Fast Training of SVM with β-neighbor Editing

Wan Zhang

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Department of Computer Science & Engineering

Fast Training of SVM with β-neighbor Editing

submitted by

Wan Zhang

for the degree of Master of Philosophy

at the Chinese University of Hong Kong

Abstract

Recently, the Support Vector Machine (SVM) has shown its powerful abilities to perform classification, estimation, and regression. The major task in the SVM training procedure is to locate the points, or rather support vectors, based on which we construct the discriminant boundary in classification task. But one of its potential drawbacks is its slow training procedure. During the process of locating the support vectors, reducing the size of the training set for SVM will speed up the training procedure. In this thesis, we approach to the solution by reducing the size of the reference set from the computational geometry perspective. In particular, we propose the β-neighbor edited algorithm which locates a set of "edited" set that can be used to locate support vectors in order to achieve that target. To further speed up the β-neighbor edited algorithm, we propose two approaches: one through parallelism reduction and the other through a structural reduction using spatial indexing techniques. Furthermore the combination of the two approaches is applied in the implementation of the β-neighbor edited algorithm to accelerate the editing step. We compare the classification methods, including SVM method, k-Nearest Neighbor method and $C4.5$, in which the "edited" set is used as the training dataset, against with the classification methods with the original

training dataset. Reduced SVM, an improved algorithm of SVM to reduce the training time, is also compared to our approach in this thesis. From our experiments, we show that with the "edited" set as the training set the classification methods can preserve the high prediction accuracy. With the aid of parallelism and spatial indexing structure, our method in which we first edit the training set and then do SVM training on the reduced training set (edited set), reduces the real training time for the large dataset.

Acknowledgment

First of all I would like to express my gratitude to my supervisor, Prof. Irwin King. During the years of my MPhil. studies, he helped me focus on the more interesting topics, and he was always ready to listen to my questions and ideas, no matter how good(or bad) the ideas were. Without his encouragement and guidance, I can not favorably complete the study in CUHK. And without the insightful advice from him, my research have not been finished reasonably.

My gratitude also goes to Prof. Ada Fu for her great help and support through my study. Furthermore, I want to thank Prof. Michael Lyu who gave me valuable suggestions on term paper.

I also owe much to Yin-Ling Cheung for her fruitful discussion and advice. We work out some experiments together and talk about the details about the proposed algorithms and ideas. Thanks to Haiqin and Kaizhu for their help with the discussions on my idea.

Finally, I want to thank my parents, my sister and my niece for their love. All the things that I have done are for them.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction to Classification

Classification plays a key role in many areas of nature science, finance and industry, such as health-care outcome, marketing and automatic article classification, etc. Here is an example of classification. In a hospital, doctors want to predict whether a patient, hospitalized due to a heart attack, will have a second heart attack. The prediction is made based on demographic, diet and clinical measurements for that patient. In this scenario, we have two outcome categories (heart attack/no heart attack), which we wish to predict based on a set of $features$ (diet and clinical measurements). We have many existing medical records (a set of $data$), in which we observe the outcome and feature measurements for a set of objects (patients). Using the data, we build a prediction model, or $classifier$, which enables us to predict the outcome for new unseen objects (patients to be diagnosed).

Classification is the process of finding a set of models (or functions) that describe and distinguish data categories, for the purpose of being able to use the model to predict the class of objects whose categorical value is unknown [43].

We typically denote X, a d-dimensional vector, as the combination of features, in which each component is transferred to be real-valued. The categorical variable indicating outcome is denoted by Y. Observed values are written

in lowercase, i.e., the ith observed value of X is denoted by x_i. The model yields predictions, $f(x;\theta)$, where θ represents the parameters of the model structure.

Classification involves two major steps (See Figure 1.1).

1. Build a classification model using training data. Every object in the dataset must be preclassified, i.e. its class label must be known in advance. Many classification algorithms, which will be discussed later, can be used to build the model. The model is represented in the form of classification rules or mathematical formulae [43].

2. Assign a class label to each object in a test dataset by the model constructed in the preceding step. Each sample of test data is also preprocessed and preclassified in advance. The accuracy of the classification model is determined by comparing true class labels of the testing data with those assigned by the model.

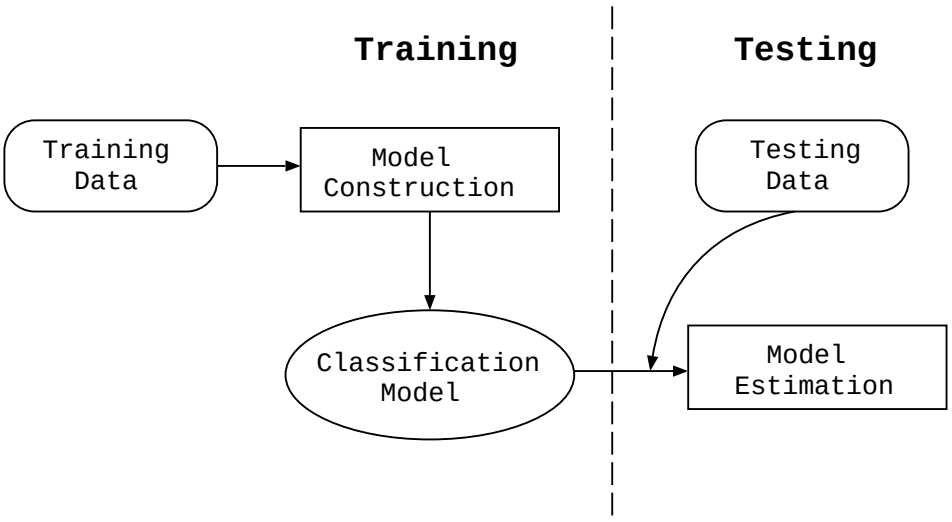

Figure 1.1: Classification Process from Training to Testing

Consider the nature of the mapping function f in the classification model

for a simple problem in two-dimensional X space. The mapping in effect produces a piecewise constant surface over the plane; that is only in certain regions does the surface belongs to one predefined class. The union of all such regions is known as the *decision region* for that class.

Knowing where these decision regions are located in the X plane is equivalent to knowing where the *decision boundaries* between the regions lie. Thus we can think of the problem of learning a classification function f as being equivalent to learning decision boundaries between classes.

We can make simple parametric assumptions about the functional form of the boundaries. For example, a classic approach is to use a linear hyperplane in the d-dimensional X space to define a decision boundary between two classes. That is, the model partitions the X-space into disjoint decision regions (one for each class), where the decision regions are separated by linear boundaries. A more complex model means to allow higher-order terms, which yield polynomial decision boundaries. The neural network classifiers can be used to build the non-linear boundaries directly [84]. Another way to allow flexible forms for non-linear boundary is to combine multiple simple local models, such as decision tree classifiers [81]. The Nearest-Neighbor classifier is to classify a new unclassified data point according to the label of its nearest neighbor in the training samples. Although this technique is generally considered as a method rather than a model, it does in fact implicitly define a piecewise linear decision boundary (when using Euclidean distance to define neighbors) [44].

There are also many other classification techniques, providing different ways to model decision boundaries, such as Bayesian method [62] [34], and support vector machine [94] [53], etc.

A good classifier is one that accurately predicts the class label for new unseen objects. A performance of a classifier not only depends on its mechanics, but also on the characteristics of the data. Even on the same pattern, different classifiers may behave quite diversely. The user has to choose the

most appropriate method for the problems in various situations. A large research effort has been focused on comparative studies of various classifiers in the context of different applications. One of the most extensive efforts was the Statlog project [69], in which a wide range of classifiers was tested using a large number of different data sets. The project tried to relate performance of algorithms to characteristics of datasets and verified that the performance of the various algorithms is much influenced by the particular application.

1.2 Problem Definition

Support vector machine has come to play a very dominant role in data classification using a linear or nonlinear classifier [13] [25]. There are three problems that impelled the initial development of SVM. They are the bias variance tradeoff [39], capacity control [42], and overfitting [70]. In fact, the three problems share the same root. Roughly speaking, for a given learning task, with a given finite amount of training data, the best generalization will be achieved if the right balance is struck between the accuracy attained on that particular training set, and the "capacity" of the machine (the ability of the machine to learn any training set without any error). The exploration and formalization of these concepts forms the basis for the Statistical Learning theory [93] [20]. The following section is an introduction to support vector machine.

We are given the training data, $\{x_i, y_i\}$, $i = 1, \cdots, m$, in which each example has d inputs ($x_i \in \mathbf{R}^d$), and one of two values ($y_i \in \{-1, 1\}$) as the class label. In SVM, all the hyperplanes in R^d are parameterized by a vector ω and a constant b, expressed in the equation

$$\omega \cdot x + b = 0, \tag{1.1}$$

where ω is the vector orthogonal to the hyperplane. We define the margin as the sum of the distances from the separating hyperplane to the closest points

in two classes [20]. In fact, the basic concept behind SVM is to find the tradeoff between the largest margin (distance) and training errors, so the generalized optimal separating hyperplane is regarded as the solution to Eq. (1.2) as follows [94],

$$\min(\frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{m}\xi_i) \tag{1.2}$$
$$\text{subject to } y_i(x\cdot\omega + b) \geq 1 - \xi_i, \xi_i \geq 0.$$

For solving high dimension problems, SVM maps x into higher space through a mapping function ϕ in such a way that $\phi(x)\cdot\phi(y) = K(x,y)$ for some known and easy-to-evaluate set of functions, K [94]. These functions, K, are called kernel functions.

Although using SVM can achieve high performance and optimal solution in the real-life classification problems, there are still some deficiencies in the SVM method.

1. **Iterative Procedure For Solving Problems**

   SVMs are constructed on the basis of quadratic optimization techniques for the objective function. There are a number of methods for solving quadratic optimization problems. For example, methods can be constructed based on the conjugate gradient procedure [55], the interior point method [92], the projection procedure [95], the decomposition method [53] [51] [76]. Any of these can be used for constructing an SV machine. But in all procedures, an iterative procedure for solving the quadratic problem is required. This makes the speed of the convergence difficult to control.

2. **Model Selection and Parameter Setting**

   The difficulty for SVM used in practice is the selection of the kernel function type and the parameters of the objective function. For example, the

popular kernel functions are RBF (Radial Basis Function) [78], polynomial function [94], two-layer sigmoidal neural network [94]. In general, each function has its own parameters, whose values are in a wide range. Users like to tune the parameters which can lead to the best general performance. This process is time consuming.

All existing works on improving the performance of SVM in practice is to refine the algorithm of SVM itself, such as revising the method for solving the quadratic optimization problem [49], auto model selection [59], randomly selected portion of the dataset in the procedure of solving quadratic optimization: RSVM [60]. In this thesis we reduce the training point set by performing preprocessing on the dataset. Recall that only the training examples that become support-vectors are actually needed in determining the optimal hyperplane. If there are some ways to throw out the data points, which are not support vectors, we can reduce the problems size. Yang [101] provides a method to determine the data points (potential support vectors) by solving a series of Linear Programming problems, which are more efficient than conventional methods using QPs.

Are there any other methods to reduce the size of training data set? Are those reduced set suitable for SVM training? Can we do some improvements on those methods?

All these issues motivated and directed our research.

1.3 Major Contributions

In this paper, the contributions are:

1. Show a concrete connection between SVM and some concepts in Computational geometry [104].

2. Based on the connection, a preprocessing step using β-neighbor edited

Table 1.1: Four Trials for the Testing Idea

	Non-parallel	**Parallel**
Non-index	original	β-neighbor$_{para}$
SR-tree Index	SRtree-β-neighbor	SRtree-β-$_{para}$

algorithm is proposed to reduce the training dataset for SVM from the geometric viewpoint [103]. As a result, the computation of SVM is lowered while the convergence is speeded up, with the high performance preserved.

3. Make improvements on the geometric method with some effective techniques, such as parallelism and SR-tree indexing [22]. Four kinds of experiments are conducted to confirm our theoretical claims [102] (See Table 1.1).

1.4 Thesis Organization

In the next chapter, we will present the main classification methods in statistical field, neural network and decision tree. Chapter 3 presents the related topic with SVM from the viewpoint of Computational Geometry. Ideas of data editing and the effective techniques to speed up the editing procedure will be introduced in Chapter 4. In Chapter 5, we conduct a series of experiments on different data sets and compare the performance of the different classifiers. Lastly, we conclude and make some final remarks in Chapter 6.

Chapter 2

Literature Review

A wide variety of approaches has been developed towards classification task. In this chapter, three main historical strands of research are illustrated: *statistical method*, *decision tree*, and *neural network* (See Fig. 2.1). Those have largely involved different professional and academic groups, and emphasized different issues. However, all groups have had some objectives in common [69]. They have all attempted to derive procedures that would be able:

- to equal, if not exceed, a human decision-maker's behavior, but have the advantage of consistency and, to variable extent, explicitness;
- to handle various problems and, given enough data, to be extremely general;
- to be used in practical cases with proven success.

2.1 Fisher's Linear Discriminant

This is one of the oldest classification methods, and is the most commonly implemented in computer packages. It is a classification method that projects high-dimensional data onto a line and performs classification in this one-dimensional space. The projection maximizes the distance between the means

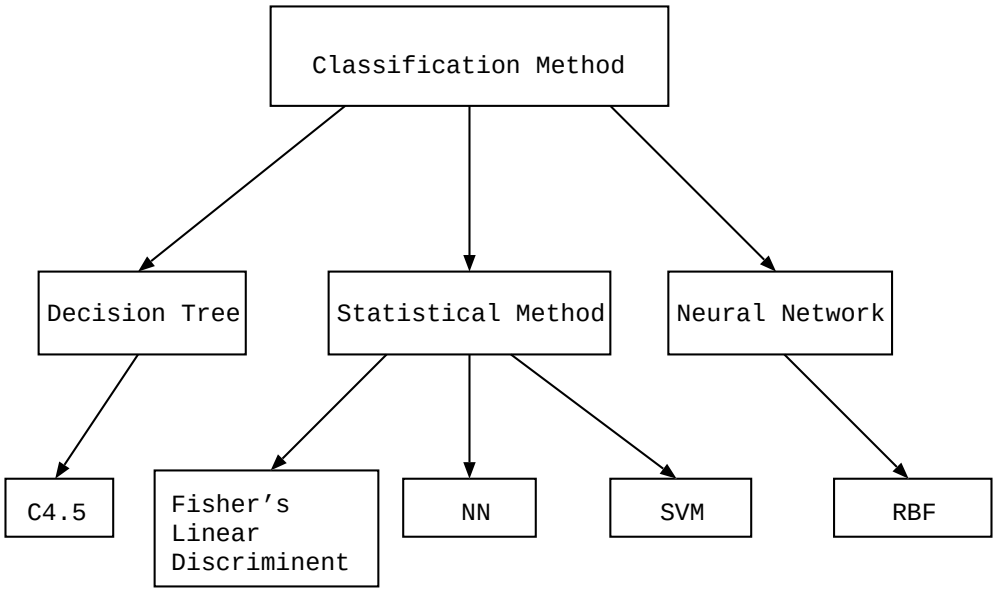

Figure 2.1: Review of Classification Methods

of the two classes while minimizing the variance within each class. This defines the Fisher criterion [31], which is maximized over all linear projections, p,:

$$J(p) = \frac{|m_1 - m_2|^2}{s_1^2 + s_2^2}, \tag{2.1}$$

where m represents a mean, s^2 represents a variance, and the subscripts denote the two classes.

