{(i)}..., x_{Q,d}^{(i)}$ are the input attributes and $c_Q^{(i)}$ is the class label of the sample $\mathbf{x_Q^{(i)}}$.
- A positive integer $k \in \{1, 2..., d\}$. If the root node of a k-D tree is being created, set $k = 1$.

OUTPUT A pointer to a new node N_Q of a k-D tree.

1. Initialize the vector $\mathbf{y_Q} = (y_{Q,1}, y_{Q,2}..., y_{Q,d})$, where $y_{Q,i}$, $i = 1, 2..., d$, is the minimum value of the i^{th} input attribute of the set S_Q of training samples.
2. Initialize the vector $\mathbf{z_Q} = (z_{Q,1}, z_{Q,2}..., z_{Q,d})$, where $z_{Q,i}$, $i = 1, 2..., d$, is the maximum value of the i^{th} input attribute of the set S_Q of training samples.
3. Initialize the set $S_{Q,i}$, $i = 1, 2..., C$, of training samples of class i arriving at the node N_Q.
4. IF $|S_Q| < n_Q$ OR $\mathbf{y_Q} = \mathbf{z_Q}$, THEN the node N_Q is declared as a leaf node and go to step 10.
5. Set $k_Q = (k \bmod d) + 1$.
6. WHILE $y_{Q,k_Q} = z_{Q,k_Q}$ DO
   Set $k_Q = (k_Q \bmod d) + 1$.
7. Set $\gamma_Q = \frac{1}{|S_Q|} \sum_{i=1}^{|S_Q|} x_{Q,k_Q}^{(i)}$.
8. Set L_Q = `createkDTree`(S_Q^L, k_Q), where $S_Q^L = \{\mathbf{x} \in S_Q | x_{k_Q} < \gamma_Q\}$, where x_i, $i = 1, 2..., d$, is the value of the i^{th} input attribute of a sample $\mathbf{x} \in S_Q$.
9. Set R_Q = `createkDTree`(S_Q^R, k_Q), where $S_Q^R = S_Q \setminus S_Q^L$.
10. Return the pointer to the node N_Q.

Figure 5.6: The Algorithm of the Procedure `createkDTree()`

exists $i \in \{1, 2..., 2^d\}$ such that $\Gamma_{Q,i}$ is a null pointer. Note that $[y_{Q,1}, z_{Q,1}] \times [y_{Q,2}, z_{Q,2}] ... \times [y_{Q,d}, z_{Q,d}]$ represents the smallest hyperrectangle containing the set S_Q of training samples arriving at the node N_Q of a generalized quadtree. Figure 5.7 shows an example generalized quadtree used in this chapter, where $d = 2$. To construct a generalized quadtree using the set S_h of training samples arriving at node N_h of a decision tree, the root node of a generalized quadtree is created first. At node N_Q of a generalized quadtree, at most 2^d child nodes are created when the number of training samples arriving at that node is greater than or equal to a positive integer n_Q and there exists $k_Q \in \{1, 2..., d\}$ such that $y_{Q,k_Q} \neq z_{Q,k_Q}$.

When a new node N_Q is created, the attributes are initialized using the set S_Q of training samples arriving at that node. All the input attributes are applied to partition the set S_Q of training samples into at most 2^d disjoint subsets. Suppose $S_Q = \{\mathbf{x}_\mathbf{Q}^{(1)}, \mathbf{x}_\mathbf{Q}^{(2)} ..., \mathbf{x}_\mathbf{Q}^{(|\mathbf{S_Q}|)}\}$, where $\mathbf{x}_\mathbf{Q}^{(\mathbf{i})} = (x_{Q,1}^{(i)}, x_{Q,2}^{(i)} ..., x_{Q,d}^{(i)}, c_Q^{(i)})^T$, $1 \leq i \leq |S_Q|$, $x_{Q,1}^{(i)}, x_{Q,2}^{(i)} ..., x_{Q,d}^{(i)}$ are the input attributes and $c_Q^{(i)}$ is the class label of the sample $\mathbf{x}_\mathbf{Q}^{(\mathbf{i})}$, the value of $\gamma_{Q,j}$, $j = 1, 2..., d$, is given by:

$$\gamma_{Q,j} = \frac{1}{|S_Q|} \sum_{i=1}^{|S_Q|} x_{Q,j}^{(i)}. \tag{5.3}$$

Figure 5.8 shows the algorithm of the procedure `createQuadtree()`, which outlines the steps of creating a new node of a generalized quadtree. To create the root node of a generalized quadtree, the procedure `createQuadtree()` accepts the set S_h of training samples arriving at node N_h of a decision tree as the parameter.

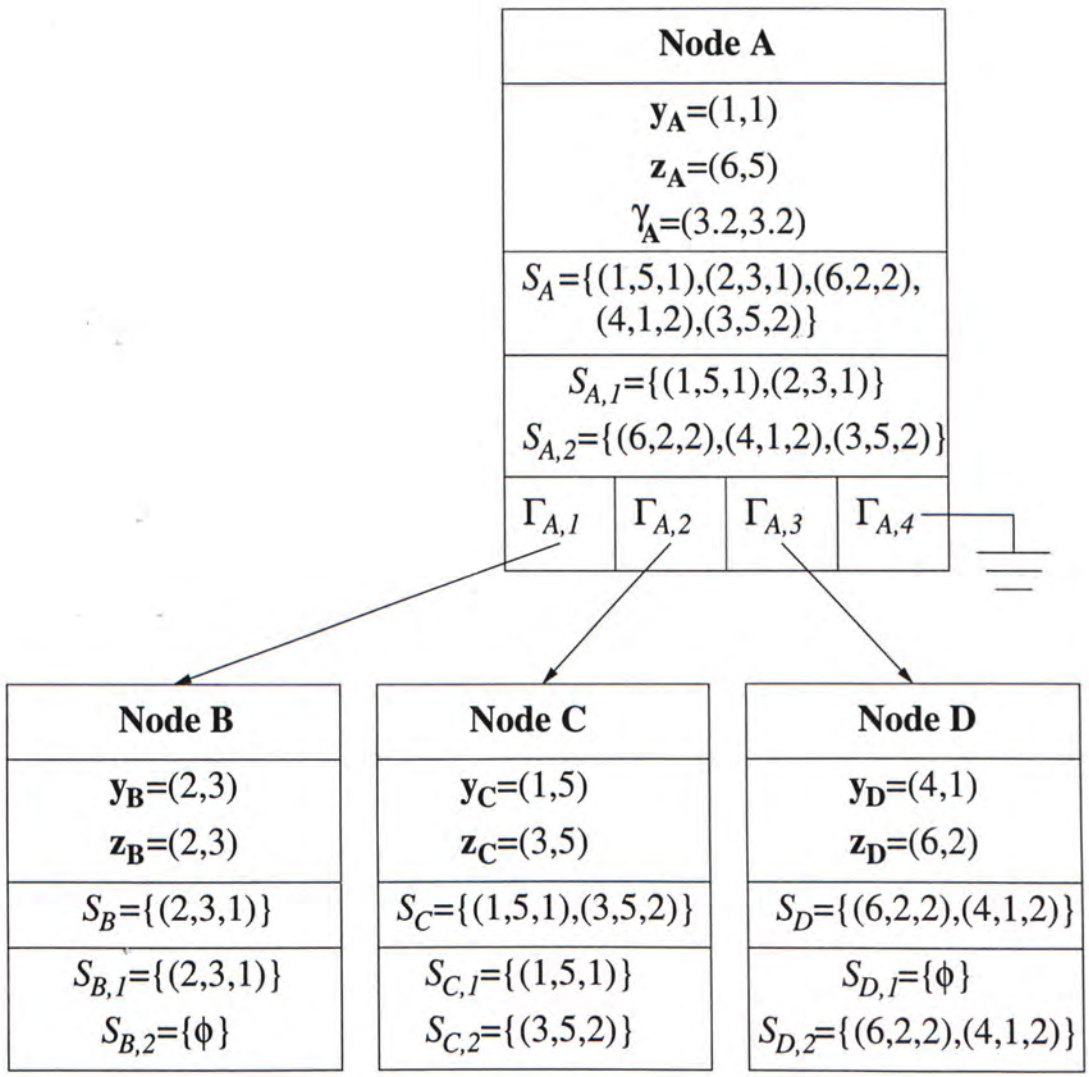

Figure 5.7: An Example Generalized Quadtree

PROCEDURE `createQuadtree`

INPUT A set of training samples $S_Q = \{\mathbf{x_Q^{(1)}}, \mathbf{x_Q^{(2)}}..., \mathbf{x_Q^{(|S_Q|)}}\}$, where $\mathbf{x_Q^{(i)}} = (x_{Q,1}^{(i)}, x_{Q,2}^{(i)}..., x_{Q,d}^{(i)}, c_Q^{(i)})^T$, $i = 1, 2..., |S_Q|$, $x_{Q,1}^{(i)}, x_{Q,2}^{(i)}..., x_{Q,d}^{(i)}$ are the input attributes and $c_Q^{(i)}$ is the class label of the sample $\mathbf{x_Q^{(i)}}$.

OUTPUT A pointer to a new node N_Q of a generalized quadtree.

1. Initialize the vector $\mathbf{y_Q} = (y_{Q,1}, y_{Q,2}..., y_{Q,d})$, where $y_{Q,i}$, $i = 1, 2..., d$, is the minimum value of the i^{th} input attribute of the set S_Q of training samples.

2. Initialize the vector $\mathbf{z_Q} = (z_{Q,1}, z_{Q,2}..., z_{Q,d})$, where $z_{Q,i}$, $i = 1, 2..., d$, is the maximum value of the i^{th} input attribute of the set S_Q of training samples.

3. Initialize the set $S_{Q,i}$, $i = 1, 2..., C$, of training samples of class i arriving at the node N_Q.

4. IF $|S_Q| < n_Q$ OR $\mathbf{y_Q} = \mathbf{z_Q}$, THEN the node N_Q is declared as a leaf node and go to step 9.

5. FOR $i = 1$ TO d DO
   Set $\gamma_{Q,i} = \frac{1}{|S_Q|} \sum_{j=1}^{|S_Q|} x_{Q,i}^{(j)}$

6. FOR $i = 1$ TO 2^d DO
   Set $X_{Q,i} = \{\phi\}$.

7. FOR $i = 1$ TO $|S_Q|$ DO

   (a) Set $k = 0$.

   (b) FOR $j = 1$ TO d DO

      i. Set $k = k \times 2$.

      ii. IF $x_{Q,j}^{(i)} \geq \gamma_{Q,j}$, THEN set $k = k + 1$.

   (c) Set $X_{Q,k} = X_{Q,k} \cup \{\mathbf{x_Q^{(i)}}\}$

8. FOR $i = 1$ TO 2^d DO
   IF $|X_{Q,i}| \neq 0$, THEN set $\Gamma_{Q,i} =$ `createQuadtree`$(X_{Q,i})$
   ELSE set $\Gamma_{Q,i}$ as a null pointer.

9. Return the pointer to the node N_Q.

Figure 5.8: The Algorithm of the Procedure `createQuadtree()`

5.4 Induction of Oblique Decision Trees using Spatial Data Structures

An oblique decision tree is usually constructed using a top-down approach. The optimal hyperplane is determined at each non-leaf node of an oblique decision tree. The optimality of a hyperplane is defined as the impurity reduction after dividing a set of training samples into two disjoint subsets. To measure the impurity reduction after partitioning a set of training samples into two disjoint subsets, it is required to find the number of training samples for each class such that (5.1) is satisfied.

In this chapter, the Gini-index is used to measure the impurity of a set of training samples. Suppose S_h is the set of training samples arriving at node N_h of an oblique decision tree. Recall that the impurity of the set S_h is defined as:

$$g_1 = 1 - \sum_{i=1}^{C} (\frac{|S_{h,i}|}{|S_h|})^2 \tag{5.4}$$

where $S_{h,i}$, $i = 1, 2..., C$, is the set of training samples of class i arriving at the node N_h and C is the number of classes. Suppose R_h is the set of training samples arriving at the node N_h such that (5.1) is satisfied and $L_h = S_h \setminus R_h$, the weighted average impurity of the subsets L_h and R_h is defined as:

$$g_2 = \frac{|R_h|}{|S_h|}(1 - \sum_{i=1}^{C} (\frac{|R_{h,i}|}{|R_h|})^2) + \frac{|L_h|}{|S_h|}(1 - \sum_{i=1}^{C} (\frac{|L_{h,i}|}{|L_h|})^2) \tag{5.5}$$

where $R_{h,i}$, $i = 1, 2..., C$, is the set of training samples of class i arriving at the node N_h such that (5.1) is satisfied and $L_{h,i} = S_{h,i} \setminus R_{h,i}$, $i = 1, 2..., C$. Recall that the impurity reduction after partitioning the set S_h into the subsets L_h and R_h is defined as:

$$g' = g_1 - g_2. \tag{5.6}$$

Therefore, it is necessary to find the values of $|S_{h,i}|$, $|R_{h,i}|$ and $|L_{h,i}|$, $i = 1, 2..., C$, before evaluating the impurity reduction after dividing the set S_h into two disjoint subsets. The value of $|S_{h,i}|$, $i = 1, 2..., C$, can be found using the set S_h. Note that $|L_{h,i}| = |S_{h,i}| - |R_{h,i}|$, $i = 1, 2..., C$. Therefore, the most time-consuming task before evaluating the impurity reduction is to find the values of $|R_{h,1}|, |R_{h,2}|..., |R_{h,C}|$.

In this section, $r_{h,i}$, $i = 1, 2..., C$, is defined as the number of elements in the set $R_{h,i}$. The values of $r_{h,1}, r_{h,2}..., r_{h,C}$ can be intuitively determined by considering each of the training samples in the set S_h. Let $w_0, w_1..., w_d$ be the coefficients of the linear decision function in (5.1). Suppose $S_h = \{\mathbf{x_h^{(1)}}, \mathbf{x_h^{(2)}}..., \mathbf{x_h^{(|S_h|)}}\}$, where $\mathbf{x_h^{(i)}} = (x_{h,1}^{(i)}, x_{h,2}^{(i)}..., x_{h,d}^{(i)}, c_h^{(i)})^T$, $i = 1, 2 ..., |S_h|$, $x_{h,1}^{(i)}$, $x_{h,2}^{(i)}$ ..., $x_{h,d}^{(i)}$ are the input attributes and $c_h^{(i)}$ is the class label of the sample $\mathbf{x_h^{(i)}}$, the sign of the following expression is evaluated for each training sample $\mathbf{x_h^{(i)}} \in S_h$:

$$\sum_{j=1}^{d} x_{h,j}^{(i)} w_j - w_0. \tag{5.7}$$

The following outlines the steps to find the values of $r_{h,1}, r_{h,2}..., r_{h,C}$ when the linear decision function in (5.1) is used to partition the set S_h of training samples into two disjoint subsets:

1. Set $r_{h,i} = 0$, $i = 1, 2..., C$.
2. FOR $i = 1$ TO $|S_h|$ DO
   (a) Set $\delta_i = \sum_{j=1}^{d} x_{h,j}^{(i)} w_j - w_0$.
   (b) IF $\delta_i > 0$ THEN, $r_{h,c_h^{(i)}} = r_{h,c_h^{(i)}} + 1$.

Alternatively, spatial data structures including k-D trees and generalized quadtrees can be applied to determine the values

of $r_{h,1}, r_{h,2}..., r_{h,C}$ by considering a subset of the training samples in the set S_h. When node N_Q of a k-D tree or a generalized quadtree is being considered, it is necessary to determine whether the linear decision function in (5.1) intersects the smallest hyperrectangle $H_Q = [y_{Q,1}, z_{Q,1}] \times [y_{Q,2}, z_{Q,2}]... \times [y_{Q,d}, z_{Q,d}]$ containing the set S_Q of training samples arriving at that node.

If the linear decision function in (5.1) intersects the hyperrectangle H_Q and the node N_Q is a non-leaf node of a k-D tree or a generalized quadtree, the children of the node N_Q are considered instead. The process is repeated recursively until a leaf node of a k-D tree or a generalized quadtree is considered, or the smallest hyperrectangle formed by a node is not intersected by the linear decision function.

If the linear decision function in (5.1) intersects the hyperrectangle H_Q and the node N_Q is a leaf node of a k-D tree or a generalized quadtree, the sign of the following expression is evaluated for each training sample $\mathbf{x}_\mathbf{Q}^{(\mathbf{i})} \in S_Q$:

$$\sum_{j=1}^{d} x_{Q,j}^{(i)} w_j - w_0. \tag{5.8}$$

There are two possible scenarios that the linear decision function in (5.1) does not intersect the smallest hyperrectangle H_Q containing the set S_Q of training samples arriving at node N_Q of a k-D tree or a generalized quadtree. The first scenario is that all the training samples in the set S_Q satisfy the linear decision function in (5.1). In this case, the minimum value of the following expression is greater than zero for all $\mathbf{x} = (x_1, x_2..., x_d) \in H_Q$:

$$\sum_{j=1}^{d} x_j w_j - w_0. \tag{5.9}$$

Note that $S_{Q,i} \subseteq R_{h,i}$, $i = 1, 2..., C$. The minimum value of the

expression in (5.9) within the hyperrectangle H_Q is given by:

$$\frac{1}{2}\sum_{j=1}^{d} w_j[y_{Q,j}(1+sgn(w_j))+z_{Q,j}(1-sgn(w_j))]-w_0, \quad (5.10)$$

where

$$sgn(x)=\begin{cases} 1 & \text{if } x>0, \\ 0 & \text{if } x=0, \\ -1 & \text{if } x<0. \end{cases} \quad (5.11)$$

The second scenario is that the linear decision function in (5.1) is violated for all training samples in the set S_Q. Note that $R_h \bigcap S_Q = \phi$. The maximum value of the expression in (5.9) is less than zero for all $\mathbf{x} = (x_1, x_2..., x_d) \in H_Q$. The maximum value of the expression in (5.9) within the hyperrectangle H_Q is given by:

$$\frac{1}{2}\sum_{j=1}^{d} w_j[y_{Q,j}(1-sgn(w_j))+z_{Q,j}(1+sgn(w_j))]-w_0. \quad (5.12)$$

The time required to calculate the minimum and the maximum values of the expression in (5.9) is about twice the time required to determine whether a sample satisfies the linear decision function in (5.1). It is more time-consuming to determine whether the linear decision function in (5.1) intersects the smallest hyperrectangle containing at most two training samples before testing whether the linear decision function is satisfied for each of these training samples. When a node N_Q of a k-D tree or a generalized quadtree contains at most two training samples, neither (5.10) nor (5.12) is evaluated.

Figure 5.9 shows the algorithm of the procedure `processkDTree()`, which outlines the steps to determine the values of $r_{h,1}, r_{h,2}..., r_{h,C}$ using a constructed k-D tree. On the other hand, Figure 5.10 shows the algorithm of the procedure `processQuadtree()`, which outlines the steps to perform the same task using a constructed generalized quadtree.

PROCEDURE `processkDTree`

INPUTS
1. A node N_Q of a k-D tree.
2. A set of training samples $S_Q = \{\mathbf{x_Q^{(1)}}, \mathbf{x_Q^{(2)}}..., \mathbf{x_Q^{(|S_Q|)}}\}$, where $\mathbf{x_Q^{(i)}} = (x_{Q,1}^{(i)}, x_{Q,2}^{(i)}..., x_{Q,d}^{(i)}, c_Q^{(i)})^T$, $i = 1, 2..., |S_Q|$, $x_{Q,1}^{(i)}, x_{Q,2}^{(i)}..., x_{Q,d}^{(i)}$ are the input attributes and $c_Q^{(i)}$ is the class label of the sample $\mathbf{x_Q^{(i)}}$.
3. A $(d+1)$-dimensional vector $\mathbf{w} = (w_0, w_1, ..., w_d)^T$, where $w_0, w_1..., w_d$ are the coefficients of the linear decision function in (5.1).

INPUT/OUTPUT A C-dimensional vector $\mathbf{r_h} = (r_{h,1}, r_{h,2}..., r_{h,C})$, where $r_{h,i}$, $i = 1, 2..., C$, is the number of training samples of class i arriving at node N_h of an oblique decision tree such that (5.1) is satisfied.

1. IF the node N_Q is the root node of a k-D tree, THEN set $r_{h,i} = 0$, $i = 1, 2..., C$.
2. IF $|S_Q| \leq 2$, THEN
   (a) FOR $i = 1$ TO $|S_Q|$ DO
   IF $\sum_{j=1}^{d} w_j x_{Q,j}^{(i)} > w_0$, THEN set $r_{h,c_Q^{(i)}} = r_{h,c_Q^{(i)}} + 1$.
   (b) Return.
3. IF $\frac{1}{2}\sum_{j=1}^{d} w_j[y_{Q,j}(1 + sgn(w_j)) + z_{Q,j}(1 - sgn(w_j))] > w_0$,
   THEN set $r_{h,i} = r_{h,i} + |S_{Q,i}|$, $i = 1, 2, ..., C$
   ELSE IF $\frac{1}{2}\sum_{j=1}^{d} w_j[y_{Q,j}(1 - sgn(w_j)) + z_{Q,j}(1 + sgn(w_j))] < w_0$, THEN
   IF the node N_Q is a non-leaf node, THEN
   (a) Set $S_Q^L = \{\mathbf{x} \in S_Q | x_{k_Q} < \gamma_Q\}$, where x_{k_Q} is the value of the k_Q^{th} input attribute of a training sample $\mathbf{x} \in S_Q$.
   (b) Set $S_Q^R = S_Q \setminus S_Q^L$.
   (c) Invoke the procedure `processkDTree`(L_Q, S_Q^L, $\mathbf{w}$, $\mathbf{r_h}$).
   (d) Invoke the procedure `processkDTree`(R_Q, S_Q^R, $\mathbf{w}$, $\mathbf{r_h}$).

   ELSE FOR $i = 1$ TO $|S_Q|$ DO
   IF $\sum_{j=1}^{d} w_j x_{Q,j}^{(i)} > w_0$, THEN set $r_{h,c_Q^{(i)}} = r_{h,c_Q^{(i)}} + 1$.

Figure 5.9: The Algorithm of the Procedure `processkDTree()`

PROCEDURE `processQuadtree`

INPUTS
1. A node N_Q of a generalized quadtree.
2. A set of training samples $S_Q = \{\mathbf{x_Q^{(1)}}, \mathbf{x_Q^{(2)}}..., \mathbf{x_Q^{(|S_Q|)}}\}$, where $\mathbf{x_Q^{(i)}} = (x_{Q,1}^{(i)}, x_{Q,2}^{(i)}..., x_{Q,d}^{(i)}, c_Q^{(i)})^T$, $i = 1, 2..., |S_Q|$, $x_{Q,1}^{(i)}, x_{Q,2}^{(i)}..., x_{Q,d}^{(i)}$ are the input attributes and $c_Q^{(i)}$ is the class label of the sample $\mathbf{x_Q^{(i)}}$.
3. A $(d+1)$-dimensional vector $\mathbf{w} = (w_0, w_1, ..., w_d)^T$, where $w_0, w_1..., w_d$ are the coefficients of the linear decision function in (5.1).

INPUT/OUTPUT A C-dimensional vector $\mathbf{r_h} = (r_{h,1}, r_{h,2}..., r_{h,C})$, where $r_{h,i}$, $i = 1, 2..., C$, is the number of training samples of class i arriving at node N_h of an oblique decision tree such that (5.1) is satisfied.

1. If the node N_Q is the root node of a generalized quadtree, set $r_{h,i} = 0$, $i = 1, 2..., C$.
2. IF $|S_Q| \leq 2$, THEN
   (a) FOR $i = 1$ TO $|S_Q|$ DO
   IF $\sum_{j=1}^{d} w_j x_{Q,j}^{(i)} > w_0$, THEN set $r_{h,c_Q^{(i)}} = r_{h,c_Q^{(i)}} + 1$.
   (b) Return.
3. IF $\frac{1}{2}\sum_{j=1}^{d} w_j[y_{Q,j}(1 + sgn(w_j)) + z_{Q,j}(1 - sgn(w_j))] > w_0$,
   THEN set $r_{h,i} = r_{h,i} + |S_{Q,i}|$, $i = 1, 2, ..., C$
   ELSE IF $\frac{1}{2}\sum_{j=1}^{d} w_j[y_{Q,j}(1 - sgn(w_j)) + z_{Q,j}(1 + sgn(w_j))] < w_0$, THEN
   IF the node N_Q is a non-leaf node, THEN
   (a) Let $X_{Q,i}$, $i = 1, 2..., 2^d$, be the set of training samples arriving at the i^{th} child node of the node N_Q.
   (b) FOR $i = 1$ TO 2^d DO
   IF $\Gamma_{Q,i}$ is not a null pointer, THEN invoke the procedure `processQuadtree`($\Gamma_{Q,i}$, $X_{Q,i}$, $\mathbf{w}$, $\mathbf{r_h}$).

   ELSE FOR $i = 1$ TO $|S_Q|$ DO
   IF $\sum_{j=1}^{d} w_j x_{Q,j}^{(i)} > w_0$, THEN set $r_{h,c_Q^{(i)}} = r_{h,c_Q^{(i)}} + 1$.

Figure 5.10: The Algorithm of the Procedure `processQuadtree()`

5.5 Induction of Quadratic Decision Trees using Spatial Data Structures

At each non-leaf node of a quadratic decision tree, the optimal quadratic hypersurface is found. The optimality of a quadratic hypersurface is defined as the impurity reduction after partitioning a set of training samples into two disjoint subsets. To evaluate the impurity reduction after dividing a set of training samples into two disjoint subsets, it is necessary to find the number of training samples for each class such that (4.1) is satisfied.

The impurity reduction after dividing the set S_h of training samples into two disjoint subsets is calculated using (4.2), (4.3) and (4.4). From (4.2), (4.3) and (4.4), it is required to find the values of $|S_{h,i}|$, $|R_{h,i}|$ and $|L_{h,i}|$, $i = 1, 2..., C$, before evaluating the impurity reduction. The most time-consuming task before evaluating the impurity reduction is to find the values of $|R_{h,1}|, |R_{h,2}|..., |R_{h,C}|$, because the values of $|S_{h,1}|, |S_{h,2}|..., |S_{h,C}|$ can be determined using the set S_h and $|L_{h,i}| = |S_{h,i}| - |R_{h,i}|$, $i = 1, 2..., C$.

In this section, $r_{h,i}$ is defined such that $r_{h,i} = |R_{h,i}|$, $i = 1, 2..., C$. Suppose $S_h = \{\mathbf{x_h^{(1)}}, \mathbf{x_h^{(2)}}..., \mathbf{x_h^{(|S_h|)}}\}$, where $\mathbf{x_h^{(i)}} = (x_{h,1}^{(i)}, x_{h,2}^{(i)}..., x_{h,d}^{(i)}, c_h^{(i)})^T$, $i = 1, 2 ..., |S_h|$, $x_{h,1}^{(i)}$, $x_{h,2}^{(i)}$ ..., $x_{h,d}^{(i)}$ are the input attributes and $c_h^{(i)}$ is the class label of the sample $\mathbf{x_h^{(i)}}$, the following outlines the steps to calculate the values of $r_{h,1}$, $r_{h,2}$ ..., $r_{h,C}$ when the decision function in (4.1) is applied to partition the set S_h of training samples into two disjoint subsets:

1. Set $r_{h,i} = 0$, $i = 1, 2..., C$.
2. FOR $i = 1$ TO $|S_h|$ DO
   (a) Set $\zeta_i = (\widetilde{\mathbf{x}}_\mathbf{h}^{(\mathbf{i})})^\mathbf{T} A \widetilde{\mathbf{x}}_\mathbf{h}^{(\mathbf{i})} + \mathbf{b}^\mathbf{T} \widetilde{\mathbf{x}}_\mathbf{h}^{(\mathbf{i})} - \gamma$, where $\widetilde{\mathbf{x}}_\mathbf{h}^{(\mathbf{i})} = (x_{h,1}^{(i)}, x_{h,2}^{(i)}..., x_{h,d}^{(i)})^T$.
   (b) IF $\zeta_i > 0$ THEN, $r_{h,c_h^{(i)}} = r_{h,c_h^{(i)}} + 1$.