2.2 Radial Basis Function Networks

RBF networks emerged as a variant of artificial neural network in late 80's [19]. RBFs are embedded in a two layer neural network, where each hidden unit implements a radial activated function. The output units implement a weighted sum of hidden unit outputs. The input into an RBF network is nonlinear while the output is linear. Their excellent approximation capabilities have been studied in [74] [77]. Due to their nonlinear approximation properties,

RBF networks are able to model complex mappings, which perceptron neural networks can only model by means of multiple intermediary layers [46].

In order to use a RBF network we need to specify the hidden unit activation function, the number of processing units, a criterion for modelling a given task and a training algorithm for finding the parameters of the network. Finding the RBF weights is called network training. After training, the RBF network can be used with data whose underlying statistics is similar to the training set. RBF networks have been successfully applied to a large diversity of applications including speech recognition, chaotic time-series modelling, etc.

2.3 Decision Tree

In this section we briefly review a large class of nonlinear classifiers known as decision trees. They are multistage decision systems in which classes are sequentially rejected until we reach a finally accepted class. To this end, the feature space is split into unique regions, corresponding to the classes, in a sequential manner. Upon the arrival of a feature vector, the searching of the region to which the feature vector will be assigned is achieved via a sequence of decisions along a path of *nodes* of an *tree*, appropriately constructed. Such schemes perform well especially when a large number of classes are involved. The most popular decision trees are those that split the space into hyperrectangles with sides parallel to the axis. The sequence of decisions is applied to individual features, and the questions to be answered are of the form "is feature $x_i \leq \alpha$?" where α is a threshold value. Such trees are known as ordinary binary classification trees (OBCT) [87]. We can also build other types of trees, which split the space into convex polyhedral cells or into pieces of spheres.

The basic idea of an OBCT can be demonstrated via the simple example (See Fig. 2.2). By a successive sequential splitting of the space we have created regions corresponding to the various classes. Fig. 2.3 shows the respective

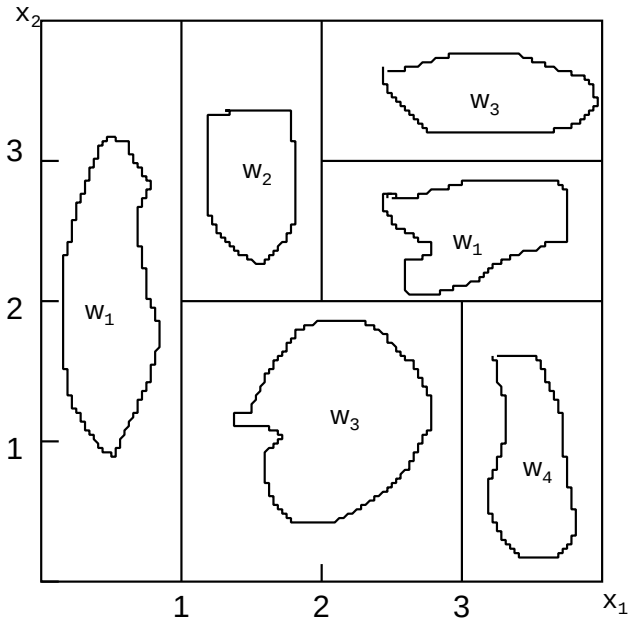

Figure 2.2: Decision Tree Partition

binary tree with its decision nodes and leaves.

In general, in order to construct a decision tree, a splitting criterion (i.e., optimizing function) may be adopted for each node. For further information and a deeper study of this class of classifiers the interested reader may consult the seminal book [18]. A nonexhaustive sample of later contributions in the area is given by [24] [85] [81] [30].

Finally, it should be stated that there are close similarities between the decision trees and the neural network classifiers. Both of them aim at forming complex decision boundaries in the feature space. A major difference lies in the way decisions are made. Decision trees employ a hierarchically structured decision function in a sequential fashion. In contrast, neural networks utilize a set of soft (not final) decisions in a parallel fashion [87].

Furthermore, their training procedures are activated by different philosophies. However, despite their differences, it has been shown that linear tree

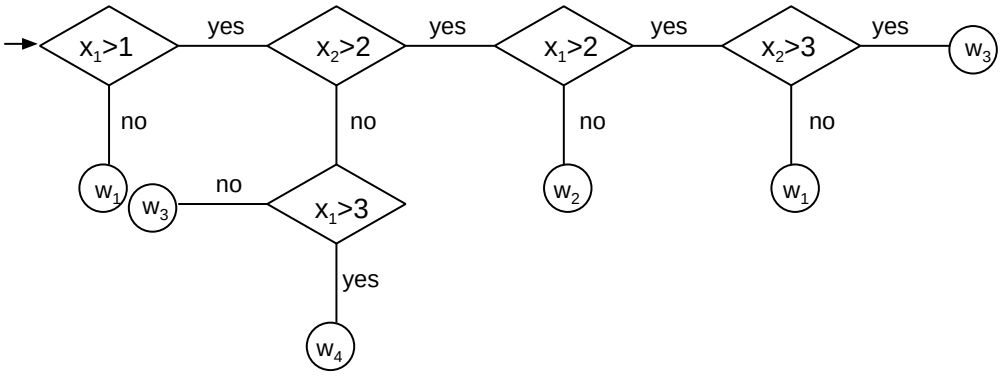

Figure 2.3: Decision Tree Classification

classifiers (with a linear splitting criterion) can be adequately mapped to a multi layer perceptron structure [86] [85] [75].

2.4 Nearest Neighbor

kNN method was first given in an unpublished report by Fix and Hodges [32]. There is a very extensive literature on Nearest Neighbor classifiers, much of which is reviewed or reprinted in [26].

Close is measured by Euclidean distance function here. The basic steps of nearest neighbor method to do classification are described as follows. When NN method wants to classify a new point, x, it simply finds out the k data points in the training dataset closest to x. Then, they assign the new point, x, to the class that has the majority of points among these k points.

In theoretical terms, we are taking a small volume of the space of variables, centered at x, and with radius the distance to the kth nearest neighbor. Then the maximum likelihood estimators of the probability that a point in this small volume belongs to each class are given by the proportion of training points in this volume that belong to each class. The k-nearest neighbor method assigns

a new point to the class that has the largest estimated probability [44].

This simple outline leaves a lot unsaid. In particular, we must choose a value for k and a metric through which to define *close*. The most basic form takes $k = 1$, but this makes a rather unstable classifier (high variance, sensitive to the data), and the predictions can often be made more consistent by increasing k (reduces the variance, but may increase the bias of the method since there is more averaging). However, increasing k means that the training data points now being included are not necessarily very close to the object to be classified. This means that the "small volume" may not be small at all. Since the estimates are estimations of the average probability of the value at any particular point within the volume and this deviation is likely to be larger as the volume is larger. The dimensionality d plays an important role here: for a fixed number of data points m, we increase d (adding attributes) to the data points and then the data points become more and more sparse. This means that the predicted probability may be biased from the true probability at the point in question [44].

2.5 Support Vector Machine

Support Vector Machine [94] has been extensively used in machine learning and data mining [66] [16] [15]. It was introduced by Vladimir Vapnik and colleagues. The earliest mention was in [93], but the first main paper seems to be in [94]. In the classification problem, we attempt to classify points coming from different classes by a linear or nonlinear separating surface. The learning process of classification utilizes the data point in the training set to generate a separating surface which assigns each training input data point to the appropriate category. The separating surface is then tested on the unseen data. Now let us consider how SVM performs classification in the following cases.

2.5.1 Linear Separable Case

Given a set of input points, $\{x_i, y_i\}, (i = 1, \cdots, m,\ y_i \in \{-1, 1\},\ x_i \in \mathbf{R}^d)$, suppose there exists a hyperplane which could separate the positive from the negative examples. It means that the points x which lie on the hyperplane satisfy $\omega \cdot x + b = 0$, where ω is normal to the hyperplane, $|b|/\|\omega\|$ is the perpendicular distance form the hyperplane to the origin, and $\|\omega\|$ is the Euclidean norm of ω. The margin is defined as the sum of the distance of the separating hyperplane to the closest positive and negative points [20]. For the linearly separable case, SVM simply finds the separating hyperplane with the largest margin. It could be formulated as follows:

$$\omega \cdot x_i + b \geq +1, \quad for \quad y_i = +1 \tag{2.2}$$

$$\omega \cdot x_i + b \leq -1, \quad for \quad y_i = -1. \tag{2.3}$$

These can be combined into a set of linear constraints for all the training data points:

$$y_i(\omega \cdot x_i + b) - 1 \geq 0, \quad \forall i. \tag{2.4}$$

From Fig. 2.4 and Eq. (2.4), we can calculate the margin as $2/\|\omega\|$. Then we can find the separating hyperplane with the largest margin under the constraint in Eq. (2.4) by

$$\min \frac{1}{2}\|\omega\|^2 \tag{2.5}$$

$$\text{subject to } y_i(x \cdot \omega + b) \geq 1, i = 1, \cdots, m.$$

In short, the training of this classifier is achieved by solving a linearly constrained optimization problem. Let $\alpha_i, i = 1, \cdots, m$ be the m nonnegative Lagrange multipliers, one for each inequality constraints in Eq. (2.4), the solution to Eq. (2.5) equals to the solution to the constrained quadratic optimization problem using the Wolfe dual theory [20] as,

$$L_P \equiv \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j x_i x_j - \sum_i \alpha_i \tag{2.6}$$

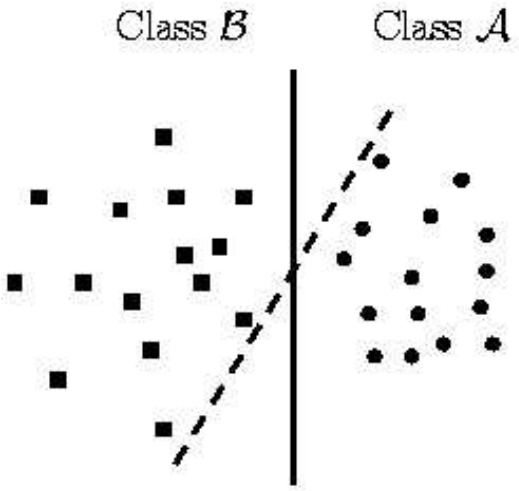

Figure 2.4: Linear Separating hyperplanes for the Separable Case

$$\text{subject to } 0 \leq \alpha_i \leq 1, \sum_i \alpha_i y_i = 0.$$

This is a quadratic programming (QP) problem, and all constraints are linear. Then we can apply various techniques to solve the QP problem and obtain the optimal solution for SVM.

2.5.2 Non Linear-separable Case

The above algorithm for separable data, when applied to non linear-separable data, will give no feasible solution. So we need extend these ideas to handle non linear-separable data by introducing some non-negative slack variables $\xi_i, i = 1, \cdots, m$ and allowing some points to be misclassified (Fig. 2.5). Eq 2.5 becomes:

$$\begin{aligned} y_i(x \cdot \omega + b) &\geq 1 - \xi_i, \\ \xi_i &\geq 0 \quad \forall i. \end{aligned}$$

Clearly, when an error occurs, ξ_i must exceed zero. So $\sum_i \xi_i$ is an upper bound of the number of training errors. Therefore considering the extra cost for errors, the objective function will be changed from $\frac{1}{2}\|\omega\|^2$ to $\frac{1}{2}\|\omega\|^2 + C(\sum_i \xi_i)$, where C is a parameter chosen by users to decide the penalty rule.

In the soft margin (non linear-separable case) formulation of [94], the generalized optimal separating hyperplane is the solution to Eq. (2.7) as follows,

$$\begin{aligned} \min_{\omega,b}(\frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{m}\xi_i) & \qquad (2.7) \\ y_i(x_i \cdot \omega + b) &\geq 1 - \xi_i, \\ \xi_i &\geq 0, \\ i &= 1, \cdots, m, \\ C &> 0, \end{aligned}$$

where C is the penalty parameter. When data are not linear separable, there is a penalty term $C\sum_{i=1}^{m}\xi_i$ which can reduce the number of the training errors. The basic concept behind SVM is to find a balance between the regularization term (maximum margin) $\frac{1}{2}\|\omega\|^2$ and the training error. Furthermore, the α value is bounded by C, which will limit the search space for the QP problem above, i.e. the possible range for the α_i value.

It can be proven that, for any misclassified training data, x_i, the corresponding α_i must lie at the upper bound. This can be understood by imaging that a particular data point tries to assert a stronger influence on the boundary so that it can be classified correctly, by increasing the corresponding α_i value. When the α_i value reaches its maximum bound, it cannot increase its influence further, hence this point will stay misclassified. This analogy is consistent with the fact that C, the upper bound for α_i, is the trade-off between maximum margin and classification error. A larger C corresponds to assigning a higher penalty to errors, and consequently α is allowed to have a larger value; in this manner, each misclassified data can assert a stronger influence on the boundary.