Alternatively, spatial data structures including k-D trees and generalized quadtrees can be applied to find the values of $r_{h,1}$, $r_{h,2}$ ..., $r_{h,C}$ by considering a subset of the training samples in the set S_h. When node N_Q of a k-D tree or a generalized quadtree is being considered, it is necessary to determine whether the quadratic decision function in (4.1) intersects the smallest hyperrectangle $H_Q = [y_{Q,1}, z_{Q,1}] \times [y_{Q,2}, z_{Q,2}] \ldots \times [y_{Q,d}, z_{Q,d}]$ containing the set S_Q of training samples arriving at that node.

If the quadratic decision function in (4.1) intersects the hyperrectangle H_Q and the node N_Q is a non-leaf node of a k-D tree or a generalized quadtree, the children of the node N_Q are considered instead. The process is repeated recursively until a leaf node of a k-D tree or a generalized quadtree is considered, or the smallest hyperrectangle formed by a node is not intersected by the quadratic decision function.

If node N_Q is a leaf node of a k-D tree or a generalized quadtree, and the corresponding hyperrectangle H_Q is intersected by the quadratic decision function in (4.1), the sign of the following expression is evaluated for each training sample $\mathbf{x}_\mathbf{Q}^{(\mathbf{i})} \in S_Q$:

$$(\widetilde{\mathbf{x}}_\mathbf{Q}^{(\mathbf{i})})^\mathbf{T} A \widetilde{\mathbf{x}}_\mathbf{Q}^{(\mathbf{i})} + \mathbf{b}^\mathbf{T} \widetilde{\mathbf{x}}_\mathbf{Q}^{(\mathbf{i})} - \gamma, \tag{5.13}$$

where $\widetilde{\mathbf{x}}_\mathbf{Q}^{(\mathbf{i})} = (x_{Q,1}^{(i)}, x_{Q,2}^{(i)} \ldots, x_{Q,d}^{(i)})^T$.

There are two possible scenarios that the quadratic decision function in (4.1) does not intersect the smallest hyperrectangle H_Q containing the set S_Q of training samples arriving at node N_Q of a k-D tree or a generalized quadtree. The first scenario is that all the training samples in the set S_Q satisfy the quadratic decision function in (4.1). In this case, the minimum value of the following expression is greater than zero for all $\mathbf{x} = (x_1, x_2 \ldots, x_d) \in H_Q$:

$$\mathbf{x}^\mathbf{T} A \mathbf{x} + \mathbf{b}^\mathbf{T} \mathbf{x} - \gamma. \tag{5.14}$$

Three methods of estimating the minimum value of the above

quadratic expression within a hyperrectangle will be described later.

The second scenario is that the quadratic decision function in (4.1) is violated for all training samples in the set S_Q. The maximum value of the expression in (5.14) within the hyperrectangle H_Q is less than zero.

Three methods of estimating the minimum and the maximum values of the expression in (5.14) will be introduced. In most cases, all of these methods tend to overestimate the maximum value of the expression in (5.14) but underestimate the minimum value of this expression. Therefore, it is possible that a quadratic hypersurface is regarded as cutting a hyperrectangle but this actually does not occur.

However, all of these methods neither underestimate the maximum value of the expression in (5.14) nor overestimate the minimum value of this expression. If the estimated maximum value of the expression in (5.14) is less than zero, its actual value is also less than zero. If the estimated minimum value of the expression in (5.14) is greater than zero, its actual value is also greater than zero. If the estimated maximum value of the expression in (5.14) within a hyperrectangle is less than zero or the estimated minimum value is greater than zero, the corresponding quadratic hypersurface cuts the hyperrectangle. The impurity reduction due to a quadratic hypersurface is evaluated accurately and the classification accuracy of a constructed quadratic decision tree is preserved even k-D trees or generalized quadtrees are employed.

Given the hyperrectangle $H_Q = [y_{Q,1}, z_{Q,1}] \times [y_{Q,2}, z_{Q,2}] \ldots \times [y_{Q,d}, z_{Q,d}]$, where d is the number of input attributes of a sample, the following describes the first algorithm estimating the maximum value ζ_{max} and the minimum value ζ_{min} of the expression in (5.14) within the hyperrectangle H_Q:

1. Set $\zeta_{max} = \sum_{i=1}^{d} \frac{a_{i,i}}{2} [y_{Q,i}^2 (1 - sgn(a_{i,i})) + z_{Q,i}^2 (1 + sgn(a_{i,i}))]$.

2. Set $\zeta_{min} = \sum_{i=1}^{d} \frac{a_{i,i}}{2}[y_{Q,i}^2(1 + sgn(a_{i,i})) + z_{Q,i}^2(1 - sgn(a_{i,i}))]$.
3. FOR $i = 1$ TO $d - 1$ DO
   FOR $j = i + 1$ TO d DO
   - IF $a_{i,j} > 0$, THEN
     (a) Set $\zeta_{max} = \zeta_{max} + 2a_{i,j}z_{Q,i}z_{Q,j}$.
     (b) Set $\zeta_{min} = \zeta_{min} + 2a_{i,j}y_{Q,i}y_{Q,j}$.
     ELSE
     (a) Set $\zeta_{max} = \zeta_{max} + 2a_{i,j}y_{Q,i}y_{Q,j}$.
     (b) Set $\zeta_{min} = \zeta_{min} + 2a_{i,j}z_{Q,i}z_{Q,j}$.
4. Set $\zeta_{max} = \zeta_{max} + \sum_{i=1}^{d} \frac{b_i}{2}[y_{Q,i}(1 - sgn(b_i)) + z_{Q,i}(1 + sgn(b_i))]$.
5. Set $\zeta_{min} = \zeta_{min} + \sum_{i=1}^{d} \frac{b_i}{2}[y_{Q,i}(1 + sgn(b_i)) + z_{Q,i}(1 - sgn(b_i))]$.
6. Set $\zeta_{max} = \zeta_{max} - \gamma$ and $\zeta_{min} = \zeta_{min} - \gamma$.

In the first algorithm, the maximum value $\xi_{i,max}$ and the minimum value $\xi_{i,min}$ of the following quadratic expression for $x_i \in [y_{Q,i}, z_{Q,i}]$, $i = 1, 2..., d$, are estimated:

$$a_{i,i}x_i^2 + b_i x_i \tag{5.15}$$

The following outlines the steps to estimate the values of $\xi_{i,max}$ and $\xi_{i,min}$, $i = 1, 2..., d$:

1. Set $\xi_{i,max} = \frac{a_{i,i}}{2}[y_{Q,i}^2(1 - sgn(a_{i,i})) + z_{Q,i}^2(1 + sgn(a_{i,i}))]$.
2. Set $\xi_{i,min} = \frac{a_{i,i}}{2}[y_{Q,i}^2(1 + sgn(a_{i,i})) + z_{Q,i}^2(1 - sgn(a_{i,i}))]$.
3. Set $\xi_{i,max} = \xi_{i,max} + \frac{b_i}{2}[z_{Q,i}(1 + sgn(b_i)) + y_{Q,i}(1 - sgn(b_i))]$.
4. Set $\xi_{i,min} = \xi_{i,min} + \frac{b_i}{2}[z_{Q,i}(1 - sgn(b_i)) + y_{Q,i}(1 + sgn(b_i))]$.

Note that the maximum value $\xi_{i,max}$, $i = 1, 2..., d$, of the expression in (5.15) within the hyperrectangle H_Q may be overestimated. However, the minimum value $\xi_{i,min}$, $i = 1, 2..., d$,

of the expression in (5.15) within the hyperrectangle H_Q may be underestimated. The impurity reduction due to a quadratic decision function can be evaluated accurately.

Example 1 Estimate the maximum value ζ_{max} and the minimum value ζ_{min} of the quadratic function $f(x_1, x_2) = 2x_1^2 - x_2^2 - 2x_1x_2 - 8x_1 + 10x_2$ within the hyperrectangle $[1, 3] \times [1, 2]$ using the first algorithm.

$$\zeta_{max} = 2 \times 3^2 - 1^2 - 2 \times 1 \times 1 - 8 \times 1 + 10 \times 2 = 27$$

$$\zeta_{min} = 2 \times 1^2 - 2^2 - 2 \times 3 \times 2 - 8 \times 3 + 10 \times 1 = -28$$

In the second algorithm estimating the maximum and the minimum values of the expression in (5.14) within the hyperrectangle H_Q, the maximum value $\xi_{i,max}$ and the minimum value $\xi_{i,min}$ of the expression in (5.15) for $x_i \in [y_{Q,i}, z_{Q,i}]$, $i = 1, 2..., d$, are calculated rather than estimated. Figure 5.11 outlines the steps to calculate the values of $\xi_{i,max}$ and $\xi_{i,min}$, $i = 1, 2..., d$. Although the estimation time is increased when compared with the first algorithm, but the second algorithm provides the same or better estimated maximum and minimum values.

The second algorithm estimating the maximum value ζ_{max} and the minimum value ζ_{min} of the expression in (5.14) within the hyperrectangle H_Q is described as follows:

1. Set $\zeta_{max} = (\sum_{i=1}^{d} \xi_{i,max}) - \gamma$, where $\xi_{i,max}$ is the maximum value of the expression in (5.15) for $x_i \in [y_{Q,i}, z_{Q,i}]$, $i = 1, 2..., d$.

2. Set $\zeta_{min} = (\sum_{i=1}^{d} \xi_{i,min}) - \gamma$, where $\xi_{i,min}$ is the minimum value of the expression in (5.15) for $x_i \in [y_{Q,i}, z_{Q,i}]$, $i = 1, 2..., d$.

3. FOR $i = 1$ TO $d - 1$ DO
   FOR $j = i + 1$ TO d DO

1. Set $p = -\frac{b_i}{2a_{i,i}}$.
2. IF $a_{i,i} > 0$, THEN
   - IF $p < y_{Q,i}$, THEN
     (a) Set $\xi_{i,max} = a_{i,i}z_{Q,i}^2 + b_i z_{Q,i}$.
     (b) Set $\xi_{i,min} = a_{i,i}y_{Q,i}^2 + b_i y_{Q,i}$.
     ELSE IF $p > z_{Q,i}$, THEN
     (a) Set $\xi_{i,max} = a_{i,i}y_{Q,i}^2 + b_i y_{Q,i}$.
     (b) Set $\xi_{i,min} = a_{i,i}z_{Q,i}^2 + b_i z_{Q,i}$.
     ELSE IF $p < \frac{y_{Q,i}+z_{Q,i}}{2}$, THEN
     (a) Set $\xi_{i,max} = a_{i,i}z_{Q,i}^2 + b_i z_{Q,i}$.
     (b) Set $\xi_{i,min} = -\frac{b_i^2}{4a_{i,i}}$.
     ELSE
     (a) Set $\xi_{i,max} = a_{i,i}y_{Q,i}^2 + b_i y_{Q,i}$.
     (b) Set $\xi_{i,min} = -\frac{b_i^2}{4a_{i,i}}$.

   ELSE IF $a_{i,i} < 0$, THEN
   - IF $p < y_{Q,i}$, THEN
     (a) Set $\xi_{i,max} = a_{i,i}y_{Q,i}^2 + b_i y_{Q,i}$.
     (b) Set $\xi_{i,min} = a_{i,i}z_{Q,i}^2 + b_i z_{Q,i}$.
     ELSE IF $p > z_{Q,i}$, THEN
     (a) Set $\xi_{i,max} = a_{i,i}z_{Q,i}^2 + b_i z_{Q,i}$.
     (b) Set $\xi_{i,min} = a_{i,i}y_{Q,i}^2 + b_i y_{Q,i}$.
     ELSE IF $p < \frac{y_{Q,i}+z_{Q,i}}{2}$, THEN
     (a) Set $\xi_{i,max} = -\frac{b_i^2}{4a_{i,i}}$.
     (b) Set $\xi_{i,min} = a_{i,i}z_{Q,i}^2 + b_i z_{Q,i}$.
     ELSE
     (a) Set $\xi_{i,max} = -\frac{b_i^2}{4a_i}$.
     (b) Set $\xi_{i,max} = a_{i,i}y_{Q,i}^2 + b_i y_{Q,i}$.

   ELSE
   (a) Set $\xi_{i,max} = \frac{b_i}{2}[z_{Q,i}(1 + sgn(b_i)) + y_{Q,i}(1 - sgn(b_i))]$.
   (b) Set $\xi_{i,min} = \frac{b_i}{2}[z_{Q,i}(1 - sgn(b_i)) + y_{Q,i}(1 + sgn(b_i))]$.

Figure 5.11: An Algorithm to Calculate the Maximum and the Minimum Values of the Quadratic Expression in (5.15) for $x_i \in [y_i, z_i], i = 1, 2..., d$

- IF $a_{i,j} > 0$, THEN
  (a) Set $\zeta_{max} = \zeta_{max} + 2a_{i,j}z_{Q,i}z_{Q,j}$.
  (b) Set $\zeta_{min} = \zeta_{min} + 2a_{i,j}y_{Q,i}y_{Q,j}$.
  ELSE
  (a) Set $\zeta_{max} = \zeta_{max} + 2a_{i,j}y_{Q,i}y_{Q,j}$.
  (b) Set $\zeta_{min} = \zeta_{min} + 2a_{i,j}z_{Q,i}z_{Q,j}$.

Example 2 Estimate the maximum value ζ_{max} and the minimum value ζ_{min} of the quadratic function $f(x_1, x_2) = 2x_1^2 - x_2^2 - 2x_1x_2 - 8x_1 + 10x_2$ within the hyperrectangle $[1, 3] \times [1, 2]$ using the second algorithm.