Similarly, the corresponding Lagrangian formulation is defined as

$$\begin{aligned} L_P &= \frac{1}{2}\|\omega\|^2 + C\sum_i \xi_i \qquad (2.8) \\ &\quad - \sum_i \alpha_i\{y_i(x_i \cdot \omega) + b) - 1 + \xi_i\} - \sum_i \mu_i\xi_i. \end{aligned}$$

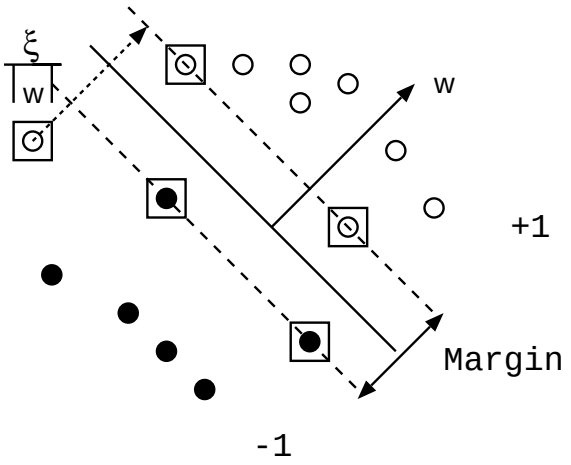

Figure 2.5: Handling Non Linear-separable Case with Slack Variables

The optimal solution to this problem satisfies Karush-Kuhn-Tucker (KKT) conditions [94]. Then all the following conditions are satisfied:

$$\begin{gathered}\omega = \sum_i \alpha_i y_i x_i, \sum_i \alpha_i y_i = 0, C - \alpha_i = \mu_i, \\ y_i(x_i \cdot \omega + b) - 1 + \xi_i \geq 0, 0 \leq \alpha_i \leq C, \\ \alpha_i\{y_i(\omega \cdot x_i + b) - 1 + \xi_i\} = 0, \mu_i \xi_i = 0, \\ \mu_i, \xi_i \geq 0, i = 1, \cdots, m, C > 0.\end{gathered} \tag{2.9}$$

Some of the conditions above can be substituted into Eq. (2.8) to give the new Lagrangian in Eq. (2.10).

$$\begin{gathered}\min_{b,\alpha} \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i x_j - \sum_i \alpha_i \\ \text{subject to } 0 \leq \alpha_i \leq C, \sum_i \alpha_i y_i = 0.\end{gathered} \tag{2.10}$$

Solving Eq. (2.10) will give the value for all α_i. b is found by using Eq. (2.9), by selecting any training data with nonzero α_i value when those free variables exist.

2.5.3 Nonlinear Case

So far the SVM classifier can only have a linear hyperplane as its decision surface. This formulation can be further extended to build a nonlinear decision SVM. The motivation for this extension is that an SVM with nonlinear decision surface can classify nonlinearly separable data (See Fig. 2.6).

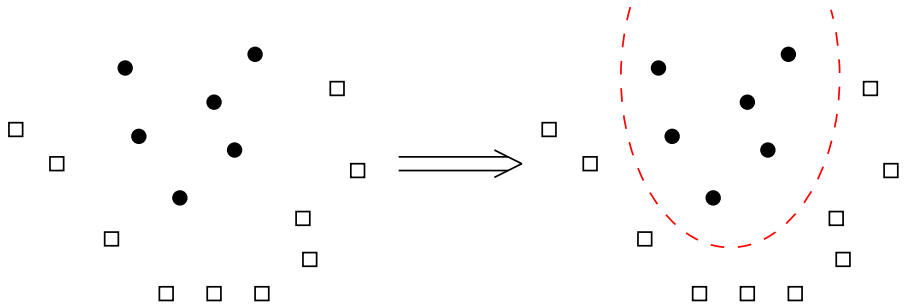

Figure 2.6: Nonlinear Case for SVM

For solving nonlinear separable problems, SVM maps the training data from R^d to a Hilbert space $\mathcal{H}$ of a higher dimension (maybe infinite), and fits an optimal linear classifier in $\mathcal{H}$. Then the linear SVM formulation above can be applied to these data.

In the SVM formulation, the training data only appear in the form of dot products, $x_i \cdot x_j$ and the same is true in the decision function $\omega \cdot x + b$ ($\omega = \sum_i \alpha_i y_i x_i$). These can be replaced by dot products in Euclidean space $\mathcal{H}$, i.e. $\phi(x_i) \cdot \phi(x_j)$, where ϕ is a mapping function: $\mathbf{R}^n \rightarrow \mathcal{H}$

$$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j) \tag{2.11}$$

The dot product in the high dimension space can also be replaced by a kernel function, 2.11. By computing the dot product directly using a kernel function, one avoid the mapping $\phi(x)$. This is desirable because $\mathcal{H}$ has possibly infinite dimensions and $\phi(x)$ can be tricky or impossible to compute. Using

Table 2.1: Some Possible Kernel Functions and the Types of Decision Surface They Define

Kernel Function	Type of Classifier
K(x,y)=exp$(-\frac{\|x-y\|^2}{2\sigma^2})$	Gaussian RBF
K(x,y)=$(x \cdot y + 1)^d$	Polynomial of degree d

a kernel function, an SVM that operates in infinite dimensional space can be constructed. Furthermore, set $Q_{ij} = y_i y_j K(x_i, x_j)$, the objective function is changed as follows

$$R(\alpha) = \frac{1}{2}\alpha \cdot Q \cdot \alpha - \sum \alpha_i. \tag{2.12}$$

For any kernel function suitable for SVM, there must exist at least one pair of $\{\mathcal{H}, \phi\}$, such that Eq. (2.11) is true, i.e. the kernel function represents the dot product of the data in $\mathcal{H}$. The kernel that has these properties satisfies the Mercer's theorem [94]. By using kernel functions in the dual SVM problem, SVM can efficiently and effectively emulate many types of well known classifiers which are introduced in [73], as shown in Table 2.1.

The primary appeal of SVM is that it can be simply and elegantly applied to nonlinear discrimination. With only minor changes, SVM methods can construct a wide class of two-class nonlinear discriminants by solving a single QP problem. The basic idea is that data points in the training dataset are mapped to a higher dimensional space so that the dual SVM problem is used to construct a linear discriminant in the higher dimensional space that is nonlinear in the original attribute space.

2.5.4 Multi-class SVM

The solution of binary classification problems using support vector machine (SVM) is well developed, but multi-class problems with more than two classes have typically been solved using voting scheme methods based on combing

many binary classification decision functions [97] [17]. Currently there are two approaches to solve multi-class problem. One approach is by constructing and combining several binary classifiers, while the other is a formulation of SVM which solves the multi-class problem in a single optimization [48].

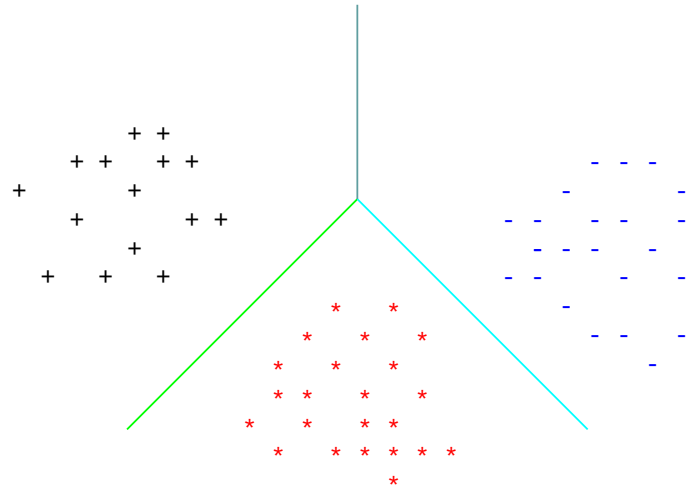

Figure 2.7: Multi-class Case with One-against-one Method

Here, we consider the multi-class SVM problem as the combination of many binary classification problems. Based on binary classifications, the "One-against-one" method is introduced in [58]. For this method, $(\frac{k(k-1)}{2})$ classifiers can be constructed, separating each class from each other, where k is the number of categories. Each classifier is trained on data from the two classes. For example, we solve the following binary classification problem with the data points in the training set from class i and class j:

$$min_{\omega^{ij},b^{ij},\xi^{ij}} \quad \frac{1}{2}(\omega^{ij}) \cdot \omega^{ij} + C \sum_t \xi_t^{ij} \tag{2.13}$$

subject to

$$\begin{aligned} (\omega^{ij}) \cdot \phi(x_t) + b^{ij} &\geq 1 - \xi_j^{ij}, \quad y_t = i; \\ (\omega^{ij}) \cdot \phi(x_t) + b^{ij} &\leq -1 + \xi_j^{ij}, \quad y_t = j; \end{aligned}$$

$$\xi_t^{ij} \geq 0.$$

There are many different methods to do the future testing after all $k(k-1)/2$ classifiers are constructed. A majority vote across the classifiers or some other measure can be applied to classify a new point. For example, the following voting strategy suggested in [33]: if sign $((\omega^{ij}) \cdot \phi(x_t) + b^{ij})$ says x is in the ith class, then the vote for class x is added by one. Otherwise, the vote for class j is increased by one. Then we predict x to be in the class with the largest vote. The voting strategy is also called "Max Wins". If two classes have the same votes, we simply select the one with smaller index.

In practice, we solve the dual problem of Eq. (2.13) whose number of variables is the same as the number of data points in two classes. Hence if in average each class has m/k data points, we have to solve $k(k-1)/2$ quadratic programming problems where each of them has about $2m/k$ variables [48] (See Fig 2.7).

2.5.5 RSVM

The motivation for RSVM (Reduced Support Vector Machine) comes from the practical objective of generating a nonlinear separating surface for a large dataset which requires a small portion of the dataset for its characterization. When confronting the large data classification by a nonlinear kernel, the major problem is the size of the mathematical programming problem that needs to be solved and the time it takes to solve [60]. The key point of RSVM is to reduce the matrix Q in Eq. (2.12) from $m \times m$ to $m \times n$, where n is the size of randomly selected subset of the training data considered as candidates of support vectors. RSVM is different from directly solving smaller SVM problems with a subset of training data, because the m constraints in the primal problem 2.7 are still kept during the optimization process [64].

We describe the main procedure of modifying standard SVM to RSVM in

the following. First, the 1-norm of the slack variable ξ minimized with weight C is replaced by the square of 2-norm of the slack variable ξ. In addition, the distance between the two bounding planes is measured in the $(d+1)$-dimensional space of $(\omega, b) \in R^{d+1}$, which is similar to that of [14] [36] [35] [61]. Measuring the margin in this $(d+1)$-dimensional space instead of R^d has little or no effect on the problem as shown in [67]. The experimental results in [50] also verify that. Using twice the reciprocal squared of the margin instead, the modified SVM is as follows:

$$\min_{\omega,b,\xi} \quad \frac{1}{2}(\omega \cdot \omega + b^2) + C \sum_{i=1}^{m} \xi_i^2 \tag{2.14}$$

$$\text{subject to} \quad y_i(\omega \cdot \phi(x_i) + b) \geq 1 - \xi_i.$$

Its dual form becomes a simpler bound-constrained problem:

$$\min_{\alpha} \quad \frac{1}{2}\alpha \cdot (Q + \frac{I}{2C} + yy^T) \cdot \alpha - \sum_{i=1}^{m} \alpha_i \tag{2.15}$$

$$\text{subject to} \quad \alpha_i \geq 0.$$

By substituting ω with $\sum_{i=1}^{m} y_i \alpha_i \phi(x_i)$, which is obtained at the optimal solution, the optimization problem turns into [64]:

$$\min_{\alpha,b,\xi} \quad \frac{1}{2}(\alpha \cdot Q\alpha + b^2) + C \sum_{i=1}^{m} \xi_i^2 \tag{2.16}$$

$$\text{subject to} \quad Q\alpha + by \geq e - \xi.$$

Though (2.16) is different from (2.15), the dual problem, we can show that for any optimal α of (2.16), the corresponding ω defined by $\sum_{i=1}^{m} y_i \alpha_i \phi(x_i)$ is also an optimal solution of (2.14), so we can solve (2.16) instead of (2.15). (See details in [64].)

RSVM achieves reducing the number of support vectors by randomly select a subset of n samples to construct ω:

$$\omega = \sum_{i \in R} y_i \alpha_i \phi(x_i), \tag{2.17}$$

where R contains indices of the subset. Then the new problem is obtained by substituting 2.17 in 2.16:

$$\begin{aligned} \min_{\bar{\alpha},b,\xi} \quad & \frac{1}{2}(\bar{\alpha} \cdot Q_{RR}\bar{\alpha} + b^2) + C\sum_{i=1}^{m} \xi_i^2 \\ \text{subject to} \quad & Q_{:,R}\bar{\alpha} + by \geq e - \xi, \end{aligned} \tag{2.18}$$

where the size of R is n, $\bar{\alpha}$ is the collection of all α_i, $i \in R$, and $Q_{:,R}$ is the submatrix of columns corresponding R [64]. At last, with the idea of generalized SVM [65], Lee [60] simplified the term $\frac{1}{2}\bar{\alpha} \cdot Q_{RR}\bar{\alpha}$ to $\frac{1}{2}\bar{\alpha} \cdot \bar{\alpha}$ so RSVM solves

$$\begin{aligned} \min_{\bar{\alpha},b,\xi} \quad & \frac{1}{2}(\bar{\alpha} \cdot \bar{\alpha} + b^2) + C\sum_{i=1}^{m} \xi_i^2 \\ \text{subject to} \quad & Q_{:,R}\bar{\alpha} + by \geq e - \xi. \end{aligned} \tag{2.19}$$

The main purpose of this thesis is to speed up the training part for SVM. RSVM is one of the methods, which aim to achieve that target. It refines the SVM algorithm from the inner structure, while in our work the emphasis is on the preprocessing step of the dataset, in which the size of the data set is reduced. The connection between our method and SVM will be shown in the next chapter. Furthermore, comparisons between our method and RSVM will be presented in Chapter 4.

2.6 Summary

In previous sections, we simply describe the basic ideas of the different classification methods. Table 2.2 shows the comparisons of the classifiers and gives more explicit explanations of the uses of the various classifiers according to the selection criteria which include speed, ability to deal with complex data and implementation. Generally the neural network classifier would be used only if the other classification methods failed. So we will only conduct the experiments on the last three classifiers to show the further comparisons of their performances, predictive accuracy, time to construct the model and time to use the model. All the experimental results will be discussed in Chapter 4.

Table 2.2: Comparisons of the Classifiers Referred to Above

Classifiers	Advantages	Disadvantages
Fisher's Linear Discriminant	Fast training Fast evaluation of the decision function	Simple decision boundary Only solve binary class problems
Neural Network	Learn complicated class boundaries Fast evaluation of the learned decision function	Slow training time Hard to interpret Hard to implement: trials of choosing number of nodes
Decision Tree	Reasonable training time Fast evaluation of the learned decision rules	Simple decision boundaries Can not handle complicated relationship between attributes
k-Nearest Neighbor	Fast training Learn complex decision boundaries	Slow testing procedure Notion to "close" vague
SVM	Learn complicated boundaries Fast evaluation of the decision function Good generalization	Hard to implement: trials of parameters Slow training

Chapter 3

Computational Geometry

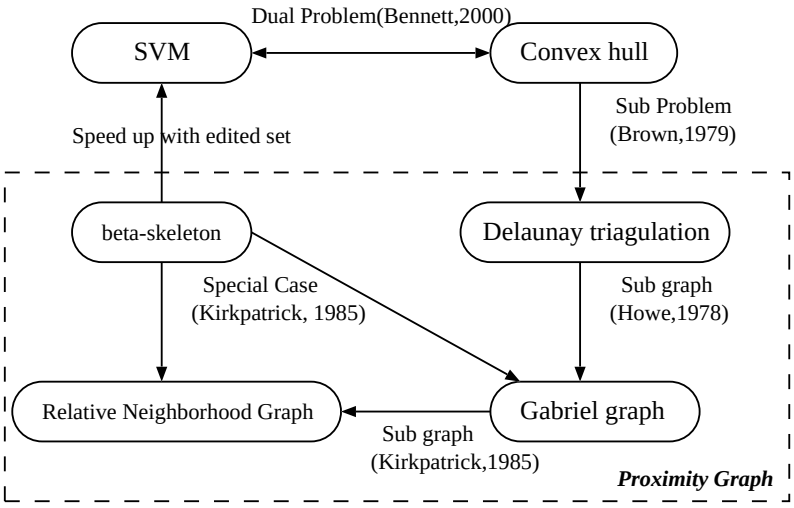

Figure 3.1: The Relationship among the Terms

SVM is a very powerful classification method. The intuition of finding the maximal margin separating hyperplane, which is the basic idea of SVM, is a geometric one. To solve it, Lagrangian multipliers are used to set up the QP problem, and then the following steps to solve that QP problem are pure mathematics. However, finding the optimal hyperplane can also be done in a geometric setting, too.

The solution to the Convex hull problem provides a way from a computational geometric viewpoint to locate support vectors, which are used to build the separating hyperplane [6]. So far in Computational geometry, the most significant building blocks that we have learned about are Convex hull, Voronoi diagram, and Delaunay triangulation. Edelsbrunner et al. [29] disclosed the relationships between of all of these concepts. There is an intimate connection between Delaunay triangulations in R^d and Convex hulls in $R^d + 1$. Furthermore, Delaunay triangulation (DT) contains, as subgraphs, various structures with diverse applications. These things inspire us to exploit a geometric method to reduce the size of the training set for SVM (See Fig. 3.1).

3.1 Convex hull

When we construct the decision boundary with convex hull, the separating plane is the hyperplane which is orthogonal to the line segment and bisects the line segment which connects the nearest points of the two convex hulls of the data sets.

3.1.1 Separable Case

In Fig. 3.2, the convex hull of class $A(B)$ consists of all the points which could be written as convex combination of the points in $A(B)$. A convex combination of points in A, u, is denoted by $u{=}\sum_{i\in A}\beta_i x_i$, where $\beta_i \geq 0$ and $\sum_{i\in A}\beta_i{=}1$. The convex combination of the points in B, v, is denoted by $v{=}\sum_{j\in B}\beta_j x_j$, where $\beta_j \geq 0$ and $\sum_{j\in B}\beta_j{=}1$. Assume U is the convex hull of A, V is the convex hull of B.

The problem of finding two nearest points in the convex hulls could be written as follows:

$$\min \|u - v\| \tag{3.1}$$

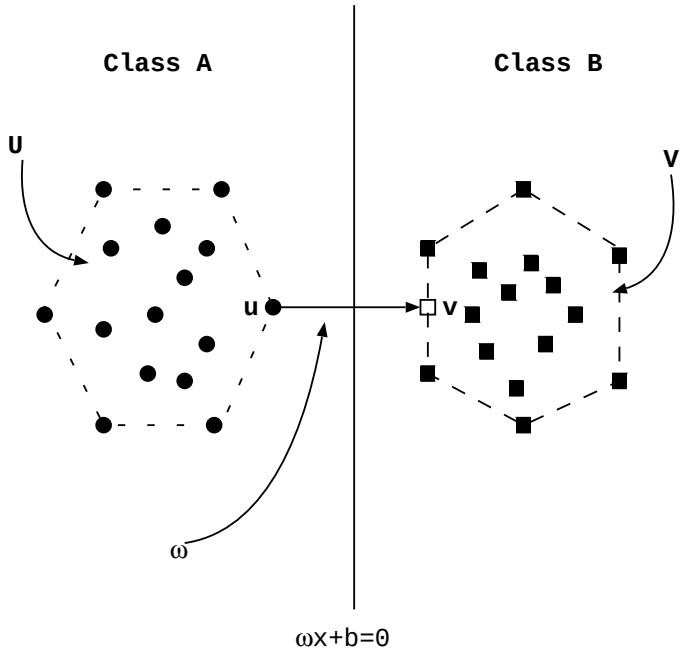

Figure 3.2: Two Closest Points of the Two Convex Hulls Determine the Separating Plane

$$\text{such that } u \in U, \quad and \quad v \in V.$$

If (ω, b) is the optimal solution to Eq. (2.6), and (u, v) is the optimal solution of Eq. (3.1), with the fact that the maximal margin of the two sets$=2/\|\omega\| = \|u - v\|$, and ω has the same direction with $(u - v)$, so

$$\begin{aligned} \omega &= \frac{2}{\|u - v\|^2}(u - v), \\ b &= \frac{\|v\|^2 - \|u\|^2}{\|u - v\|^2}. \end{aligned}$$

We could introduce a new variable, σ, into Eq. (2.6) and then rewrite

$$\begin{aligned} \sum_{i \in A} \alpha_i y_i &= \sigma, \\ \sum_{i \in B} \alpha_i y_i &= \sigma. \end{aligned}$$

Now, we define $\beta_t = \frac{\alpha_t}{\sigma}, \forall t$, so Eq. (2.6) could be rewritten as follows

$$\min \frac{\sigma^2}{2} \sum_t \sum_k \beta_t \beta_k y_t y_k x_t \cdot x_k - 2\sigma \tag{3.2}$$

$$\text{such that } 0 \leq \beta_i \leq 1, \sum_{i \in A} \beta_i = 1, \sum_{j \in B} \beta_j = 1.$$

When $\sigma = \frac{2}{\sum_t \sum_k \beta_t \beta_k y_t y_k x_t \cdot x_k}$, it is equivalent to the following problem:

$$\min \quad \frac{1}{2} \sum_t \sum_k \beta_t \beta_k y_t y_k x_t \cdot x_k \tag{3.3}$$

$$\text{such that } 0 \leq \beta_i \leq 1, \sum_{i \in A} \beta_i = 1, \sum_{j \in B} \beta_j = 1.$$

We now define the matrix P with $y_1 x_1, y_2 x_2, \cdots, y_t x_t$ as its columns. We can obtain the following equation from Eq. (3.3) as

$$\sum_m \sum_k \beta_t \beta_k y_t y_k x_t \cdot x_k = \|P\beta\| .$$

If β satisfies the constraints, then $P\beta = u - v$,where $u \in U$ and $v \in V$. So Eq. (2.6) is equivalent to Eq. (3.1).

The final separating plane is the plane halfway between the two parallel planes: $x \cdot \omega + b = 0$. Note that the maximum distance between the supporting planes yields the distance between the two convex hulls. Then the two closest points of each convex hull must lie on the supporting hyperplanes, otherwise a contradiction exists. It means that either the two supporting planes are not as far as possible, or the two points are not the closest points in the convex hulls. Therefore, the solutions of those two methods are same. When we construct the hyperplane within each space, if there is no degeneracy, we will always obtain the same separating plane [56].

3.1.2 Inseparable Case

If the problems are inseparable, the two convex hulls of the two sets will intersect (See Fig. 3.3). The difficulty in solving these problems is that some

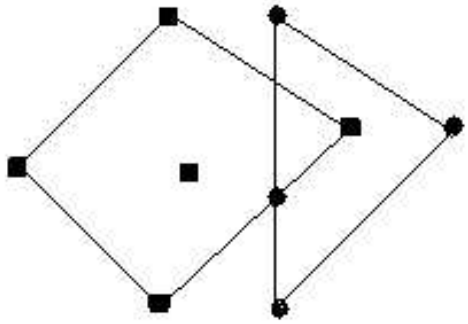

Figure 3.3: The Convex Hulls of Inseparable Case Intersect

points locate in both of the convex hulls, which makes it difficult to classify those points. For the problems belonging to linear classification case, most points of one class will not be in the convex hull of the other class. What we need to do is to restrict the influence of the outlying points and transform the problem into the regular convex hull formulation. It is unsatisfactory to let a point, specially a difficult point, excessively affect the solution. We want to make decision based on lot of points, not on few bad points. For example, we want to solve the problem based on at least k points, then we reduce or contract the convex hull by assigning an upper bound on the multiplier of the combination for each point. The reduced convex hull can be defined as following [6].

Definition 1 (Reduced Convex Hull)

$$\min \|u - v\| \tag{3.4}$$

subject to

$$u = \sum_{i \in A} \beta_i x_i, \quad v = \sum_{j \in B} \beta_j x_j,$$
$$\sum_{i \in A} \beta_i = 1, \quad \sum_{j \in B} \beta_j = 1,$$

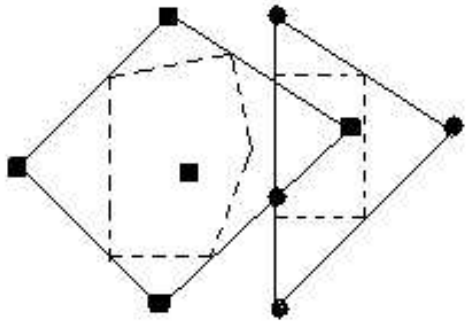

Figure 3.4: Reduced Convex Hulls with K=2

$$0 \leq \beta_j \leq D, \quad 0 \leq \beta_i \leq D,$$
$$D < 1,$$

where D is the upper bound of the multiplier, usually $D = \frac{1}{K}$ and $K > 1$. If $K \leq m$, where m is the number of points in set A, the reduced convex hull will be nonempty.

We reduce the convex hull based on the choice of K, and avoid the intersecting of the two convex hulls by reducing the feasible set away from the boundaries of the convex hulls. In Fig. 3.4, we can find the reduced convex hulls with $K = 2$ no longer intersect. More examples of the reduced convex hull can be found in [56]. If the set has lots of points, reducing the convex hull has little effect. But if the set has few points, the influence will be apparent. When D is small enough, the reduced convex hulls no longer intersect. So we need to choose K sufficiently large enough to make sure that the reduced convex hulls do not intersect. Then we will minimize the distance between the closest points of the reduced convex hulls so that no extreme point or noisy point can excessively influence the solution.

When we choose $D = \frac{1}{k}$, each point can only contribute to the optimal solution no more than $\frac{1}{k}$. It means that the optimal solution depends on at

least $2k$ points. If K is too large, and correspondingly D is too small, the solution will be infeasible. So K must be smaller than the number of points in each set. If we have varying set size of the class, we can choose the different values of D for each class. More properties about the reduced convex hull can be found in [56].

The concept of reducing convex hulls to avoid error is the dual concept of enlarging the margins by softening them to tolerate error. When we add a soft margin term to the linear separable SVM problem Eq. (2.5), the formulation turns into Eq. (2.7), which can solve the inseparable SVM problem. The reduced convex hulls problem is the dual problem of the classic inseparable SVM (See detailed proof in [6]).

So the simple geometric argument of finding the closest points in the convex hulls or reduced convex hulls of the two classes can be used to derive a geometric SVM formulation. But except as the geometric explanation for SVM, basically the convex hull at some dimension d can be used to compute the Delaunay triangulation in dimension $d-1$ (See details in [4] [11]).

At the same time Convex hull can also be one shape descriptor, which also makes a connection between Convex hull and Delaunay triangulation. In many applications in image processing and pattern recognition, an object is specified in terms of a set of points. The shape reconstruction and subsequent identification of this object from the set of input points turns out to be one of the main problems in these fields. Shape descriptors are divided into two categories: those that capture the "external" shape of a set of points as opposed to those that capture what is called the "internal" shape [57].

The external shape of a set of points is obtained by identifying the "essential" extreme points of the set and, among these joining "essential" neighbors [57]. One famous example of "external" shape is the convex hull. Contrarily, the internal shape of a set of points is the shape exhibited by identifying the "essential" internal points of the set and, among these, joining "essential"

neighbors [82]. Some well-known internal shape descriptors are the Delaunay triangulation (DT), the Gabriel graph (GG), and the Relative neighborhood graph (RNG), all of which belong to proximity graph. We discuss these concepts in the following sections.

3.2 Proximity Graph

Many problems in the fields of pattern classification and computational geometry make use of the underlying structure of a set of data points (See examples [37] [88] [68] [52]). The structure referred to is called the *skeleton* or *internal shape* and revealed by means of a *proximity graph*. A proximity graph attempts to exhibit the relation between points in a point set. Two points are joined by an edge if they are deemed *close* by some proximity measure. It is the measure that determines the type of graph that results. Many different measures of proximity have been defined, giving rise to many different types of proximity graphs [12]. One technique for defining a proximity graph on a set of points is to select a geometric region defined by two points of point set and such a region will be referred to as a *region of influence* of the two points. Four such definitions follow.

3.2.1 Voronoi Diagram and Delaunay Triangulation

The Voronoi diagram for a set of points in Euclidean space is one of the fundamental data structures of computational geometry and its properties have been studied extensively [79] [72] [8] [11]. The Voronoi diagram [96] of a set of points can be informally defined as a division of the space according to the nearest-neighbor rule, where each point from the point set is associated with a region of the Euclidean space closest to a given point from the point set. For example, a set of points $M := \{M_1, M_2, ...\}$ such that for each cell corresponding to point M_i, the points q in that cell are nearer to M_i than to

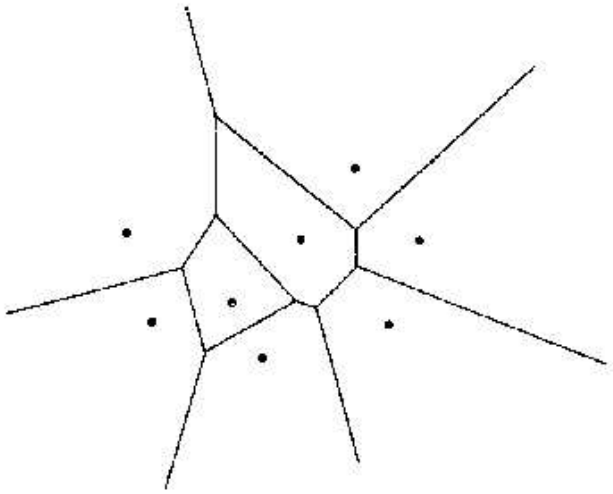

Figure 3.5: The Planar Voronoi Diagram

any other point in M (See Fig. 3.5). In other words, the points q satisfies the following inequality:

$$dist(q, M_i) < dist(q, M_j), \quad M_i, M_j \in M, j \neq i.$$

Along with the investigation of Voronoi diagrams goes the investigation of related constructs. Among them, the Delaunay triangulation [27] is most prominent. It contains a (straight-line) edge connecting two sites if and only if their Voronoi regions share a boundary. It is the dual of Voronoi diagram in a graphtheoretical sense [2]. From Fig. 3.6, it can be seen that Delaunay edges (solid) are orthogonal to their corresponding Voronoi edges (dashed), but do not necessarily insect them.