$\because \frac{-(-8)}{2\times 2} = 2 = \frac{1+3}{2} \wedge 2 > 0$

$\therefore \xi_{1,max} = 2 \times 1^2 - 8 \times 1 = -6, \xi_{1,min} = \frac{-8^2}{4\times 2} = -8$

$\because \frac{-10}{2\times(-1)} = 5 > 2 \wedge -1 < 0$

$\therefore \xi_{2,max} = -2^2 + 10 \times 2 = 16, \xi_{2,min} = -1^2 + 10 \times 1 = 9$

$\zeta_{max} = -6 + 16 - 2 \times 1 \times 1 = 8$

$\zeta_{min} = -8 + 9 - 2 \times 3 \times 2 = -11$

The expression in (5.14) can be written as:

$$\begin{aligned} & \sum_{i=1}^{d} a_{i,i}x_i^2 + 2\sum_{i=1}^{d-1}\sum_{j=i+1}^{d} a_{i,j}x_ix_j + \sum_{i=1}^{d} b_ix_i - \gamma \\ = & \sum_{i=1}^{d} a_{i,i}x_i^2 - \sum_{i=1}^{d-1}\sum_{j=i+1}^{d} a_{i,j}(x_i^2 - 2x_ix_j + x_j^2) + \sum_{i=1}^{d} b_ix_i - \gamma \\ & + \sum_{i=1}^{d-1}\sum_{j=i+1}^{d} a_{i,j}x_i^2 + \sum_{i=1}^{d-1}\sum_{j=i+1}^{d} a_{i,j}x_j^2 \\ = & \sum_{i=1}^{d}\sum_{j=1}^{d} a_{i,j}x_i^2 + \sum_{i=1}^{d-1}\sum_{j=i+1}^{d} a_{i,j}(x_i - x_j)^2 + \sum_{i=1}^{d} b_ix_i - \gamma \\ = & \sum_{i=1}^{d}(a_i'x_i^2 + b_ix_i) + \sum_{i=1}^{d-1}\sum_{j=i+1}^{d} a_{i,j}(x_i - x_j)^2 - \gamma \end{aligned}$$

where $a_i' = \sum_{j=1}^{d} a_{i,j}$. In the third algorithm estimating the maximum value ζ_{max} and the minimum value ζ_{min} of the expression in (5.14) within the hyperrectangle H_Q, it is necessary to calculate:

- The maximum value $\xi'_{i,max}$ and the minimum value $\xi'_{i,min}$ of the following quadratic expression for $x_i \in [y_i, z_i]$, $i = 1, 2..., d$.

$$a'_i x_i^2 + b_i x_i \tag{5.16}$$

- The maximum value $\xi'_{i,j,max}$ and the minimum value $\xi'_{i,j,min}$ of the following quadratic expression for $x_i \in [y_i, z_i]$, $i = 1, 2..., d-1$, $j = i+1, i+2, ..., d$.

$$(x_i - x_j)^2 \tag{5.17}$$

The algorithm to calculate the values of $\xi'_{i,j,max}$ and $\xi'_{i,j,min}$ is described as follows:

1. Set $p_1 = 0.5(z_{Q,i} + y_{Q,i})$ and $p_2 = 0.5(z_{Q,j} + y_{Q,j})$.
2. IF $p_1 > p_2$, THEN set $v_1 = z_{Q,i} - y_{Q,j}$ ELSE set $v_1 = y_{Q,i} - z_{Q,j}$.
3. IF $z_{Q,i} < y_{Q,j}$, THEN set $v_2 = z_{Q,i} - y_{Q,j}$
   ELSE IF $y_{Q,i} > z_{Q,j}$, THEN set $v_2 = y_{Q,i} - z_{Q,j}$
   ELSE set $v_2 = 0$.
4. IF $a_{i,j} > 0$, THEN
   (a) Set $\xi'_{i,j,max} = a_{i,j} v_1^2$.
   (b) Set $\xi'_{i,j,min} = a_{i,j} v_2^2$.

   ELSE
   (a) Set $\xi'_{i,j,max} = a_{i,j} v_2^2$. 3
   (b) Set $\xi'_{i,j,min} = a_{i,j} v_1^2$.

The following describes the third algorithm estimating the maximum value ζ_{max} and the minimum value ζ_{min} of the expression in (5.14) within the hyperrectangle H_Q:

1. Set $\zeta_{max} = (\sum_{i=1}^{d} \xi'_{i,max}) - \gamma$, where $\xi'_{i,max}$ is the maximum value of the expression in (5.16) for $x_i \in [y_{Q,i}, z_{Q,i}]$, $i = 1, 2..., d$.

2. Set $\zeta_{min} = (\sum_{i=1}^{d} \xi'_{i,min}) - \gamma$, where $\xi'_{i,min}$ is the minimum value of the expression in (5.16) for $x_i \in [y_{Q,i}, z_{Q,i}]$, $i = 1, 2..., d$.

3. FOR $i = 1$ TO $d - 1$ DO
   FOR $j = i + 1$ TO d DO

   (a) Set $\zeta_{max} = \zeta_{max} + \xi'_{i,j,max}$.

   (b) Set $\zeta_{min} = \zeta_{min} + \xi'_{i,j,min}$.

Example 3 Estimate the maximum value ζ_{max} and the minimum value ζ_{min} of the quadratic function $f(x_1, x_2) = 2x_1^2 - x_2^2 - 2x_1x_2 - 8x_1 + 10x_2$ within the hyperrectangle $[1, 3] \times [1, 2]$ using the third algorithm.

$$2x_1^2 - x_2^2 - 2x_1x_2 - 8x_1 + 10x_2$$

$$= 2x_1^2 - x_2^2 - x_1^2 + (x_1^2 - 2x_1x_2 + x_2^2) - x_2^2 - 8x_1 + 10x_2$$

$$= x_1^2 - 8x_1 - 2x_2^2 + 10x_2 + (x_1 - x_2)^2$$

$$\because 0.5 \times (1 + 2) < 0.5 \times (1 + 3) \wedge 3 > 1 \wedge 1 < 2$$

$$\therefore \xi'_{1,2,max} = (1 - 3)^2 = 4, \xi'_{1,2,min} = 0$$

$$\because \frac{-(-8)}{2\times(1)} = 4 > 3 \wedge 1 > 0$$

$$\therefore \xi'_{1,max} = 1^2 - 8 \times 1 = -7, \xi'_{1,min} = 3^2 - 8 \times 3 = -15$$

$$\because \frac{-10}{2\times(-2)} = 2.5 > 2 \wedge -2 < 0$$

$$\therefore \xi'_{2,max} = -2\times2^2+10\times2 = 12, \xi'_{2,min} = -2\times1^2+10\times1 = 8$$

$$\zeta_{max} = -7 + 12 + 4 = 9$$

$$\zeta_{min} = -15 + 8 + 0 = -7$$

Figure 5.12 shows the algorithm of the procedure `processkDTreeCurve()`, which outlines the steps to determine the values of $r_{h,1}, r_{h,2}..., r_{h,C}$ using a constructed k-D tree. On the other hand, Figure 5.13 shows the algorithm of the procedure `processQuadtreeCurve()`, which outlines the steps to perform the same task using a generalized quadtree.

5.6 Performance Evaluation

In this section, the performance of applying spatial data structures to the induction of oblique and quadratic decision trees is evaluated. Two oblique decision tree algorithms, called Binary Tree-Genetic Algorithm with k-D trees (BTGA with k-D Trees) and Binary-Tree Genetic Algorithm with Quadtrees (BTGA with Quadtrees), are proposed by extending BTGA. In the BTGA with k-D Trees and the BTGA with Quadtrees, a k-D tree and a generalized quadtree are respectively constructed before finding the optimal hyperplane at each non-leaf node of a linear decision tree.

Similarly, two quadratic decision tree algorithms, called Genetic Algorithm-based Quadratic Decision Tree with k-D Trees (GA-based QDT with k-D Trees) [41] and Genetic Algorithm-based Quadratic Decision Tree with Quadtrees (GA-based QDT with Quadtrees), are introduced by extending the GA-based QDT. In the GA-based QDT with k-D Trees and the GA-based QDT with Quadtrees, a k-D tree and a generalized quadtree are respectively built before searching for the optimal quadratic decision function at each internal node of a quadratic decision tree.

GA-based QDT with k-D Trees V1, GA-based QDT with k-D Trees V2 and GA-based QDT with k-D Trees V3 are respectively

PROCEDURE `processkDTreeCurve`

INPUTS
1. A node N_Q of a k-D tree.
2. A set of training samples $S_Q = \{\mathbf{x_Q^{(1)}}, \mathbf{x_Q^{(2)}}..., \mathbf{x_Q^{(|S_Q|)}}\}$, where $\mathbf{x_Q^{(i)}} = (x_{Q,1}^{(i)}, x_{Q,2}^{(i)}..., x_{Q,d}^{(i)}, c_Q^{(i)})^T$, $i = 1, 2..., |S_Q|$, $x_{Q,1}^{(i)}, x_{Q,2}^{(i)}..., x_{Q,d}^{(i)}$ are the input attributes and $c_Q^{(i)}$ is the class label of the sample $\mathbf{x_Q^{(i)}}$.
3. The symmetric matrix $A = (a_{j,k})$ of order d specified in (4.1).
4. The d-dimensional vector $\mathbf{b} = (b_1, b_2..., b_d)^T$ specified in (4.1).
5. The real constant γ specified in (4.1).

INPUT/OUTPUT A C-dimensional vector $\mathbf{r_h} = (r_{h,1}, r_{h,2}..., r_{h,C})$, where $r_{h,i}$, $i = 1, 2..., C$, is the number of training samples of class i arriving at node N_h of a decision tree such that (4.1) is satisfied.

1. If the node N_Q is the root node of a k-D tree, set $r_{h,i} = 0$, $i = 1, 2..., C$.
2. IF $|S_Q| \leq 2$, THEN
   (a) FOR $i = 1$ TO $|S_Q|$ DO
      i. Set $\mathbf{\tilde{x}_Q^{(i)}} = (x_{Q,1}^{(i)}, x_{Q,2}^{(i)}..., x_{Q,d}^{(i)})^T$.
      ii. IF $(\mathbf{\tilde{x}_Q^{(i)}})^\mathbf{T} A \mathbf{\tilde{x}_Q^{(i)}} + \mathbf{b^T \tilde{x}_Q^{(i)}} > \gamma$, THEN set $r_{h,c_Q^{(i)}} = r_{h,c_Q^{(i)}} + 1$.
   (b) Return.
3. Estimate the maximum value ζ_{max} and the minimum value ζ_{min} of the expression in (5.14) within the hyperrectangle $H_Q = [y_{Q,1}, z_{Q,1}] \times [y_{Q,2}, z_{Q,2}]... \times [y_{Q,d}, z_{Q,d}]$.
4. IF $\zeta_{min} > 0$,
   THEN set $r_{h,i} = r_{h,i} + |S_{Q,i}|$, $i = 1, 2, ..., C$
   ELSE IF $\zeta_{max} < 0$, THEN
   IF the node N_Q is a non-leaf node, THEN
   (a) Set $S_Q^L = \{\mathbf{x} \in S_Q | x_{k_Q} < \gamma_Q\}$, where x_{k_Q} is the value of the k_Q^{th} input attribute of a training sample $\mathbf{x} \in S_Q$.
   (b) Set $S_Q^R = S_Q \setminus S_Q^L$.
   (c) Invoke the procedure `processkDTreeCurve`(L_Q, S_Q^L, A, $\mathbf{b}$, γ, $\mathbf{r_h}$).
   (d) Invoke the procedure `processkDTreeCurve`(R_Q, S_Q^R, A, $\mathbf{b}$, γ, $\mathbf{r_h}$).

   ELSE FOR $i = 1$ TO $|S_Q|$ DO
   (a) Set $\mathbf{\tilde{x}_Q^{(i)}} = (x_{Q,1}^{(i)}, x_{Q,2}^{(i)}..., x_{Q,d}^{(i)})^T$.
   (b) IF $(\mathbf{\tilde{x}_Q^{(i)}})^\mathbf{T} A \mathbf{\tilde{x}_Q^{(i)}} + \mathbf{b^T \tilde{x}_Q^{(i)}} > \gamma$, THEN set $r_{h,c_Q^{(i)}} = r_{h,c_Q^{(i)}} + 1$.

Figure 5.12: The Algorithm of the Procedure `processkDTreeCurve()`

PROCEDURE `processQuadtreeCurve`

INPUTS
1. A node N_Q of a generalized quadtree.
2. A set of training samples $S_Q = \{\mathbf{x_Q^{(1)}}, \mathbf{x_Q^{(2)}}..., \mathbf{x_Q^{(|S_Q|)}}\}$, where $\mathbf{x_Q^{(i)}} = (x_{Q,1}^{(i)}, x_{Q,2}^{(i)}..., x_{Q,d}^{(i)}, c_Q^{(i)})^T$, $i = 1, 2..., |S_Q|$, $x_{Q,1}^{(i)}, x_{Q,2}^{(i)}..., x_{Q,d}^{(i)}$ are the input attributes and $c_Q^{(i)}$ is the class label of the sample $\mathbf{x_Q^{(i)}}$.
3. The symmetric matrix $A = (a_{j,k})$ of order d specified in (4.1).
4. The d-dimensional vector $\mathbf{b} = (b_1, b_2..., b_d)^T$ specified in (4.1).
5. The real constant γ specified in (4.1).

INPUT/OUTPUT A C-dimensional vector $\mathbf{r_h} = (r_{h,1}, r_{h,2}..., r_{h,C})$, where $r_{h,i}$, $i = 1, 2..., C$, is the number of training samples of class i arriving at node N_h of a decision tree such that (5.1) is satisfied.

1. If the node N_Q is the root node of a generalized quadtree, set $r_{h,i} = 0$, $i = 1, 2..., C$.
2. IF $|S_Q| \leq 2$, THEN
   (a) FOR $i = 1$ TO $|S_Q|$ DO
      i. Set $\mathbf{\tilde{x}_Q^{(i)}} = (x_{Q,1}^{(i)}, x_{Q,2}^{(i)}..., x_{Q,d}^{(i)})^T$.
      ii. IF $(\mathbf{\tilde{x}_Q^{(i)}})^\mathbf{T} A \mathbf{\tilde{x}_Q^{(i)}} + \mathbf{b^T \tilde{x}_Q^{(i)}} > \gamma$, THEN set $r_{h,c_Q^{(i)}} = r_{h,c_Q^{(i)}} + 1$.
   (b) Return.
3. Estimate the maximum value ζ_{max} and the minimum value ζ_{min} of the expression in (5.14) within the hyperrectangle $H_Q = [y_{Q,1}, z_{Q,1}] \times [y_{Q,2}, z_{Q,2}]... \times [y_{Q,d}, z_{Q,d}]$.
4. IF $\zeta_{min} > 0$,
   THEN set $r_{h,i} = r_{h,i} + |S_{Q,i}|$, $i = 1, 2, ..., C$
   ELSE IF $\zeta_{max} < 0$, THEN
   IF the node N_Q is a non-leaf node, THEN
   (a) Let $X_{Q,i}$, $i = 1, 2..., 2^d$, be the set of training samples arriving at the i^{th} child node of the node N_Q.
   (b) FOR $i = 1$ TO 2^d DO
   IF $\Gamma_{Q,i}$ is not a null pointer, THEN invoke the procedure `processQuadtreeCurve`($\Gamma_{Q,i}$, $X_{Q,i}$, A, $\mathbf{b}$, γ, $\mathbf{r_h}$).

   ELSE FOR $i = 1$ TO $|S_Q|$ DO
   (a) Set $\mathbf{\tilde{x}_Q^{(i)}} = (x_{Q,1}^{(i)}, x_{Q,2}^{(i)}..., x_{Q,d}^{(i)})^T$.
   (b) IF $(\mathbf{\tilde{x}_Q^{(i)}})^\mathbf{T} A \mathbf{\tilde{x}_Q^{(i)}} + \mathbf{b^T \tilde{x}_Q^{(i)}} > \gamma$, THEN set $r_{h,c_Q^{(i)}} = r_{h,c_Q^{(i)}} + 1$.

Figure 5.13: The Algorithm of the Procedure `processQuadtreeCurve()`

the GA-based QDT with k-D Trees algorithms using the first, the second and the third method of estimating the maximum and the minimum values of the expression in (5.14) within a hyperrectangle. GA-based QDT with Quadtrees V1, GA-based QDT with Quadtrees V2 and GA-based QDT with Quadtrees V3 are respectively the GA-based QDT with Quadtrees algorithms using the first, the second and the third method of estimating the maximum and the minimum values of the expression in (5.14) within a hyperrectangle.

In this chapter, the experiments are divided into four parts:

- The performance of the BTGA with k-D Trees, the BTGA with Quadtrees, and all versions of the GA-based QDT with k-D Trees and the GA-based QDT with Quadtrees is compared with that of various supervised classification algorithms in terms of validation accuracy and execution time.
- The effects of modifying the minimum number of training samples at each leaf node of a k-D tree are investigated.
- The effects of changing the minimum number of training samples at each leaf node of a generalized quadtree are studied.
- The effects of changing the size of datasets are investigated.

All the experiments were executed on a dual Intel Xeon 2.2GHz machine.

5.6.1 Performance Comparison with Various Supervised Classification Algorithms

In this subsection, the performance of the BTGA with k-D Trees, the BTGA with Quadtrees, and all variants of the GA-based QDT with k-D Trees and the GA-based QDT with Quadtrees

is compared with that of various supervised classification algorithms, including C4.5, OC1, NDT, OC1-GA, OC1-ES, BTGA and GA-based QDT.

The first dataset, called ADS9, is an artificial dataset with 300 samples. ADS9 is a three-class problem. Two straight lines are used to separate the samples into three classes. Each sample is a two-dimensional vector (x_1, x_2), where $x_1, x_2 \in [0, 1000]$. If a sample satisfies (5.18), it is labeled as class 1. If a sample violates (5.18) but satisfies (5.19), it is labeled as class 2. If a sample satisfies neither (5.18) nor (5.19), it is labeled as class 3. Figure 5.14 shows the dataset ADS9.

$$-0.1x_1 + 0.9x_2 > 400 \tag{5.18}$$

$$0.2x_1 - 0.8x_2 < 350 \tag{5.19}$$

The second dataset, called ADS10, is an artificial dataset with 1000 samples. ADS10 is a three-class problem. Each sample is a two-dimensional vector (x_1, x_2), where $x_1, x_2 \in [0, 1000]$. If a sample satisfies (5.20), it is labeled as class 1. If a sample violates (5.20) but satisfies (5.21), it is labeled as class 2. If a sample satisfies neither (5.20) nor (5.21), it is labeled as class 3. Figure 5.15 shows the dataset ADS10.

$$\frac{(x_1 - 500)^2}{202500} + \frac{(x_2 - 500)^2}{122500} > 1.0 \tag{5.20}$$

$$\frac{(x_1 - 500)^2}{90000} + \frac{(x_2 - 510)^2}{62500} > 1.0 \tag{5.21}$$

The third dataset, called ADS11, is also an artificial dataset with 1000 samples. ADS11 has three possible classes. Each sample is a two-dimensional vector (x_1, x_2), where $x_1, x_2 \in [0, 1000]$. If a sample satisfies (5.22), it is labeled as class 1. If a sample violates (5.22) but satisfies (5.23), it is labeled as class 2. If a

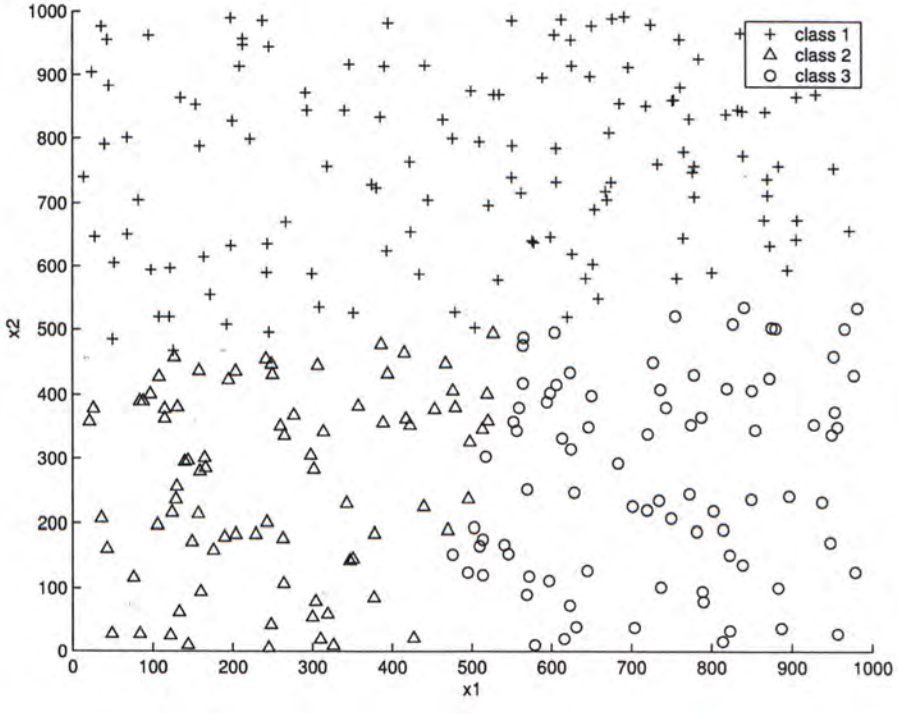

Figure 5.14: The Dataset ADS9

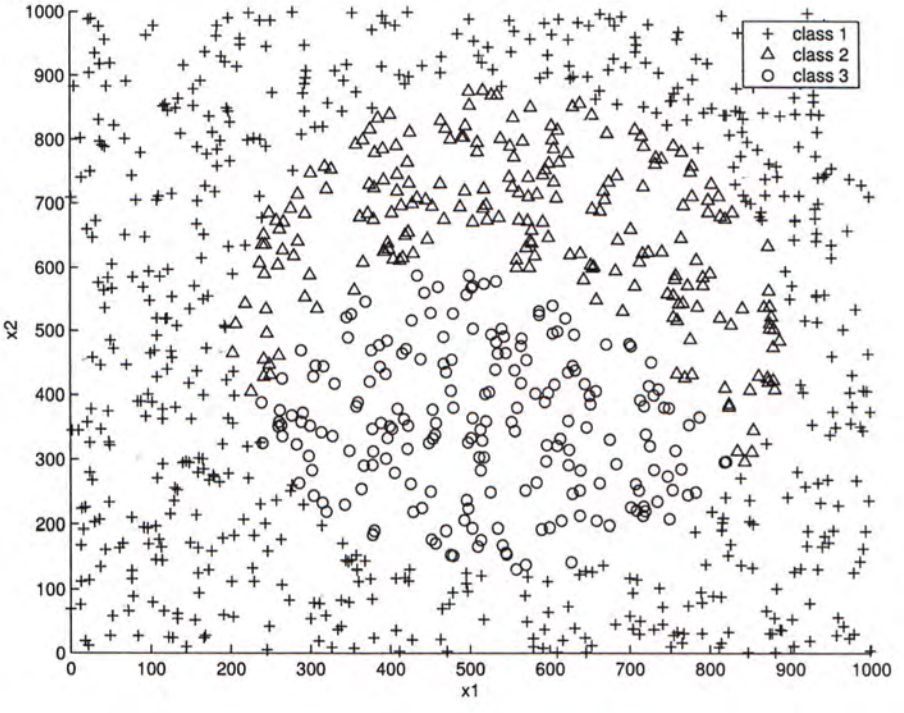

Figure 5.15: The Dataset ADS10

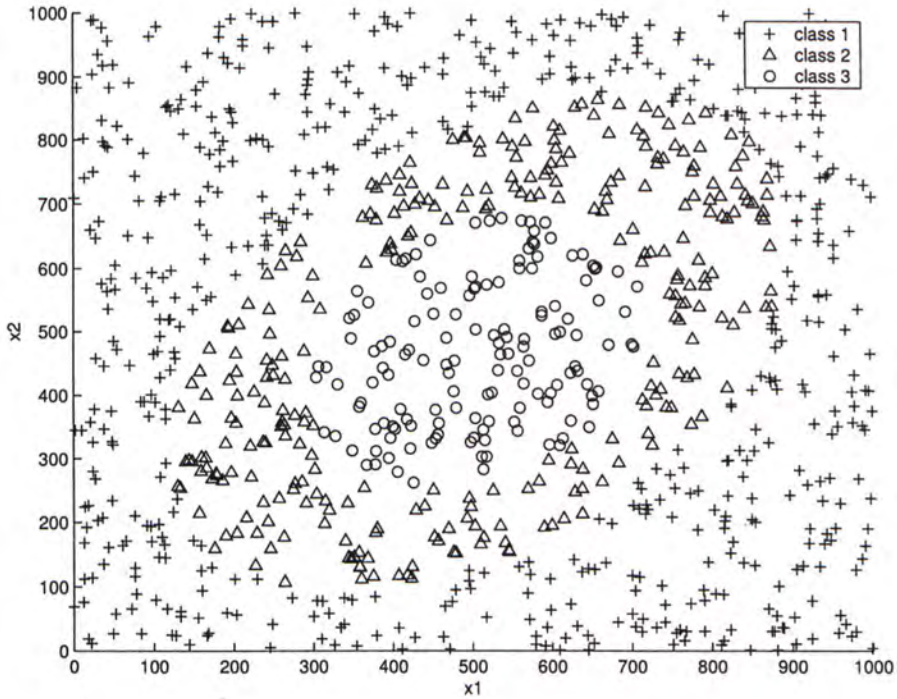

Figure 5.16: The Dataset ADS11

sample satisfies neither (5.22) nor (5.23), it is labeled as class 3. Figure 5.16 shows the dataset ADS11.

$$\frac{(x_1 - 550)^2}{122500} + \frac{(x_2 - 500)^2}{144000} > 1.0 \tag{5.22}$$

$$x_2 > 600 sin(\frac{x_1 \pi}{1000}) \tag{5.23}$$

The fourth dataset, called ECOLI, is a public domain dataset from the UCI machine learning repository. This dataset has 336 samples and 8 classes. Each sample has 7 numeric input attributes.

The fifth dataset, called BALANCE, is also a public domain dataset from the UCI machine learning repository. This dataset has 625 samples and 3 possible classes. Each sample has 4 numeric input attributes.

Dataset	Number of Generations
ADS9	60,000
ADS10	60,000
ADS11	60,000
ECOLI	300,000
BALANCE	150,000

Table 5.1: Number of Generations for OC1-ES on ADS9, ADS10, ADS11, ECOLI and BALANCE

Parameters	ADS9	ADS10	ADS11	ECOLI	BALANCE
Population Size	100				
Number of Generations	6000	2000	5000	4000	2000
Crossover Probability	0.6	0.9	1.0	0.9	0.9
Mutation Probability	0.1	0.1	0.1	0.3	0.15

Table 5.2: Parameters of OC1-GA on ADS9, ADS10, ADS11, ECOLI and BALANCE

The implementation of the OC1-ES algorithm is same as that in [8]. Table 5.1 shows the number of generations for the OC1-ES algorithm on each dataset.

The implementation of the OC1-GA algorithm in this section is same as that in Section 3.3.1. Table 5.2 shows the parameters of the OC1-GA algorithm so as to maximize its validation accuracy on each dataset.

Table 5.3 shows the parameters of the BTGA and all of its variants so that their validation accuracies are maximized on each dataset. If the number of training samples at a node is less than a positive integer n_0 or the impurity reduction is less than a threshold g_0, no child node is created in the BTGA.