The Voronoi editing algorithm, as described below, finds a reduced set by using the Voronoi diagram of the training set.

1. Construct the Voronoi diagram for the training set.

2. Visit each node, marking it if all its Voronoi neighbors are of the same class as the current node.

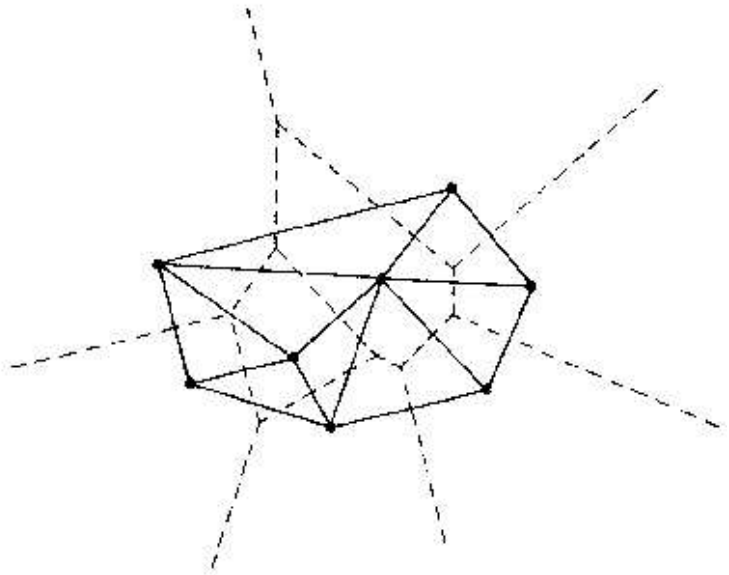

Figure 3.6: Delaunay Triangulation and Voronoi Diagram

3. Delete all marked nodes, exiting with the remaining ones as the Voronoi edited set.

There are many other methods for editing data set, which are based on the well known graph structure computed on the training set. Those methods are derived from Voronoi diagram and make use of the subgraphs of the Delaunay triangulation (DT).

3.2.2 Gabriel Graph and Relative Neighborhood Graph

Gabriel graph of a set of points, V, has an edge between points p and q in V, called Gabriel neighbors, if and only if the diametral sphere of p and q does not contain any other points. The resulting points from the above process make up of the Gabriel edited set. We shall see that the decision boundary can be constructed from those Gabriel neighbors (p and q) such that p and q are of different classes (See Fig 3.7).

The Gabriel edited set is always a subset of the Voronoi edited set because of the fact that a Gabriel graph of a set of points is a subgraph of Delaunay

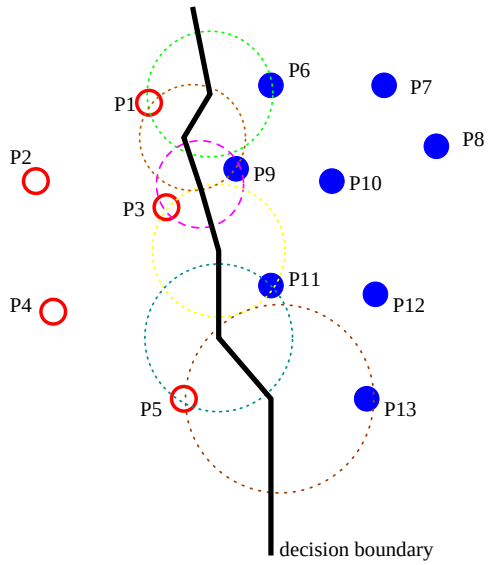

Figure 3.7: Gabriel Neighbor Pairs

triangulation for that set, which only contains those edges in DT(V) that do intersect their dual Voronoi edges. Thus, Gabriel editing, which is the procedure for finding the Gabriel neighbors, reduces the size of the training set more than Voronoi editing. Although the resulting Gabriel editing does not preserve the original decision boundary, the changes occur mainly outside of the zones of interest.

The Gabriel editing algorithm is the same with the Voronoi Editing algorithm except using Gabriel neighbors instead of Voronoi neighbors. Clearly, the Gabriel neighbors can be verified by brute force if for every potential pair of neighbors A and B, we just verify if any other point X is contained in the diametral sphere such that $L_2(A,X) + L_2(B,X) < L_2(A,B)$ where L_2 is the square of the distance between the two points.

Relative neighbors in Relative Neighborhood Graph are defined similarly compared to that of Gabriel graph except for the fact that Relative neighbors use the different definition of "close". Two points p_i and p_j are Relative

neighbors if $d(p_i, p_j) \leq \max[d(p_i, p_k), d(p_j, p_k)]$, $\forall k = 1, ..., n,\ k \neq i, j$. The procedure of searching for the Relative neighbor edited set is almost the same with that of Gabriel neighbor edited set. So we do not present the details of Relative editing algorithm here.

3.2.3 β-skeleton

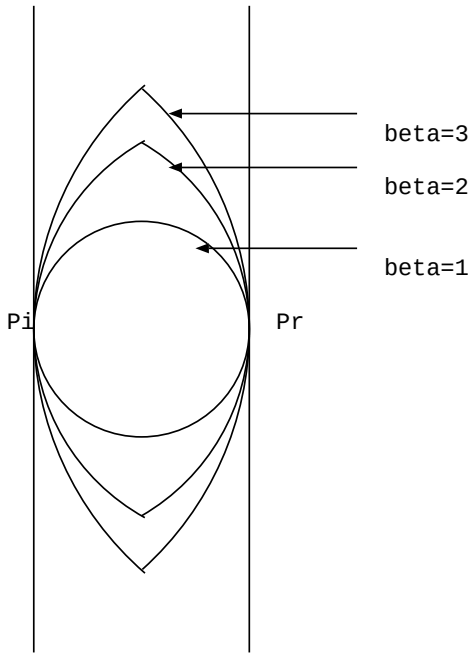

Figure 3.8: The Lune-based β-neighbor for Various β

Another family of graphs, defined by empty discs and thus consisting of subgraphs of DT, is the family of β-skeleton [3]. β-skeletons were introduced by Kirkpatrick and Radke [57] as a class of empty neighborhood graphs. In fact, in [57] both a circle-based and a lune-based version of β-skeleton are proposed. Here, we only consider the latter, as defined below.

Let V be a set of points in $\mathbf{R}^d$, each pair of points $(p, q) \in V \times V$ with a neighborhood $U_{p,q} \subset \mathbf{R}^d$. Let $U = \{U_{p,q} | (p, q) \in V \times V\}$, $\delta(x, y)$ denotes the distance between point x and y, and $B(x, r)$ denotes the circle centered at x with the radius r. That is to say $B(x, r)$=$\{y | \delta(x, y) < r\}$. The neighborhood

$U_{p,q}(\beta)$ is defined, for any fixed β $(1 \leq \beta < \infty)$ as the intersection of two spheres:

$$U_{p,q}(\beta) = B((1 - \frac{\beta}{2})p + \frac{\beta}{2}q, \frac{\beta}{2}\delta(p,q)) \cap B((1 - \frac{\beta}{2})q + \frac{\beta}{2}p, \frac{\beta}{2}\delta(p,q)).$$

A neighborhood graph consists of vertices V and the set of edges E, which is required to satisfy the condition that $(p,q) \in E$ if and only if $U_{p,q}$ has the the property in Eq. 3.5. So β-skeleton of V, $G_\beta(V)$, is a neighborhood graph with the set of edges:

$$(p,q) \in E \text{ if and only if } U_{p,q}(\beta) \cap V = \phi. \tag{3.5}$$

It is easy to see that the radii of the empty discs defining a β-skeleton are not fixed but depend on the inner-point distances in V. Note that $\beta(V)$ is a subgraph of $\beta'(V)$ if $\beta > \beta'$. Different values of the parameter β give rise to different graphs. For $\beta = 1$, Gabriel graph of V is obtained and when $\beta = 2$ it becomes with Relative Neighborhood Graph (See Fig. 3.8).

So the β-neighbor edited Algorithm 1 can be used as a generalized method for constructing the Gabriel neighbor edited set and the Relative neighbor edited set through the different β value's setting.

Algorithm 1: β-neighbor Edited Set Algorithm

Data : $P = \{P_1, P_2, \ldots, P_n\}$,
C_i presents the class of P_i.
Result : E = the edited set.
begin
Let E be $\{\}$
for $P_i \in P$ **do**
Potential β-neighbor set of P_i is initialized to $N_i = \{P_r | P_r \in P,$ and $r \neq i\}$
for $\forall P_r \in N_i$ **do**
1 **for** $\forall P_k \in P$, *and* $k \neq i, k \neq r$ **do**
if P_k *is in* $J(P_i, P_r)$ **then**
$N_i = N_i - P_r$
end
else
select another P_r from N_i, do step 1
end
if $P_k \in N_i$ **then**
if P_r *is in* $J(P_i, P_k)$ **then**
$N_i = N_i - P_k$
end
end
end
β-neighbor set of P_i, $NP_i = N_i$;
end
if $C_i \neq \forall C_r, P_r \in NP_i$ **then**
$E = E + \{P_i\}$
end
end

Chapter 4

Data Editing

The theoretical connections among those concepts in Chapter 3 inspire us to choose the β-neighbor edited set as the training set for SVM training. While along with going deep into the data editing, we find that β-neighbor edited method is only one kind of method for reducing the size of the stored data for the nearest neighbor decision rule. Many other methods can be used to reduce the size of data, which will be briefly introduced in the following sections.

4.1 Hart's Condensed Rule and Its Relatives

In 1968 Hart was the first to propose that kind of algorithm [45]. Hart defined a consistent subset of the data as one that classified the remaining data correctly with the nearest neighbor rule. Then he proposed an algorithm for selecting a consistent subset by heuristically searching for data that were near the decision boundary. We simply describe the algorithm as following. We scan the training dataset iteratively and transfer the misclassified points to the resulted dataset until using the 1-NN decision rule with the resulted dataset does not misclassify any remaining data points in the training dataset. The goal of the algorithm is to keep only a subset of data that are necessary to determine the decision boundary of the training data. The motivation for this is the intuition that the data points far from the decision boundary can be neglected and the

misclassified points must lie close to the boundary [89]. By construction the reduced set resulted classifies all the training data correctly and hence it is referred to here as a *training-set consistent* subset. In the literature Hart's algorithm is called CNN.

CNN may keep points far from the decision boundary. To combat this, Gates [38] proposed what he called the *reduced nearest neighbor rule* or RNN. RNN consists of first performing CNN and then adding a post-processing step. In this post-processing step the elements in the resulted dataset are visited and deleted from the resulted dataset if their deletion does not result in misclassifying any elements in the remaining training dataset.

4.2 Order-independent Subsets

CNN, RNN have the undesirable property that the resulting reduced consistent subsets are a function of the order in which the data are processed. A successful solution to obtaining order-independent training-set consistent subset by generalizing Hart's CNN procedure was proposed by Devi and Murty [28]. Recalling Hart's procedure, the difference between CNN and the method of Devi and Murty [28], which is called the modified condensed nearest neighbor rule (MCNN), is that MCNN initializes the reduced set by transferring one representative of each class from training set to the reduced set in batch mode.

4.3 Minimal Size Training-set Consistent Subsets

The first researchers to deal with computing a minimal-size training-set consistent subset were Ritter et al. [83]. They proposed a procedure, called *selective* nearest neighbor rule (SNN), to obtain a minimal-size training-set consistent

subset, with one additional property that Hart's CNN does not have. Any training-set consistent subset C obtained by CNN has the property that every element of training set is nearer to an element in C of the same class than to any element in C of a different class. While the training-set consistent subset S of Ritter et al. has the additional property that any element of training set is nearer to an element in S of the same class than to any element, in the complete set of a different class. This property of SNN tends to keep the points closer to the decision boundary than does CNN [28].

4.4 Proximity Graph Methods

In 1980 the relative neighborhood graph (RNG) was proposed as a tool for extracting the shape of a planar [88]. There is a vast literature on proximity graphs and part of them have been reviewed in previous sections. All those can be nested together in the following relationship:

$$RNG \subseteq \beta - skeleton(1 < \beta < 2) \subseteq GG \subseteq DT.$$

In 1979 Toussaint and Poulsen [91] were the first to use d-dimension Voronoi diagrams to delete "redundant" members of training set {X,Y} in order to obtain a subset of {X, Y} that implements exactly the same decision boundary as would be obtained using all of {X,Y}. They call this method as *Voronoi condensing* (In this thesis, it is called Voronoi editing). As seen from [90] the nearest neighbor decision boundary with the reduced set is identical to that obtained by using the entire set. So Voronoi edited subset is called "decision-boundary consistent". Clearly decision-boundary consistency implies training-set consistency but the converse does not necessarily hold.

In 1985 Toussaint et al. [90] generalized Voronoi condensing so that it would discard more points in a judicious and organized manner so as not to degrade performance unnecessarily. The dual of the Voronoi diagram is the Delaunay

triangulation. Then Voronoi editing can implemented through computing the Delaunay triangulation. The methods proposed in [90] substitute the Delaunay triangulation by a subgraph of the triangulation. Since a subgraph has fewer edges, its vertices have lower degree on the average. This means the probability that all the graph neighbors of vertex belong to the same class as that of the vertex is higher, which implies more elements of data set will be discarded. By selecting an appropriate subgraph of the Delaunay triangulation one can control the number of elements of {X,Y} that are discarded. Moreover, by virtue of the fact that the graph is a subgraph of the Delaunay triangulation and that the latter yields a decision-boundary consistent subset, we are confident that the former will degrade the performance as little as possible.

So in this thesis, β-neighbor edited algorithm will be carried out to edit the stored data and attain the Gabriel neighbor edited set, Relative neighbor edited set as the trials of training datasets for SVM training.

4.5 Comparing Results of Different Classifiers with Edited Dataset as the Training Set

In this chapter, we conduct the experiments with several databases, some of which are from real world and others are synthetic. Those datasets cover wide range: two/multi-class, large/common data size, real/synthetic, many/few attribute. Real Data used in the experiments include Wine Cultivar discrimination, Glass identification data set, Shuttle, German, Segment and Satimage database. Some of them are available via anonymous file transfer protocol(ftp) from UCI Repository of machine learning databases. The others are from Statlog collection. Furthermore, in the Satimage database, there is one missing class. That is, there are no examples with one class in this dataset. Table 4.1 gives out the number of classes, attributes, and size of each database. If the

Table 4.1: Problem Description on Real Datasets

Problem	**Class**	**Attribute**	**Training Set Size**	**Testing Set Size**
Shuttle	7	9	43500	14500
Satimage	7	36	4435	2000
Segment	7	19	2079	231
German	2	24	900	100
Glass	7	10	192	22
Wine	3	13	160	18

Table 4.2: Problem Description on Synthetic Datasets

Problem	**Class**	**Attribute**	**Training Set Size**	**Testing Set Size**
f2	2	4	900	100
f6	2	6	900	100
uniform.100	2	4	90	10
uniform.1000	2	4	900	100
uniform.10000	2	4	9000	1000
10k.2c	2	15	9000	1000
10k.4c	4	15	9000	1000
10k.8c	8	15	9000	1000

dataset does not include the testing set, we use 90 percent of it as the training set and the remaining as the testing set [22].