Parameters	ADS9	ADS10	ADS11	ECOLI	BALANCE
Population Size	100				
Number of Generations	1000	1000	1000	4000	2000
Crossover Probability	0.7	0.9	0.9	0.8	0.8
Mutation Probability	0.05	0.05	0.1	0.15	0.1
n_0	50	15	10	15	10
g_0	0.3	0.01	0.01	0.1	0.1

Table 5.3: Parameters of BTGA and its Variants on ADS9, ADS10, ADS11, ECOLI and BALANCE

Table 5.4 shows the parameters of the GA-based QDT and all of its variants so as to maximize their validation accuracies. The value of n_0 specifies the minimum number of training samples at each node of a quadratic decision tree and the value of g_0 specifies the minimum impurity reduction. Standard parameter settings are used in various supervised classification algorithms including C4.5, OC1 and NDT.

Table 5.5 shows the average and the standard deviation of the validation accuracy of various supervised classification algorithms on ADS9, ADS10 and ADS11 when 10-fold cross-validation is applied over 10 runs. On the other hand, Table 5.6 reports the average and the standard deviation of the validation accuracy of various supervised classification algorithms on ECOLI, BALANCE when 10-fold cross-validation is applied over 10 runs.

According to the one-sided t-tests, the GA-based QDT and all of its variants outperform the others on ADS10, ADS11, ECOLI and BALANCE in terms of validation accuracy at 95% confidence interval. The decision function at each non-terminal node of a GA-based QDT and its variants usually provides a

Parameters	ADS9	ADS10	ADS11	ECOLI	BALANCE
Population Size	100				
Number of Generations	1000				
Crossover Probability	0.9	0.9	0.9	0.9	0.9
Mutation Probability	0.1	0.15	0.15	0.1	0.1
n_0	50	100	80	20	30
g_0	0.3	0.2	0.3	0.15	0.1

Table 5.4: Parameters of the GA-based QDT and its Variants on ADS9, ADS10, ADS11, ECOLI and BALANCE

better approximation to non-linear class boundaries when compared with that of univariate and oblique decision tree algorithms. When a new node is created, the GA-based QDT and its variants are more capable of finding a better quadratic decision function than NDT because they have better capability of escaping from local optima.

On the other hand, the BTGA and all of its variants outperform the others on ADS9 in terms of validation accuracy at 95% confidence interval using the one-sided t-tests. Since the class boundaries of ADS9 are linear, two straight lines can be used to partition the samples into three classes completely. Moreover, a quadratic hypersurface tends to overfit the training samples when they are linearly separable.

The validation accuracy of BTGA is same as that of the BTGA with k-D Trees and the BTGA with Quadtrees. The main difference between the BTGA and its variants is that different algorithms are used to evaluate the impurity reduction after partitioning a set of training samples into two disjoint subsets, although the same result is obtained using either the BTGA

Algorithm	ADS9	ADS10	ADS11
C4.5	95.6 ± 0.7	95.1 ± 0.3	94.0 ± 0.6
OC1	97.4 ± 0.8	95.3 ± 0.8	95.5 ± 0.7
NDT	97.1 ± 0.5	95.5 ± 0.6	95.1 ± 0.4
OC1-GA	95.9 ± 0.9	94.6 ± 0.7	94.8 ± 0.4
OC1-ES	98.3 ± 0.4	95.7 ± 0.6	95.2 ± 0.7
BTGA	98.7 ± 0.5	96.5 ± 0.7	95.9 ± 0.4
GA-based QDT	98.1 ± 0.5	99.2 ± 0.3	98.6 ± 0.4
BTGA with k-D Trees	98.7 ± 0.5	96.5 ± 0.7	95.9 ± 0.4
BTGA with Quadtrees	98.7 ± 0.5	96.5 ± 0.7	95.9 ± 0.4
GA-based QDT with k-D Trees V1	98.1 ± 0.5	99.2 ± 0.3	98.6 ± 0.4
GA-based QDT with k-D Trees V2	98.1 ± 0.5	99.2 ± 0.3	98.6 ± 0.4
GA-based QDT with k-D Trees V3	98.1 ± 0.5	99.2 ± 0.3	98.6 ± 0.4
GA-based QDT with Quadtrees V1	98.1 ± 0.5	99.2 ± 0.3	98.6 ± 0.4
GA-based QDT with Quadtrees V2	98.1 ± 0.5	99.2 ± 0.3	98.6 ± 0.4
GA-based QDT with Quadtrees V3	98.1 ± 0.5	99.2 ± 0.3	98.6 ± 0.4

Table 5.5: Average and Standard Deviation of Validation Accuracy (%) of Various Supervised Classification Algorithms on ADS9, ADS10 and ADS11 based on 10 Independent Runs

or one of its variants. On the other hand, the GA-based QDT and all of its variants construct the same quadratic decision tree when the same set of parameters is applied.

Table 5.7 shows the average and the standard deviation of the execution time of various supervised classification algorithms on ADS9, ADS10 and ADS11 when 10-fold cross-validation is applied over 10 runs. Table 5.8 shows the average and the standard deviation of the execution time of various supervised classification algorithms on ECOLI and BALANCE when 10-fold cross-validation is applied over 10 runs.

The BTGA with k-D Trees and the BTGA with Quadtrees run faster than the BTGA on all datasets. Note that the maxi-

Algorithm	ECOLI	BALANCE
C4.5	81.6 ± 1.2	77.6 ± 0.7
OC1	80.7 ± 1.8	91.1 ± 0.6
NDT	81.1 ± 1.8	91.8 ± 1.1
OC1-GA	83.6 ± 1.3	93.9 ± 1.2
OC1-ES	80.6 ± 2.0	90.7 ± 0.9
BTGA	83.6 ± 1.4	93.1 ± 1.0
GA-based QDT	84.9 ± 0.7	97.2 ± 0.5
BTGA with k-D Trees	83.6 ± 1.4	93.1 ± 1.0
BTGA with Quadtrees	83.6 ± 1.4	93.1 ± 1.0
GA-based QDT with k-D Trees V1	84.9 ± 0.7	97.2 ± 0.5
GA-based QDT with k-D Trees V2	84.9 ± 0.7	97.2 ± 0.5
GA-based QDT with k-D Trees V3	84.9 ± 0.7	97.2 ± 0.5
GA-based QDT with Quadtrees V1	84.9 ± 0.7	97.2 ± 0.5
GA-based QDT with Quadtrees V2	84.9 ± 0.7	97.2 ± 0.5
GA-based QDT with Quadtrees V3	84.9 ± 0.7	97.2 ± 0.5

Table 5.6: Average and Standard Deviation of Validation Accuracy (%) of Various Supervised Classification Algorithms on ECOLI and BALANCE based on 10 Independent Runs

mum and the minimum values of the expression in (5.9) within a hyperrectangle can be calculated accurately. The time required to calculate these values within a hyperrectangle is approximately equal to twice the time required to find the sign of the expression in (5.9) for a training sample. When there are not too few training samples at all leaf nodes of a k-D tree or a generalized quadtree, the BTGA with k-D Trees and the BTGA with Quadtrees run faster than the BTGA. Among the BTGA and its variants, the BTGA with k-D Trees run the fastest on ECOLI and BALANCE. The BTGA with Quadtrees run the fastest on ADS9, ADS10 and ADS11.

All versions of the GA-based QDT with k-D Trees run faster than the GA-based QDT on ADS9, ADS10, ADS11 and ECOLI. The execution time of the GA-based QDT with k-D Trees V3 is longer than that of the other versions of the GA-based QDT with k-D Trees on ADS9, ECOLI and BALANCE. On the other hand, all versions of the GA-based QDT with Quadtrees run faster than the GA-based QDT on ADS9, ADS10 and ADS11. The execution time of the GA-based QDT with Quadtrees V3 is longer than that of the other versions of the GA-based QDT with Quadtrees on ADS9, ECOLI and BALANCE.

Among the GA-based QDT and all of its variants, the GA-based QDT with k-D Trees V2 run the fastest on ADS10 and BALANCE. The GA-based QDT with Quadtrees V1 run the fastest on ADS9. The GA-based QDT with Quadtrees V2 run the fastest on ECOLI. The GA-based QDT with Quadtrees V3 run the fastest on ADS11. The validation accuracy of each variant of the GA-based QDT is identical to that of each other for all datasets.

Algorithm	ADS9	ADS10	ADS11
C4.5	< 1	< 1	< 1
OC1	4.1 ± 1.0	32.2 ± 1.8	33.2 ± 1.8
NDT	2.4 ± 0.5	24.2 ± 1.2	25.0 ± 2.4
OC1-GA	21.5 ± 1.4	265.1 ± 5.1	292.3 ± 3.2
OC1-ES	24.9 ± 0.3	256.8 ± 5.8	286.5 ± 2.1
BTGA	17.2 ± 1.3	467.3 ± 6.2	265.7 ± 1.6
BTGA with k-D Trees	12.0 ± 0.5	264.4 ± 2.7	131.8 ± 1.2
BTGA with Quadtrees	11.0 ± 0.1	257.2 ± 2.5	129.6 ± 1.5
GA-based QDT	41.0 ± 0.9	219.1 ± 1.2	216.5 ± 1.0
GA-based QDT with k-D Trees V1	19.4 ± 1.0	163.4 ± 5.2	170.0 ± 5.7
GA-based QDT with k-D Trees V2	19.3 ± 0.8	83.4 ± 2.3	117.8 ± 3.6
GA-based QDT with k-D Trees V3	21.8 ± 6.3	103.6 ± 3.0	102.6 ± 1.8
GA-based QDT with Quadtrees V1	18.1 ± 0.6	150.4 ± 1.4	157.1 ± 2.5
GA-based QDT with Quadtrees V2	19.8 ± 1.5	96.2 ± 5.8	124.5 ± 11.2
GA-based QDT with Quadtrees V3	22.0 ± 2.1	94.8 ± 5.2	94.8 ± 2.0

Table 5.7: Average and Standard Deviation of Execution Time (in Seconds) of Various Supervised Classification Algorithms on ADS9, ADS10 and ADS11 based on 10 Independent Runs

Algorithm	ECOLI	BALANCE
C4.5	< 1	< 1
OC1	73.7 ± 2.1	57.8 ± 0.6
NDT	142.0 ± 2.7	90.3 ± 2.6
OC1-GA	1015 ± 27	471.2 ± 30.0
OC1-ES	1021 ± 38	476.1 ± 12.9
BTGA	1007 ± 28	485.7 ± 67.3
GA-based QDT	891.6 ± 14.0	366.7 ± 5.1
BTGA with k-D Trees	665.2 ± 14.9	218.6 ± 7.4
BTGA with Quadtrees	783.4 ± 19.5	222.0 ± 7.9
GA-based QDT with k-D Trees V1	837.3 ± 13.4	284.4 ± 9.6
GA-based QDT with k-D Trees V2	840.8 ± 12.5	300.1 ± 11.5
GA-based QDT with k-D Trees V3	861.5 ± 13.5	371.6 ± 8.9
GA-based QDT with Quadtrees V1	784.4 ± 16.0	310.2 ± 12.2
GA-based QDT with Quadtrees V2	778.6 ± 16.7	308.5 ± 14.5
GA-based QDT with Quadtrees V3	915.6 ± 15.2	372.3 ± 21.2

Table 5.8: Average and Standard Deviation of Execution Time (in Seconds) of Various Supervised Classification Algorithms on ECOLI and BALANCE based on 10 Independent Runs

N_Q	Execution Time (s)
4	22.2 ± 0.7
8	18.3 ± 0.8
16	14.9 ± 0.6
32	12.8 ± 0.6
64	12.0 ± 0.5
128	12.3 ± 0.5
256	14.2 ± 0.6

Table 5.9: Average and Standard Deviation of Execution Time (in Seconds) of BTGA with k-D Trees on ADS9 based on 10 Independent Runs when the Minimum Number of Training Samples N_Q at Each Node of a k-D Tree Varies

5.6.2 Effects of Changing the Minimum Number of Training Samples at Each Node of a k-D Tree

In this subsection, the effect of changing the minimum number of training samples at each node of a k-D tree is investigated in terms of execution time.

BTGA with k-D Trees

ADS9 is chosen to investigate the effect of changing the minimum number of training samples N_Q at each node of a k-D tree in the BTGA with k-D Trees. Table 5.9 shows the average and the standard deviation of the execution time of the BTGA with k-D Trees as the minimum number of training samples N_Q at each node of a k-D tree varies. The minimum execution time is attained when $N_Q = 64$. The execution time of the BTGA with k-D Trees is longer than that of BTGA when $N_Q = 4, 8$. (See Table 5.7) However, the validation accuracy of BTGA with k-D trees is same as that of BTGA for all values of N_Q.

N_Q	GA-based QDT with k-D Trees		
	V1	V2	V3
4	294.4 ± 9.2	323.6 ± 15.6	570.4 ± 23.2
8	284.4 ± 9.6	300.1 ± 11.5	498.0 ± 20.1
16	311.2 ± 11.1	309.6 ± 15.2	404.1 ± 16.6
32	330.4 ± 12.2	313.8 ± 9.4	380.2 ± 15.8
64	343.0 ± 10.5	351.8 ± 10.1	371.6 ± 8.9

Table 5.10: Average and Standard Deviation of Execution Time (in Seconds) of all Versions of the GA-based QDT with k-D Trees on BALANCE based on 10 Independent Runs when the Minimum Number of Training Samples N_Q at Each Node of a k-D Tree Varies

GA-based QDT with k-D Trees

The BALANCE dataset is selected to investigate the effect of modifying the minimum number of training samples N_Q at each node of a k-D tree in all versions of the GA-based QDT with k-D Trees. Table 5.10 shows the average and the standard deviation of the execution time of all versions of the GA-based QDT with k-D Trees as the minimum number of training samples N_Q at each node of a constructed k-D tree varies. The minimum execution time is attained when $N_Q = 8$ for the GA-based QDT with k-D Trees V1 and the GA-based QDT with k-D Trees V2 and $N_Q = 64$ for the GA-based QDT with k-D Trees V3. The execution time of the GA-based QDT with k-D Trees V3 is longer than that of the GA-based QDT. (See Table 5.8) The reason is that the time required to estimate the maximum and the minimum values of the expression in (5.14) using the third algorithm is so long that the computational overhead caused by processing a k-D tree cannot be completely compensated by reducing the number of tests on the decision function in (4.1). Nevertheless, the validation accuracy of the GA-based QDT is same as that of all variants of the GA-based QDT with k-D Trees.