Synthetic Data used include three kinds of data. One is generated from the classical classification functions (Function 2 and 6) in [1]. The second one consists of 2 classes of data that are randomly generated in the range of $[0, 1]$ in each dimension, there is an overlapping of data of the two classes in each dimension. The third one is a number of circular clusters, each of which represents a distinct class. Radius of the clusters is 0.2 units. Clusters may overlap with each other. Table 4.2 shows the details of the synthetic datasets. The number behind f represents function type. $uniform$ series belong to the second type. The remaining are in the third type.

Table 4.3: Notation and Description

Gab-heurist	Gabriel Neighbor Edited Heuristic algorithm
Rel-heurist	Relative Neighbor Edited Heuristic algorithm
GG (or Gab-edited)	Gabriel Neighbor Edited set
RNG (or Rel-edited)	Relative Neighbor Edited set
srtree	SR-tree indexing algorithm
kNN	k Nearest Neighbor
svm	SVM with radial basis function
rsvm	Reduced SVM with RBF
C4.5	C4.5 decision tree classification from [80]

Table 4.4: Editing Result of the Original and Normalized Datasets

Problem	Training Set Size	GG Size	RNG Size	GG Size(norm)	RNG Size(norm)
f2	900	704	446	500	274
f6	900	816	653	830	525
10k.2c	9000	400	6	1818	16
10k.4c	9000	1299	36	2424	66
10k.8c	9000	3479	157	4228	200
uniform.100	90	71	32	64	30
uniform.1000	900	594	355	537	361
uniform.10000	9000	4898	3207	4479	3214
Segment	2079	1328	539	1447	402
German	900	883	588	900	620
Satimage	4435	3654	1191	3657	1200
Shuttle	43500	5094	1002	8908	487
Glass	192	19	13	137	68
Wine	160	87	67	134	33

All the notations and their corresponding descriptions used in the experiments are presented in Table 4.3. Moreover, all the experiments are done on the regular training datasets and normalized version of the datasets whose attributes values are normalized into the same region $[0, 1]$.

First of all, we apply the proposed methods described in Chapter 5 to find the Gabriel edited sets(denoted by "Gab-edited") and the Relative edited sets(denoted by "Rel-edited") for all the training dataset. The sizes of different edited set are recorded in Table 4.4 and Fig. 4.2, 4.4, 4.3, 4.5 show the proportion of the set size reduced to the original size. Subsequently, four approaches, kNN, $C4.5$, SVM, and $RSVM$ to the classification problem will be chosen, which are introduced in the following, to show and verify the advantage of the edited set and exhibit the good performance of SVM method (See Fig. 4.1).

For kNN, we test k from 1 to 10, and the results of different selections of k

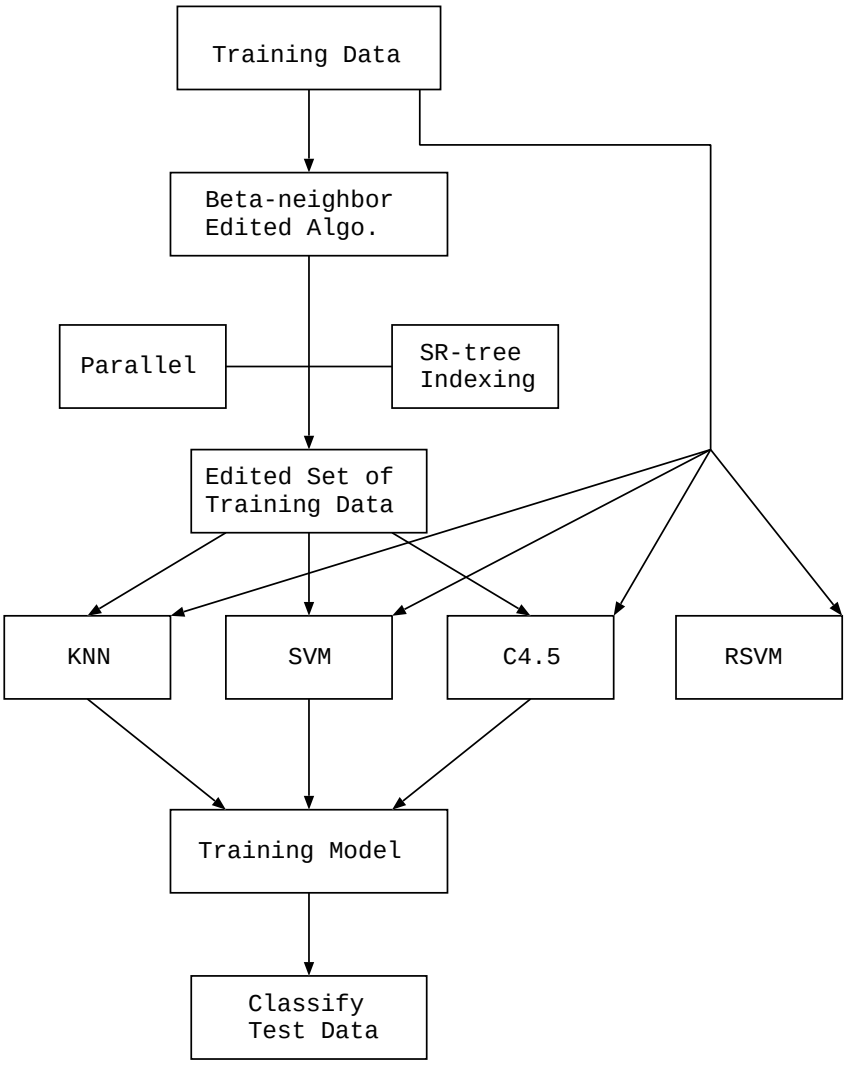

Figure 4.1: The Flowchart of the Experiments

are almost same, so the details will be ignored here. 5 is suitable as the value of k. We also try 3 kinds of voting methods to do the prediction: (1) [Inverse distance] Let d_i be the distance between the query point and the ith nearest neighbor ($i \in [1, ..., k]$). We record the score for each class c_j, which is equal to the sum of the inverse of d_i, where the ith nearest neighbor belongs to class c_j. Then we assign the query point to the class with the highest score; (2) [Inverse Square distance] Similar to the method above, let d_i be the distance between the query point and the ith nearest neighbor ($i \in [1, ..., k]$). The score for each class c_j is the sum of $1/d_i^2$. Finally, the predicted class of the query point is also the class with the highest score; (3) [Majority Voting] According to the result of the k nearest neighbor, the class including the largest number of nearest neighbor is the predicted class. The accuracy of the 3 kinds of methods with the different value of k are similar, so generally the selection of the voting method and the value of k will not effect the experiments' result so much.

Subsequently, C4.5 is chosen as another classifier for testing, which is the most recent version of the decision-tree algorithm that Quilan has been evolving and refining for many years. In the experiment using C4.5, the size of the initial window is set as 5 and the maximum number of objects that can be added to the initial window is 5.

Then, we use Libsvm [23], an integrated software for support vector classification, (C-SVC, nu-SVC), regression (ξ-SVR, nu-SVR) and distribution estimation (one-class SVM) to implement SVM method. Due to the wide application of RBF in function approximation, we train all datasets only with the RBF kernel function to reduce the search space of the parameter sets. Better solutions may be obtained with different choice of γ and C. For each problem, we estimate the accuracy using different parameters pairs of C and γ: $\gamma = [2^4, 2^3, \ldots, 2^{-10}]$ and $C = [2^{12}, 2^{11}, \ldots, 2^{-2}]$ [63]. Then we can use the

Table 4.5: Training and Testing Time Complexity

Methods	**Training**			**Testing**
	Best Case	**Average Case**	**Worst Case**	**Average**
SVM	$O(n)$	$\#iterations \times O(nqd)$		$O(c^2m)$
β-neighbor edited	$O(dn^2)$	$O(dn^3)$	$O(dn^3)$	/
k-Nearest neighbor	/	/	/	$O(mnlogn)$

accuracy as criterion to choose the optimal parameters. At last the best parameter is selected from a five-fold cross validation on the training data. The experiments for RSVM (the software is downloaded from [64]), are conducted under the similar parameter settings. In RSVM, the methods to obtain the optimal solution are various. Here, we choose Smooth SVM to conduct the experiments with the help of ATLAS [98]. Since generally RSVM uses the normalized datasets in its experiments, we do not record the results of RSVM on the original datasets here.

All the program codes use language C and the results are the average of several runs. All the experiments are done under a SUN Ultra-Enterprise machine running SunOS5.7 with 8GB Main Memory. The four benchmarks, Edited set sizes with different choice of proximity graph, Accuracy, Training and Testing time are recorded. We compare the four benchmarks on four classification techniques, give our observations, and analyze the observations in the following sections.

4.5.1 Time Complexity

Table 4.5 summarizes the theoretical analysis from the research result of Joachims [53] and Bhattacharya [9], where n is the size of training set, m is the size of testing set, d is the dimension of samples, c is the number of the classes.

Libsvm implements an SMO type algorithm where the size of the working

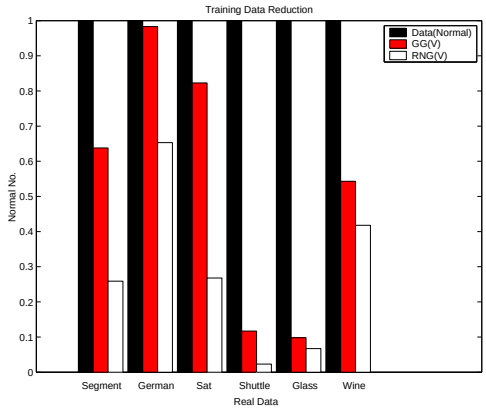

Figure 4.2: Size of Real Data vs Size of Different Beta-neighbor (in percentage)

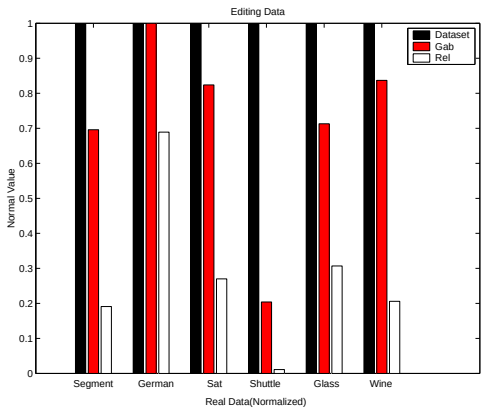

Figure 4.3: Size of Real Data (normalized) vs Size of Different Beta-neighbor (in percentage)

set is restricted to two. SMO [76] is the decomposition method to solve SVM, which is an iterative process and in each iteration the index set of variables is separated into working set and the fixed set. Then the sub-problem on variables of the working set is minimized.

In the SMO algorithm q, the size of the working set, is equal to 2. In each iteration, most time is spent on the kernel evaluations which can be used to compute the q rows of the Hessian. Therefore, the total complexity is

$$\#iterations \times O(nqd)$$

where we assume each kernel evaluation costs $O(n)$ [64].

With the exploiting of the β-neighbor edited method and indexing structure, we can stabilize the performance of SVM in the average and worst case by reducing the size of training set before SVM Training.

4.5.2 Editing Size of Training Data

Table 4.4 shows the Gab-edited size and the Rel-edited size of the original training sets and their normalized version. From the values shown in Table 4.4 and Fig. 4.2, 4.4, 4.3, 4.5 we find :

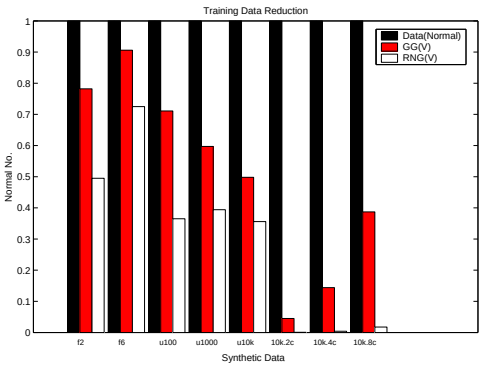

Figure 4.4: Size of Synthetic Data vs Size of Different Beta-neighbor (in percentage)

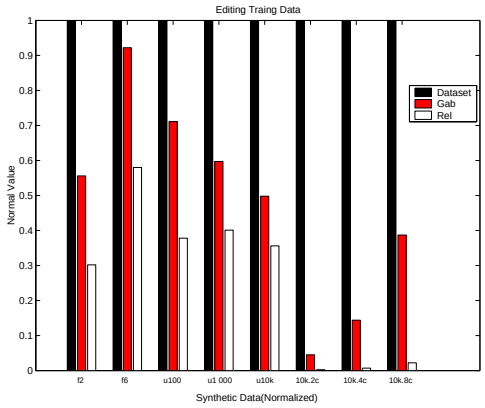

Figure 4.5: Size of Synthetic Data (normalized) vs Size of Different Beta-neighbor(in percentage)

1. Up to 90% data points in the training set will be thrown away after Gabriel editing. Especially, for the large dataset, the data editing behaves well.

2. The size of Relative neighbor edited set is smaller than that of Gab-edited, which means that searching Relative neighbors prunes more data points.

3. Comparing with the results of original datasets and normalized ones, we find that the normalized synthetic datasets are pruned more using Gabriel editing, while for the real datasets, they keep more data points through editing if the datasets are normalized. The normalized *german* dataset is even not discarded any data during the Gabriel editing. Real datasets usually have different value scale for different attributes. When they are normalized, the differences among the attributes and the points are reduced, so the points are kept more after Gabriel editing. For synthetic datasets, the differences among the scales of the features are not very significant. Normalizing makes the synthetic data converge, so Gabriel editing can prune more.

4. Relative editing does not exhibit the same performance as that of Gabriel editing. Except for $f2$ and $f6$ dataset, the normalized synthetic Rel-edited sets retain more points than those of the corresponding original data. Relative editing prunes too many points of the training data, which makes the explanations for the size of Rel-edited set less meaningful.