5.6.3 Effects of Changing the Minimum Number of Training Samples at Each Node of a Generalized Quadtree

In this subsection, the effect of modifying the minimum number of training samples at each node of a generalized quadtree is investigated in terms of execution time.

BTGA with Quadtrees

ADS9 is chosen to investigate the effect of changing the minimum number of training samples N_Q at each node of a generalized quadtree in the BTGA with Quadtrees. Table 5.11 shows the average and the standard deviation of the execution time of the BTGA with Quadtrees as the minimum number of training samples N_Q at each node of a generalized quadtree varies. The minimum execution time is attained when $N_Q = 128$. The execution time of the BTGA with Quadtrees is longer than that of the BTGA when $N_Q = 4$. (See Table 5.7) However, the validation accuracy of the BTGA with Quadtrees is same as that of BTGA no matter what the value of N_Q is.

GA-based QDT with Quadtrees

The BALANCE dataset is selected to study the effect of changing the minimum number of training samples N_Q at each node of a generalized quadtree in all versions of the GA-based QDT with Quadtrees. Table 5.12 shows the average and the standard deviation of the execution time of all versions of the GA-based QDT with Quadtrees as the minimum number of training samples N_Q at each node of a generalized quadtree varies. The minimum execution time is attained when $N_Q = 32$ for the GA-based QDT with Quadtrees V1 and the GA-based QDT with Quadtrees V2 and $N_Q = 128$ for the GA-based QDT with Quadtrees V3. The execution time of the GA-based QDT with Quadtrees V3 is

N_Q	Execution Time (s)
4	21.9 ± 0.8
8	17.1 ± 0.3
16	15.1 ± 0.7
32	12.0 ± 0.5
64	11.3 ± 0.5
128	11.0 ± 0.1
256	11.1 ± 0.1

Table 5.11: Average and Standard Deviation of Execution Time (in Seconds) of BTGA with Quadtrees on ADS9 based on 10 Independent Runs when the Minimum Number of Training Samples N_Q at Each Node of a Generalized Quadtree Varies

longer than that of the GA-based QDT. (See Table 5.8) Nevertheless, the validation accuracy of the GA-based QDT is same as that of all versions of the GA-based QDT with Quadtrees.

5.6.4 Effects of Changing the Size of Datasets

In this subsection, the effect of changing the size of datasets is investigated in terms of execution time.

BTGA and its Variants

In this part, ADS9 is replicated N_C times. Table 5.13 shows the average and the standard deviation of the execution time of the BTGA and its variants as the number of replications N_C varies. The execution time of the BTGA with k-D Trees and the BTGA with Quadtrees is lower than that of the BTGA. The BTGA with Quadtrees runs faster than the BTGA with k-D Trees. The percentage decrease in the execution time of the BTGA with k-D Trees and that of the BTGA with Quadtrees when compared with that of the BTGA increases as the number of replications N_C increases. When the size of a dataset is increased, the BTGA

N_Q	GA-based QDT with Quadtrees		
	V1	V2	V3
4	329.8 ± 10.2	324.6 ± 9.4	420.0 ± 11.1
8	322.6 ± 10.1	318.1 ± 9.8	412.8 ± 10.5
16	319.2 ± 9.8	314.6 ± 9.1	409.8 ± 11.4
32	310.2 ± 12.2	308.5 ± 14.5	385.8 ± 9.5
64	318.2 ± 9.3	312.9 ± 8.9	378.6 ± 10.3
128	322.6 ± 9.5	316.6 ± 8.6	372.3 ± 21.2

Table 5.12: Average and Standard Deviation of Execution Time (in Seconds) of all Versions of the GA-based QDT with Quadtrees on BALANCE based on 10 Independent Runs when the Minimum Number of Training Samples N_Q at Each Node of a Generalized Quadtree Varies

with k-D Trees and the BTGA with Quadtrees are more likely to run faster than the BTGA. The validation accuracy of BTGA is same as that of the BTGA with k-D trees and the BTGA with Quadtrees for all possible values of N_C.

GA-based QDT and its Variants

In this part, the BALANCE dataset is replicated N_C times. Table 5.14 shows the average and the standard deviation of the execution time of the GA-based QDT and all versions of the GA-based QDT with k-D Trees as the number of replications N_C varies. Table 5.15 reports the average and the standard deviation of the execution time of all versions of the GA-based QDT with Quadtrees as the number of replications N_C varies. The percentage change in the execution time of all versions of the GA-based QDT with k-D Trees when compared with that of the GA-based QDT decreases as the number of replications N_C increases. Similarly, the percentage change in the execution time of all versions of the GA-based QDT with Quadtrees when compared with that of the GA-based QDT decreases as the num-

N_C	BTGA	BTGA with k-D Trees	BTGA with Quadtrees
1	17.2 ± 1.3	12.0 ± 0.5	11.0 ± 0.1
2	26.9 ± 0.6	16.1 ± 0.5	14.5 ± 0.5
3	38.8 ± 1.3	18.6 ± 0.7	17.2 ± 0.4
4	50.1 ± 0.9	21.7 ± 0.7	19.4 ± 0.5
5	60.9 ± 1.0	22.9 ± 0.6	20.4 ± 0.5
6	73.2 ± 1.4	25.6 ± 0.7	21.9 ± 0.7
7	84.8 ± 2.7	26.9 ± 0.9	23.0 ± 0.8
8	96.3 ± 2.5	27.5 ± 1.0	23.9 ± 1.0
9	107.1 ± 2.5	27.7 ± 0.7	24.7 ± 0.7
10	117.6 ± 2.0	28.8 ± 0.7	25.4 ± 0.9

Table 5.13: Average and Standard Deviation of Execution Time (in Seconds) of BTGA and its Variants based on 10 Independent Runs when the Number of Replications N_C on ADS9 Varies

ber of replications N_C increases. Although the GA-based QDT with k-D Trees V3 runs slower than the GA-based QDT when $N_C = 1$, this is not the case when $N_C > 1$. The validation accuracy of the GA-based QDT is same as that of all of its variants no matter what the value of N_C is.

5.7 Chapter Summary

Spatial data structures, including k-D trees and generalized quadtrees, can be used to speed up the construction of an oblique or a quadratic decision tree when the size of a dataset is sufficiently large, without deteriorating the quality of a constructed decision tree.

In this chapter, the structures of a k-D tree and a generalized quadtree have been introduced. Several possible algorithms to construct oblique and quadratic decision trees using k-D trees and generalized quadtrees have been proposed.

N_C	GA-based QDT with k-D Trees			GA-based QDT
	V1	V2	V3	
1	354.0 ± 9.6	350.1 ± 10.5	387.2 ± 10.5	366.7 ± 5.1
2	588.1 ± 13.9	574.2 ± 12.3	668.4 ± 13.5	718.4 ± 12.5
3	806.6 ± 22.5	779.2 ± 23.5	926.8 ± 27.1	1092.6 ± 33.9
4	1001.2 ± 21.5	957.8 ± 18.8	1149.8 ± 20.3	1418.9 ± 25.9
5	1154.3 ± 15.6	1109.2 ± 14.3	1354.1 ± 11.6	1741.9 ± 25.1
6	1285.3 ± 18.2	1245.6 ± 18.7	1545.7 ± 19.8	2118.0 ± 30.3
7	1421.1 ± 12.8	1371.2 ± 25.2	1721.6 ± 20.1	2440.8 ± 33.8
8	1548.4 ± 20.1	1491.2 ± 16.9	1870.6 ± 20.1	2746.5 ± 25.4
9	1688.2 ± 26.2	1608.6 ± 23.1	2047.8 ± 17.8	3164.5 ± 51.3
10	1780.1 ± 39.2	1692.5 ± 33.4	2188.6 ± 36.1	3572.9 ± 129.3

Table 5.14: Average and Standard Deviation of Execution Time (in Seconds) of the GA-based QDT and all Versions of the GA-based QDT with k-D Trees based on 10 Independent Runs when the Number of Replications N_C on BALANCE Varies

N_C	GA-based QDT with Quadtrees			GA-based QDT
	V1	V2	V3	
1	320.4 ± 11.8	311.0 ± 7.5	358.2 ± 10.1	366.7 ± 5.1
2	500.1 ± 10.9	486.8 ± 9.5	626.6 ± 8.5	718.4 ± 12.5
3	538.2 ± 12.3	543.2 ± 13.2	830.2 ± 12.5	1092.6 ± 33.9
4	615.8 ± 10.5	615.4 ± 8.6	1006.3 ± 20.4	1418.9 ± 25.9
5	667.8 ± 9.9	647.3 ± 12.5	1076.0 ± 16.8	1741.9 ± 25.1
6	750.8 ± 11.4	734.2 ± 10.8	1133.6 ± 16.9	2118.0 ± 30.3
7	846.6 ± 15.3	833.0 ± 16.0	1220.2 ± 21.3	2440.8 ± 33.8
8	907.4 ± 12.1	884.8 ± 12.6	1322.0 ± 15.0	2746.5 ± 25.4
9	933.6 ± 12.4	924.2 ± 13.8	1377.6 ± 18.7	3164.5 ± 51.3
10	942.6 ± 17.2	935.8 ± 17.3	1422.6 ± 15.6	3572.9 ± 129.3

Table 5.15: Average and Standard Deviation of Execution Time (in Seconds) of the GA-based QDT and all Versions of the GA-based QDT with Quadtrees based on 10 Independent Runs when the Number of Replications N_C on BALANCE Varies

Binary Tree-Genetic Algorithm (BTGA) is chosen to evaluate the performance when an oblique decision tree is constructed with the aid of k-D trees and generalized quadtrees.On the other hand, Genetic Algorithm-based Quadratic Decision Tree (GA-based QDT) is selected to evaluate whether it is suitable to construct a quadratic decision tree using k-D trees and generalized quadtrees.

To construct a quadratic decision tree for classifications, it is necessary to estimate the maximum and the minimum values of a quadratic function within a hyperrectangle when the optimality of a quadratic decision function is determined using a k-D tree or a generalized quadtree. Three methods to estimate the maximum and the minimum values of a quadratic function within a hyperrectangle are introduced. The execution time of the variants of the GA-based QDT depends on the quality of the estimated maximum and minimum values of a quadratic function within a hyperrectangle and the time required to estimate these values. Although all of these methods may overestimate its maximum value and underestimate its minimum value, they neither underestimate its maximum value nor overestimate its minimum value, and therefore we can preserve the classification accuracy of the constructed quadratic decision tree. To design a suitable algorithm to estimate the maximum and the minimum values of a quadratic function within a hyperrectangle, it is necessary to strike the right balance between the quality of the estimated values and the estimation time.

The effects of modifying the minimum number of training samples at each node of a k-D tree and a generalized quadtree on the execution time of the variants of the BTGA and that of the GA-based QDT have been investigated. Experiments show that the minimum number of training samples at each node of a k-D tree or a generalized quadtree should not be too small or too large in order to minimize the execution time. Moreover, the

effects of increasing the size of datasets on the execution time of the variants of the BTGA and that of the GA-based QDT have been studied. Experiments show that the variants of the BTGA and that of the GA-based QDT are more effective in reducing their computational time as the size of datasets increases.

□ **End of chapter.**

Chapter 6

Conclusions

In this chapter, the contributions of the thesis are concluded. In addition, we provide the possible future research directions of the thesis.

6.1 Contributions

A rule induction algorithm called SCION was introduced by Leung et al. [33], [34]. The antecedent part of a classification rule is equivalent to a hyperrectangle in the attribute space. A genetic algorithm-based rule induction algorithm, called Genetic Algorithm-based Convex Polytope Rule Learning System (GA-based CPRLS), has been proposed by extending SCION. In the GA-based CPRLS, the antecedent part of a classification rule is a conjunctive set of logical expressions. The possible logical expressions include linear inequalities with several continuous attributes and nominal attribute-value pairs. The GA-based CPRLS provides an alternative rule learning algorithm. Experiments show that the GA-based CPRLS generates a better set of classification rules than SCION on datasets with non-axis parallel class boundaries. GAs are more capable of finding a better rule set than random search. Token competition and rule migration improves the performance of the GA-based CPRLS.

A genetic-algorithm based quadratic decision tree algorithm, called Genetic Algorithm-based Quadratic Decision Tree (GA-based QDT), has been proposed by extending Binary Tree-Genetic Algorithm (BTGA) [9]. At each non-leaf node of a quadratic decision tree, GAs are applied to search for the optimal quadratic decision function. A linear function is a special case of a quadratic function. Experiments show that the GA-based QDT provides a better decision tree classifier than univariate and linear decision trees on datasets with non-linear class boundaries. In addition, GAs have higher capability of searching for a better quadratic decision function than random search at each internal node of a quadratic decision tree, and therefore a better quadratic decision tree can be constructed.

We have proposed to construct a k-D tree or a generalized quadtree before searching for the optimal linear or quadratic decision function at each internal node of a linear or a quadratic decision tree. The time required to construct an oblique or a quadratic decision tree can be reduced when the size of a dataset is sufficiently large, without deteriorating the quality of a constructed decision tree.

In order to construct a linear or a quadratic decision tree with the aid of k-D trees or generalized quadtrees, it is necessary to calculate the maximum and the minimum values of a linear or a quadratic function within a hyperrectangle. We can accurately calculate the maximum and the minimum values of a linear function within a hyperrectangle. However, it is difficult to evaluate the maximum and the minimum values of a quadratic function within a hyperrectangle accurately. In this thesis, we have suggested three methods to estimate the maximum and the minimum values of a quadratic function within a hyperrectangle. Although the overall execution time can only be reduced significantly when a quadratic decision tree is built with the aid of k-D trees or generalized quadtrees provided that the

data size is large, the classification accuracy of the constructed decision tree is not sacrificed because the impurity reduction is still evaluated accurately.