4.5.3 Accuracy

Table 4.6: Accuracy Results of the Different Classification Methods

Problem	svm-all	svm-gab	svm-rel	nn-all	nn-gab	nn-rel	c4.5-all	c4.5-gab	c4.5-rel
f2	85	84	83	81	80.2	68.4	**100**	98	99.6
f6	66	66	66	57	55.6	52.4	94	**94**	94.8
10k.2c	100	**100**	100	100	100	100	100	100	90.6
10k.4c	**100**	100	100	100	100	100	100	100	88.74
10k.8c	**100**	100	100	100	100	100	100	100	95
uniform.100	100	100	90	92	92	86	100	100	88
uniform.1000	**91**	91	86	85	85.2	80.6	86	83	85.2
uniform.10000	**84**	84	84.4	83	82.9	81.1	82.7	82.7	83.7
segment	95	85	72.7	93.9	93.3	79.7	97.4	**97.8**	92.4
german	**81**	81	75	70	69.8	68.8	66	69	68.8
sat	**92**	92	80.3	88.7	88.9	83.4	84.3	85.3	77.2
shuttle	99.8	99.8	99.4	99.8	99.3	90.1	**100**	100	99.9
glass	**100**	91	90.9	97.1	60.7	48.1	95.5	68.2	53.6
wine	88	61	61	72.2	63.3	55.4	88.9	83.3	**94.4**

The primary metric for evaluating classifier performance is *classification accuracy*- the percentage of *test* samples that are correctly classified.

Table 4.6 and Fig. 4.6, 4.8, 4.10 show the accuracy of the different classification methods for the whole training dataset (denoted by "all"), the Gab-edited set (denoted by "gab") and the Rel-edited set (denoted by "rel"). From the table and figures, we can get the following observations:

1. The performance of a classification method depends greatly on the characteristics of a special dataset.

2. In general Gab-edited sets preserve the high accuracy, almost the same as the original training dataset except for the datasets *glass* and *wine*, whose size of the training dataset is so small that they are easily over-pruned.

Table 4.7: Accuracy Results of the Different Classification Methods(on normalized sets)

Problem	rsvm	svm-all	svm-gab	svm-rel	nn-all	nn-gab	nn-rel	c4.5-all	c4.5-gab	c4.5-rel
f2	93	92	91	87	88	90	87	99	**100**	100
f6	80	90	92	88	82	82	78	**96**	92	96
10k.15d.2c	100	**100**	100	100	100	100	100	100	100	93.7
10k.15d.4c	100	100	100	100	**100**	100	100	100	100	96.8
10k.15d.8c	100	**100**	100	100	100	100	100	100	100	100
uniform.100.4d	100	**100**	100	100	100	100	90	100	100	70
uniform.1000.4d	85	87	**88**	88	85	85	84	86	83	86
uniform.10000.4d	82.9	84.7	**84.8**	84.7	84.7	84.8	83.4	82.7	84.7	83.2
segment	95.6	96.9	95.2	91.7	93.5	94.3	90	97.8	**98.2**	87
german	**80**	80	80	79	72	72	70	67	67	69
sat	89.8	**91.8**	91.8	88.1	90.1	90.1	88.7	86	86.6	76.65
shuttle	99.8	99.92	99.5	99.84	99.8	96.3	76.5	99.9	**99.97**	99.8
glass	86.3	**95.4**	95.4	95.4	81.8	81.8	86.3	95.4	95.4	95.4
wine	94.4	**100**	94.4	83.3	94.44	94.4	88.9	88.9	88.9	94.4

3. For the *satimage* dataset, the Gab-edited set is more accurate than the original training dataset, since there may exist some noise in the data points of the original dataset.

4. With the SVM method, Rel-edited sets preserve the high accuracy on most of the synthetic datasets compared to the Gab-edited and original sets. Furthermore, they even provide better results on the datasets $uniform$10,000. While for *satimage*, *segment* and *wine* datasets, they perform worse.

5. With the kNN method, the performance of Gab-edited sets is better than the Rel-edited and worse than the original ones.

6. With $C4.5$ Rel-edited sets usually do not perform better than Gab-edited sets and original ones except on $f6$, $uniform10,000$, and *wine* datasets.

7. Therefore, the selection of original dataset or edited set as training set is varying with the real problem. Every problem has its suitable choice.

Taking into account that the ranges of attribute are different, we also carry out the experiments on the normalized datasets, whose attributes variables are scaled in [0,1]. From Table 4.7, Table 4.6 and Fig. 4.7, 4.9, 4.11, we can see that

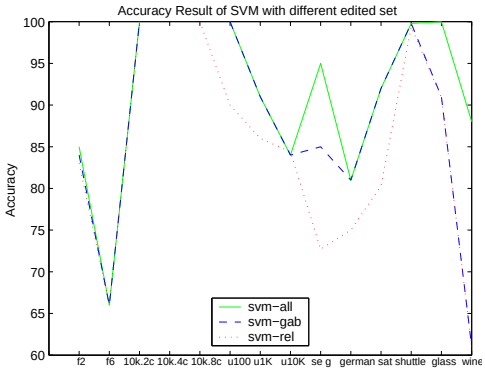

Figure 4.6: Accuracy Result of SVM with Different Edited Sets

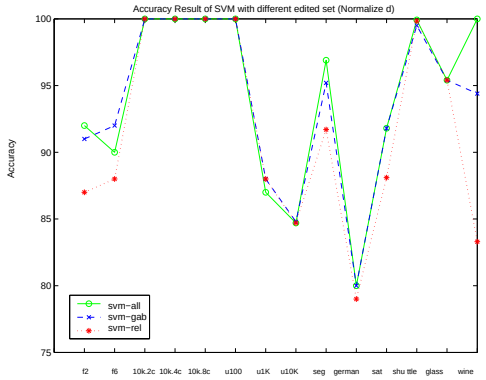

Figure 4.7: Accuracy Result of SVM with Different Edited Sets (on normalized dataset)

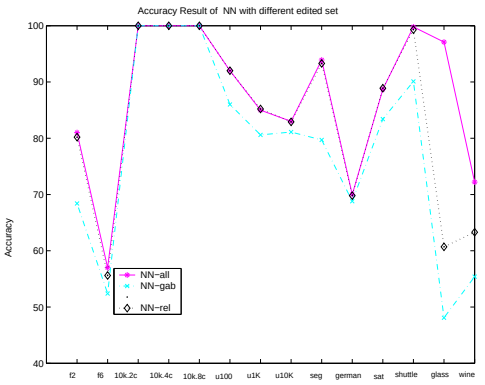

Figure 4.8: Accuracy Result of kNN with Different Edited Sets

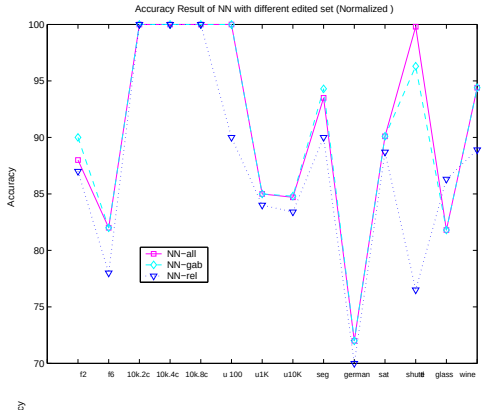

Figure 4.9: Accuracy Result of kNN with Different Edited Sets (on normalized dataset)

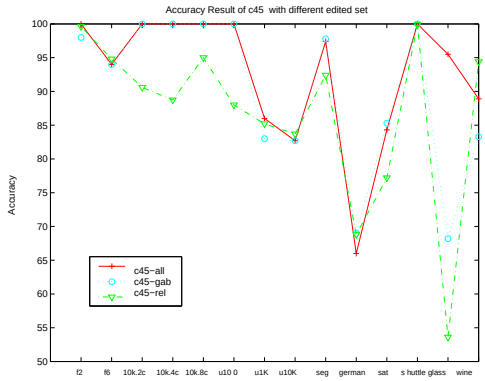

Figure 4.10: Accuracy Result of C45 with Different Edited Sets

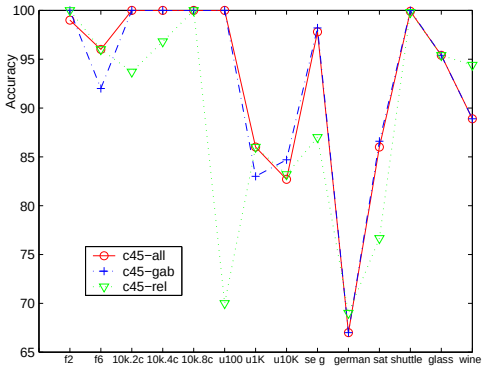

Figure 4.11: Accuracy Result of C45 with Different Edited Sets (on normalized dataset)

1. The accuracy for datasets *glass* and *wine* increase since the edited sets are larger than those obtained from the experiments with the original datasets. There are only 19 points left in the Gabriel edited set of *glass* and 13 in Relative edited set after editing the original dataset, while 137 Gab-edited points and 68 Rel-edited points for *glass* are obtained when experiments are done on the the normalized version of the dataset. Similar comparisons are made on the editing results of *wine*.

2. In the $C4.5$ method, for $f6$, $uniform$ 1000, *german*, and *wine* datasets, Rel-edited sets perform better than Gab-edited set and original dataset. The performance of applying Rel-edited sets to kNN is not as good as that of using the other two.

3. As shown in the tables above, the value in bold is the highest accuracy of the different methods with different training set. It shows that in general Standard SVM is better than other methods.

4. RSVM method preserves the performance on the prediction accuracy compared to Standard SVM in most cases. For the artificial datasets its performance is a little worse than Standard SVM with Gab-edited set.

While for the real datasets, it usually performs better than SVM with Gab-edited set.

4.5.4 Efficiency

Table 4.8: Time for Editing the Different Datasets

dataset	*Gab – heurist*	*Rel – heurist*
f2	25	22.2
f6	40	43.2
10k.15d.2c	9137	3233
10k.15d.4c	857547	54341
10k.15d.8c	1250286	446719
uniform.100	0.2	0.13
uniform.1000	26	19.6
uniform.10000	3088	2464
segment	803	537
german	289	141.5
sat	40703	14240
shuttle	-	-
glass	0.73	0.71
wine	1.12	1.03

Table 4.8 shows the editing time obtained from the application of the heuristic β-neighbor algorithm 1. The Gabriel editing time is always more than the time to find the Relative neighbor edited set except for the dataset $f6$. The observation verifies the fact that the Relative editing prunes more points than the Gabriel time during the editing procedure, which reduces the candidate points to test them more quickly. The heuristic algorithms fail to deal with the *shuttle* dataset since when the programs create the matrices to store the pair information of potential β-neighbors, the needed allocation of memory exceeds the ability of the machine.

The training and testing time for different datasets are reported in Tables 4.9, 4.10, 4.11 and 4.12. We measure the training time (denoted by "train") and testing time (denoted by "test") for the whole training dataset (denoted by "all"), the Gab-edited set (denoted by "gab") and the Rel-edited

Table 4.9: Time for Training and Testing (in second) of KNN and C4.5

data	c45-all train+test	c45-gab train+test	c45-rel train+test	5NN-all test	5NN-gab test	5NN-rel test
f2	0.87	0.86	0.84	0.3	0.17	0.09
f6	5.92	5.61	3.5	0.47	0.42	0.28
10k.15d.2c	1.77	0.48	0.07	39.2	7.4	0.35
10k.15d.4c	11.46	2.9	0.17	25	6.3	0.66
10k.15d.8c	40	14.1	0.73	20.2	9.89	1.21
uniform.100.4d	0.1	0.06	0.02	0.03	0.01	0
uniform.1000.4d	1.96	1.84	1.84	0.32	0.2	0.14
uniform.10000.4d	43.9	35.4	31.7	8.78	5.13	3.98
segment	54.07	36.67	14	2.42	1.65	0.55
german	25.75	25.6	18.7	1.3	1.3	0.89
sat	517.9	456	164.6	50.5	44	19.2
shuttle	127	12	1.55	238	167.6	25.7
glass	0.34	0.26	0.16	0.08	0.05	0.03
wine	0.63	0.6	0.12	0.08	0.06	0.02

set (denoted by "rel"). Since kNN does not need any training, we only measure the testing time for it. Note that the total time for kNN classification should be the time for finding the edited set in Table 4.8 and the testing time. The following observations and discussion are based on the experimental results shown in Fig. 4.12, 4.13, 4.14, 4.15, 4.16, 4.18, 4.17, and 4.19.

1. Even we use ATLAS to optimize the matrix computation, the training time of RSVM with the large datasets is greater than those of Standard SVM except for *shuttle* dataset.

2. The algorithm with edited set as training set reduces the training and testing time for SVM, C4.5 and the testing time for kNN since editing procedure pruned many data points.

3. Generally, the time taken by original datasets in the $C4.5$ or kNN method and that by normalized datasets are almost the same. The slight differences between them are caused by the difference between the edited set sizes for original set and those for the normalized ones. So does for the SVM method with most of the datasets.

Table 4.10: Time for Training and Testing (in second) of SVM

data	svm-all train	svm-gab train	svm-rel train	svm-all test	svm-edit test	svm-rel test
f2	3.25	3.37	1.3	0.05	0.09	0.02
f6	1.7	1.53	0.42	0.17	0.15	0.09
10k.15d.2c	152	0.33	0.01	4.92	0.64	0.11
10k.15d.4c	114	1.92	0.01	7.1	0.88	0.14
10k.15d.8c	171	33	0.05	10.28	2.3	0.3
uniform.100	0.04	0.01	0	0.01	0.01	0.01
uniform.1000	3	2.57	0.25	0.04	0.04	0.03
uniform.10000	8213	3403	7.81	2.29	2.28	1.22
segment	19	10.9	1.12	0.54	0.56	0.28
german	5.29	5.18	0.75	0.19	0.17	0.16
sat	40.86	27.04	6.16	13.43	12.16	7.13
shuttle	287	19	2.5	13.2	9.2	7.47
glass	0.25	0.01	0.01	0.01	0.01	0.01
wine	0.27	0.12	0.1	0.02	0.01	0.01

4. For large datasets, $10k.15d.2c$, $10k.15d.4c$, $10k.15d.8c$, $uniform.10k$, the reported training time of SVM with the normalized datasets is less than that of the original dataset. In fact, the cross validation time for both training are similar. But the pair of parameter for SVM training will affect the training time hugely. With the edited set as the training set, the training time of SVM on the dataset $uniform.10k$ or *shuttle* is reduced much compared to that with the original datasets.

5. During the procedure of SVM Training, the sum of cross validation time to select parameter pair is larger than C4.5 training, even than the time of finding the edited sets for the common-size datasets. While for large datasets, result shows that the training time for SVM and C4.5 are faster than that of finding the edited sets. So the heuristic editing algorithms are not suitable enough to be applied here.