Two linear decision tree algorithms, called Binary Tree-Genetic Algorithm with k-D trees (BTGA with k-D Trees) and Binary-Tree Genetic Algorithm with Quadtrees (BTGA with Quadtrees), have been proposed by extending BTGA. In the BTGA with k-D Trees and the BTGA with Quadtrees, a k-D tree and a generalized quadtree are respectively constructed before searching for the optimal linear decision function at each internal node of a linear decision tree. Experiments show that the classification accuracy of BTGA is same as that of the BTGA with k-D trees and the BTGA with Quadtrees.

Similarly, two quadratic decision tree algorithms, called Genetic Algorithm-based Quadratic Decision Tree with k-D Trees (GA-based QDT with k-D Trees) and Genetic Algorithm-based Quadratic Decision Tree with Quadtrees (GA-based QDT with Quadtrees), have been introduced by extending the GA-based QDT. In the GA-based QDT with k-D Trees and the GA-based QDT with Quadtrees, a k-D tree and a generalized quadtree are respectively constructed before searching for the optimal quadratic hypersurface at each non-terminal node of a quadratic decision tree. Three methods of estimating the maximum and the minimum values of a quadratic function within a hyperrectangle have been introduced. GA-based QDT with k-D Trees V1, GA-based QDT with k-D Trees V2 and GA-based QDT with k-D Trees V3 are respectively the GA-based QDT with k-D Trees algorithms using the first, the second and the third method of estimating the maximum and the minimum values of a quadratic function within a hyperrectangle. GA-based QDT with Quadtrees V1, GA-based QDT with Quadtrees V2 and GA-based QDT with Quadtrees V3 are respectively the GA-based QDT with Quadtrees algorithms using the first, the second and

the third method of estimating the maximum and the minimum values of a quadratic function within a hyperrectangle. To design a suitable algorithm to estimate the minimum and the maximum values of a quadratic function within a hyperrectangle, it is necessary to strike the balance between the quality of the estimated values and the estimation time. Nevertheless, the classification accuracy of the GA-based QDT is same as that of all of its variants.

The effects of changing the minimum number of training samples at each node of a k-D tree and a generalized quadtree of the variants of BTGA and the GA-based QDT have been investigated. The minimum number of training samples at each node of a k-D tree or a generalized quadtree should be carefully adjusted in order to minimize the execution time. However, the validation accuracy of all variants of BTGA and the GA-based QDT remains unchanged when the minimum number of training samples at each node of a k-D tree or a generalized quadtree is changed. The effects of increasing the size of datasets on the execution time of the variants of BTGA and the GA-based QDT have been studied. Experiments show that the variants of BTGA and that of the GA-based QDT are more effective in reducing their execution time as the size of datasets increases.

6.2 Future Work

In the GA-based CPRLS, the number of linear inequalities in the antecedent part of a classification rule needs to be specified. However, the optimal number of linear inequalities in a classification rule varies from problem to problem. When the number of linear inequalities is too small, two or more classification rules may be required to represent a cluster in the attribute space. When there are too many linear inequalities to represent a classification rule, the classification rule may overfit the train-

ing samples. An inappropriate number of linear inequalities may result in poor generalization. The determination of the optimal number of linear inequalities in a classification rule is a possible research direction of the GA-based CPRLS.

The experiments in Section 4.4.1 show that a quadratic hypersurface tends to overfit the training samples with linear class boundaries, although a hyperplane is a special case of a quadratic hypersurface. It is needed to study the possible ways to evolve the optimal quadratic hypersurface so that the impurity reduction after dividing a set of training samples into two disjoint subsets is maximized while the number of non-zero coefficients is minimized. In other words, the number of terms of a decision function should be minimized.

To evaluate the optimality of a quadratic hypersurface using a k-D tree or a generalized quadtree, it is necessary to estimate the maximum and the minimum values of a quadratic function within the smallest hyperrectangle containing a set of training samples. In order to estimate its maximum and minimum values more accurately, more computational time is usually required. New algorithms estimating the maximum and the minimum values of a quadratic function within a hyperrectangle can be explored so that the quality of these estimated values is improved while the computational time is minimized.

In all of the variants of BTGA and the GA-based QDT, the minimum number of training samples at each leaf node of a constructed k-D tree or a generalized quadtree is defined in advance. The experiments reported in Section 5.6.3 conclude that it is necessary to carefully choose the minimum number of training samples at each leaf node of a k-D tree or a generalized quadtree. It is worthwhile to study the possible ways to determine the optimal minimum number of training samples at each leaf node of a k-D tree or a generalized quadtree so as to minimize the execution time.

In all of the proposed algorithms in the thesis, the values of all of the input attributes of each sample need to be specified. When the value of an input attribute of a sample is unspecified, the GA-based CPRLS cannot determine whether the sample satisfies the antecedent part of a classification rule. Both the GA-based QDT and BTGA are incapable of choosing the appropriate terminal node when a sample with one or more missing attribute values is encountered. However, most real life datasets contain one or more samples whose values of one or more of the input attributes are missing. It is necessary to investigate how to handle a sample with one or more missing values in the proposed algorithms.

□ **End of chapter.**

Appendix A

Implementation of Data Mining Algorithms Specified in the Thesis

In order to evaluate the performance of the proposed algorithms in the thesis, it is necessary to compare the performance of other data mining algorithms with that of the proposed algorithms. This part describes how to implement other data mining algorithms mentioned in the Thesis.

C4.5 The source code of the C4.5 algorithm can be downloaded in the following website:
http://www2.cs.uregina.ca/~dbd/cs831/notes/ml/dtrees/c4.5/c4.5r8.tar.gz

OC1 The source code of the OC1 algorithm can be downloaded in the website http://www.tigr.org/%7Esalzberg/OC1.tar.gz.

NDT Since NDT is extended from OC1, it was implemented by modifying the source code of the OC1 algorithm downloaded in the Internet

OC1-ES Since OC1-ES is extended from OC1, it was implemented by modifying the source code of the OC1 algorithm downloaded in the Internet.

OC1-GA Since OC1-GA is extended from OC1, it was implemented by modifying the source code of the OC1 algorithm downloaded in the Internet.

SCION A program for SCION was written.

BTGA A program for BTGA was written.

□ **End of chapter.**

Bibliography

[1] J. Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pages 101–111. Lawrence Erlbaum, 1985.

[2] K. P. Bennett, N. Cristianini, J. Shawe-Taylor, and D. Wu. Enlarging the margins in perceptron decision trees. *Machine Learning*, 41(3):295–313, 2000.

[3] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, Sept. 1975.

[4] M. C. J. Bot. Improving induction of linear classification trees with genetic programming. In D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 403–410, Las Vegas, Nevada, USA, 10–12 2000. Morgan Kaufmann.

[5] M. C. J. Bot and W. B. Langdon. Application of genetic programming to induction of linear classification trees. In R. Poli, W. Banzhaf, W. B. Langdon, J. F. Miller, P. Nordin, and T. C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802, pages 247–258, Edinburgh, 15-16 2000. Springer-Verlag.

[6] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth International Group, 1984.

[7] C. E. Brodley and P. E. Utgoff. Multivariate decision trees. Technical Report UM-CS-1992-083, 1992.

[8] E. Cantú-Paz and C. Kamath. Inducing oblique decision trees with evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 7(1):54–68, Feb. 2003.

[9] B.-B. Chai, T. Huang, X. Zhuang, Y. Zhao, and J. Sklansky. Piecewise linear classifiers using binary tree structure and genetic algorithm. *Pattern Recognition*, 29(11):1905–1917, 1996.

[10] P. Clak and T. Niblett. The cn2 induction algorithm. *Machine Learning*, 3:261–283, 1989.

[11] L. Davis. *Handbook of Genetic Algorithm.* New York : Van Nostrand Reinhold, 1991.

[12] D. Dumitrescu, B. Lazzerini, L. Jain, and A. Dumitrescu. *Evolutionary Computation.* CRC Press, 2000.

[13] L. J. Eshelman and J. D. Shaffer. Real coded genetic algorithms and interval schemata. In *Foundations of genetic algorithms II*, pages 187–202. Morgan Kaufmann, 1993.

[14] D. B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Network*, 5:3–14, 1994.

[15] L. Fogel, A. Owens, and M. Walsh. *Artificial Intelligence through Simulated Evolution.* John Wiley and Sons, 1966.

[16] A. A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms.* New York: Springer-Verlag, 2002.

[17] J. Gama. Oblique linear tree. *Lecture Notes in Computer Science*, 1280:187–198, 1997.

[18] C. Gathercole and P. Ross. Tackling the boolean even N parity problem with genetic programming and limited-error fitness. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 119–127, Stanford University, CA, USA, 13-16 1997. Morgan Kaufmann.

[19] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[20] M. Goodrich, V. Mirelli, M. Orletsky, and J. Salowe. Decision tree construction in fixed dimensions: Being global is hard but local greed is good, 1995.

[21] J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers, 2001.

[22] I. Harvey. Evolutionary robotics and saga: The case for hill crawling and tournament selection. In *Proceedings of Workshop on Artificial Life*, volume XVI, pages 299–326. Santa Fe, NM, Addison-Wesley, 1992.

[23] D. Heath, S. Kasif, and S. Salzberg. Induction of oblique decision trees. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1002–1007, 1993.

[24] F. Herrera, M. Lozano, and J. L. Verdegay. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review*, 12(4):265–319, 1998.

[25] J. H. Holland. *Adaption in Natural and Artificial Systems.* MIT Press, 1975.

[26] A. Ittner and M. Schlosser. Non-linear decision trees - NDT. In *International Conference on Machine Learning*, pages 252–257, 1996.

[27] V. S. Iyengar. Hot: Heuristics for oblique trees. In *11th International Conference on Tools with Artificial Intelligence*, pages 91–98, 1999.

[28] J. F. Javis, C. N. Judice, and W. H. Ninke. A survey of techniques for the image display of continuous tone images on a bilevel display. *Computer Graphics and Image Processing*, 5(1):13–40, Mar. 1976.

[29] K. A. D. Jong, W. M. Spaers, and D. F. Gordon. Using genetic algorithms for concept learning. *Machine Learning*, 13:161–188, 1993.

[30] J. R. Koza. *Genetic Programming: on the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[31] J. R. Koza. *Genetic Programming: Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, 1994.

[32] J. R. Koza, F. H. Bennett, D. Andre, and M. A. Keane. *Genetic Programming: Genetic Programming III: Darwinian Invention and Problem Solving.* Morgan Kaufmann, 1999.

[33] K. S. Leung, Y. Leung, L. So, and K. F. Yam. Rule learning in expert systems using genetic algorithm: 1, concepts. In *Proceedings of the 2nd International Conference on Fuzzy Logic and Neural Networks*, volume 1, pages 201–204, 1992.

[34] K. S. Leung, Y. Leung, L. So, and K. F. Yam. Rule learning in expert systems using genetic algorithm: 2, empirical studies. In *Proceedings of the 2nd International Conference on Fuzzy Logic and Neural Networks*, volume 1, pages 205–208, 1992.

[35] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* New York: Springer-Verlag, 3rd edition, 1996.

[36] T. M. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[37] H. Muhlenbein and D. Schlierkamp-Voosen. Predictive models for the breeder genetic algorithm i: Continuous parameter optimization, 1993.

[38] S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–32, 1994.

[39] S. K. Murthy, S. Kasif, S. Salzberg, and R. Beigel. Oc1: A randomized induction of oblique decision trees. In *National Conference on Artificial Intelligence*, pages 322–327, 1993.

[40] S. C. Ng and K. S. Leung. Induction of quadratic decision trees using genetic algorithms. In *Proceedings of 2003 Intelligent Automation Conference*, pages 979–984, 2003.

[41] S. C. Ng and K. S. Leung. Induction of quadratic decision trees using genetic algorithms and k-d trees. In *Proceedings of the Third WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Databases*, 2004.

[42] A. Papagelis and D. Kalles. Breeding decision trees using evolutionary techniques. In *Proceedings of the 18th International Conference on Machine Learning*, pages 393–400. Morgan Kaufmann, 2001.

[43] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

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

[45] I. Rechenberg. *Evolutionsstrategie: Optimienrung Technischer Systeme nach Prinzipien der Biologiscen Evolution.* Frommann-Holzboogt Verlag, 1973.

[46] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[47] S. R. Safavian and D. Landgrebe. A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man and Cybernetics*, 21(3):660–674, 1991.

[48] H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS.* Addison-Wesley, 1990.

[49] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, 1990.

[50] H. P. Schewefel. *Numerical Optimization of Computer Models.* Wiley, 1981.

[51] S. Shah and P. S. Sastry. New algorithms for learining and pruning oblique decision trees. *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, 29(4):494–505, Nov. 1999.

[52] M. Shneier. Two hierarchical linear feature representation: Edge pyramids and edge quadtrees. *Computer Graphics and Image Processing*, 17(3):211–224, Nov. 1981.

[53] G. Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann Publishers, 1989.

[54] K. P. Unnikrishnan and K. P. Venugopal. Alopex: A correlation-based learning algorithm for feedforward and recurrent neural networks. *Neural Computation*, 6(3):469–490, 1994.

[55] M. L. Wong and K. S. Leung. Inducing logic programs with genetic algorithms: The genetic logic programming system. *IEEE Expert*, 10(5):68–76, 1995.

[56] M. L. Wong and K. S. Leung. *Data Mining using Grammar Based Genetic Programming and Applications.* Kluwer Academic Publishers, 2000.

[57] A. H. Wright. Genetic algorithms for real parameter optimization. In G. J. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 205–218. Morgan Kaufmann, San Mateo, CA, 1991.

[58] O. T. Yıldız and E. Alpaydın. Linear discriminant trees. In *Proc. 17th Int. Conf. Machine Learning*, pages 1175–1182, 2000.