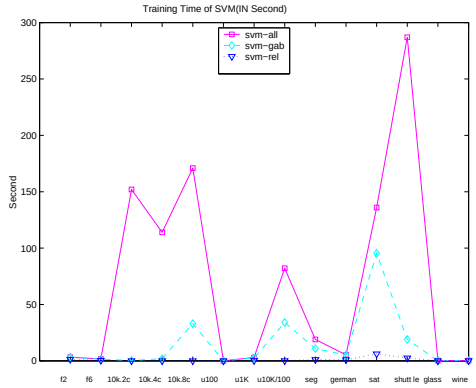

Figure 4.12: Training Time of SVM (in second)

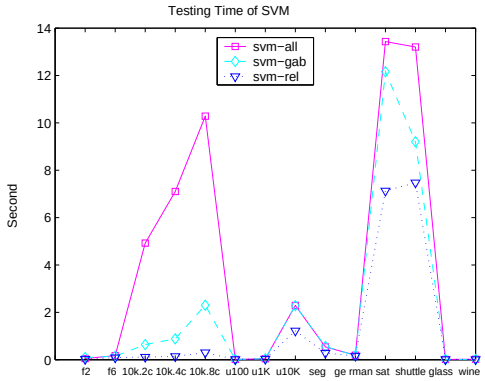

Figure 4.13: Testing Time of SVM (in second)

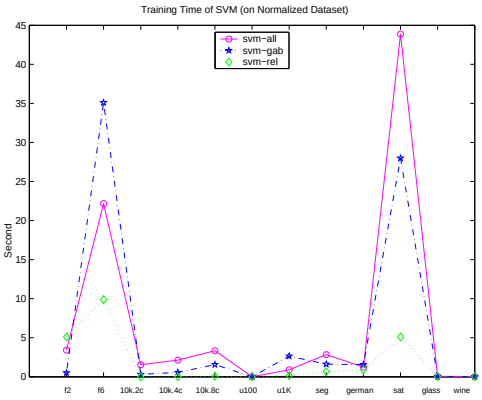

Figure 4.14: Training Time (in second) of SVM (on normalized dataset)

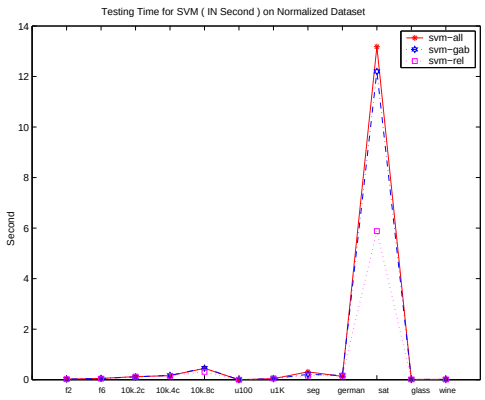

Figure 4.15: Testing Time (in second) of SVM (on normalized dataset)

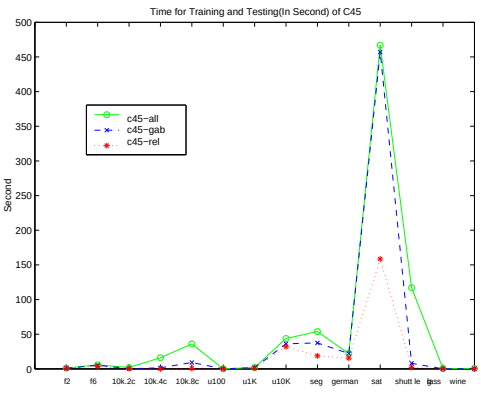

Figure 4.16: Time for Training and Testing (in second) of C45

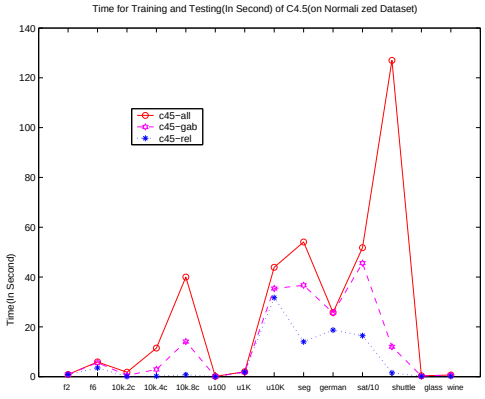

Figure 4.17: Time for Training and Testing (in second) of C45 (on normalized dataset)

Table 4.11: Time for Training and Testing (in second) of KNN and C4.5(on normalized set)

data	c45-all train+test	c45-gab train+test	c45-rel train+test	5NN-all test	5NN-gab test	5NN-rel test
f2	0.85	0.79	0.84	0.28	0.16	0.08
f6	5.76	5.48	3.42	0.46	0.41	0.26
10k.15d.2c	1.73	0.46	0.09	37.08	7.17	0.32
10k.15d.4c	11.26	2.8	0.16	24.47	6.26	0.63
10k.15d.8c	38.8	13.94	0.75	20.52	10.08	1.14
uniform.100	0.1	0.06	0.03	0.02	0.01	0
uniform.1000	1.92	1.92	1.84	0.31	0.2	0.13
uniform.10000	42.7	36.77	31.08	8.48	4.97	3.83
segment	49.84	36.62	13.87	2.97	2.15	0.72
german	25.03	25.03	18.4	1.3	1.3	0.87
sat	504.56	449.2	162.36	49.02	43.34	18.89
shuttle	261.5	11.67	1.53	79.77	20.55	29.05
glass	0.34	0.28	0.13	0.08	0.06	0.02
wine	0.62	0.59	0.14	0.07	0.07	0.01

4.5.5 Summary

In this chapter, we have presented the experiments, which regard the edited set obtained before as the training set for SVM, k-NN and C4.5 methods. Then we compare the experimental results with that of the classifiers with the original sets as training sets and RSVM with original sets. Compared to the original methods, the classification methods with the edited set as the training set can preserve the high predictive accuracy. However, from the viewpoint of efficiency, they can not outperform the traditional methods. So we need to use other techniques to speed up the editing procedure, which will be discussed in next chapter.

Table 4.12: Time for Training and Testing (in second) of SVM (on normalized set)

data	svm-all train	svm-gab train	svm-rel train	svm-all test	svm-edit test	svm-rel test	rsvm train	rsvm test
f2	3.41	0.51	5.11	0.02	0.02	0.02	12.2	0.01
f6	22.17	35.1	9.88	0.05	0.05	0.04	10.7	0.03
10k.15d.2c	1.54	0.31	0.01	0.12	0.11	0.12	406	0.4
10k.15d.4c	2.13	0.56	0.01	0.16	0.17	0.15	3742	3.89
10k.15d.8c	3.34	1.56	0.08	0.45	0.46	0.31	519.9	8.23
uniform.100.4d	0.01	0.01	0.01	0	0	0	0.02	0
uniform.1000.4d	0.9	2.65	0.14	0.04	0.04	0.05	7.79	0.03
uniform.10000.4d	51.5	20.9	14.76	2.72	2.62	2.35	162	0.17
segment	2.84	1.61	0.65	0.31	0.21	0.16	51	0.61
german	1.24	1.56	0.89	0.13	0.17	0.16	6.3	0.03
sat	43.87	27.99	5.11	13.18	12.2	5.88	612	12.1
shuttle	312.9	48.06	1.38	5.74	4.82	3.74	10029	56.79
glass	0.06	0.05	0.01	0.01	0.01	0.01	0.19	0.01
wine	0.04	0.03	0.02	0.01	0.01	0.01	0.06	0.02

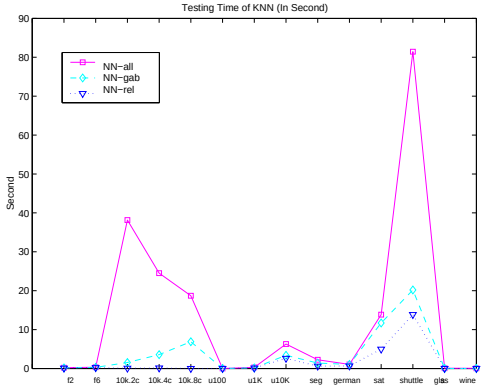

Figure 4.18: Time for Training and Testing (in second) of kNN

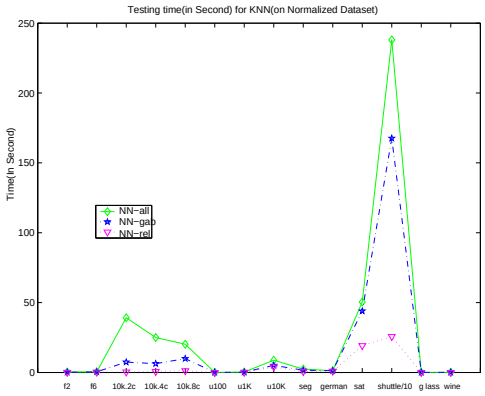

Figure 4.19: Time for Training and Testing (in second) of kNN (on normalized dataset)

Chapter 5

Techniques Speeding Up Data Editing

According to the experimental results, although the prediction accuracy is preserved high, the efficiency of our methods is not satisfactory. So before we preprocess the training dataset and take the different edited set as training set for SVM [104], we propose to shorten the procedure of finding the edited set.

The most straightforward idea is to use parallel computing to speed up the editing procedure. Our second approach to speed up the editing procedure is using the indexing structure. Recall the procedure of editing the dataset. Assuming one point in the dataset as a query point, finding the β-neighbors of the query point is equal to finding the " closest" points (may not exist), whose class attributes are different with that of the query point. Repeatedly applying the searching step above for each point in the dataset, we can get the edited set by summarizing the β-neighbors of every point. So we can apply the indexing structures for multi-dimensional searching in our β-neighbor edited algorithm to improve its performance. Finally, we combine both of the techniques with the β-neighbor edited algorithm and record the corresponding experimental results.

In this chapter, the basic idea of the two techniques will be introduced firstly. Then the details of their implementations in our methods and the

corresponding experimental results will be indicated later.

5.1 Parallel Computing

5.1.1 Basic Idea of Parallel

Algorithms for some problems can be executed quickly while those for others take a very long time. It is intuitive for people to design better algorithms, which can find the solutions for the problems faster. Since the execution time also depends on the processing speed of the computer, efforts have been made to increase the speed of computers. However, the current demands for high-speed computing cannot be satisfied only by using faster hardware components. It is necessary for us to look for a method to satisfy the demand from the viewpoint of software. Parallel processing is one kind of the solution and it performs independent operations simultaneously so that the overall computation time is reduced.

Parallel computing is a central and important problem in many computationally intensive applications, such as image processing, robotics, and so forth. Given a problem, the parallel computing is the process of splitting the problem into several subproblems, solving the subproblems simultaneously, and combining the solutions of subproblems to get the solution to the original problem [100].

Parallelism can be achieved at different levels. For example, if ten jobs are given, which are different in nature and pairwise independent, then these ten jobs can be given to ten different machines. This is a high-level parallelism, because the jobs are implemented in parallel. This is usually called *job level parallelism* or *program level parallelism*. In order to increase the efficiency of the program further, we can go in for the next level of parallelism—Each job can be divided into smaller subtasks. These subtasks can be executed in

parallel, and then results are consolidated to get the final result. For each subtask, there will be a subprogram, and these subprograms may be executed in parallel. This is usually called *subprogram level parallelism*. In any program (subprogram) there are several statements. These statements may be done in parallel. This possibility is called *statement level parallelism*. In a statement, several operations are carried out. We can think of parallelizing these operations, and this is called *operation level parallelism*. Usually, any operation consists of several micro-operations. If micro-operations are done in parallel, this will be called *micro-operation level parallelism* [100] (See Fig 5.1).

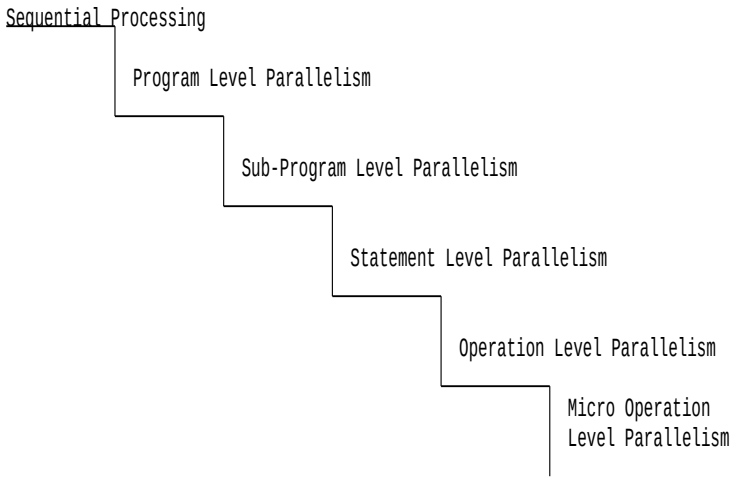

Figure 5.1: Levels of Parallelism

Programming with threads instead of conventional processes is increasingly popular because threads are less expensive than processes and because they provide a trivial mechanism for sharing data. For example, a high-performance Web server might assign a separate thread for each open connection to a Web browser, with each thread sharing a single in-memory cache of frequently requested Web pages. Another important factor in the popularity of threads is the adoption of the standard Pthreads (Posix threads) [21] interface for manipulating threads from *C* programs. The benefit of threads has been known

for some time, but their use was hindered because each computer vendor developed its own incompatible threads package. As a result, threaded programs written for one platform would not run on other platforms. The adoption of Pthreads in 1995 has improved this situation immensely. Posix threads are now available on most systems, including Linux.

5.1.2 Details of Parallel Technique

A thread is a unit of execution, associated with a process, with its own thread ID, stack, stack pointer, program counter, condition codes, and general-purpose registers. Multiple threads associated with a process run concurrently in the context of that process, sharing its code, data, heap, shared libraries, signal handlers, and open files.

Historically, hardware vendors have implemented their own proprietary versions of threads. These implementations differed substantially from each other making it difficult for programmers to develop portable threaded applications.

In order to take full advantage of the capabilities provided by threads, a standardized programming interface was required. For UNIX systems, this interface has been specified by the IEEE POSIX 1003.1c standard (1995). Implementations which adhere to this standard are referred to as POSIX threads, or Pthreads.

Pthreads are defined as a set of C language programming types and procedure calls, implemented with a *pthread.h* header/include file and a thread library - though the this library may be part of another library, such as *libc*.

The primary motivation for using Pthreads is to realize potential program performance gains. In order for a program to take advantage of Pthreads, it must be able to be organized into discrete, independent tasks which can execute concurrently [71].

In this thesis, finding the β-neighbor set for one point is not relevant with

that for another point, so it can be looked as an independent task. Then we can do parallel in the procedure of finding the distinct edited set with pthreads.

For the general β-neighbor edited algorithm, according to the number of threads assigned, we randomly divide the whole dataset into t subsets, where t is equal to the number of threads. For each data point in the subset, we find the β-neighbors for it. We run the program in the multiple threads environment, and summarize the results of all of the subsets. The experiments concerning the effect of the various number of threads on the computation time will be shown in the next subsection.

5.1.3 Comparing Effects of the Choice of Number of Threads on Efficiency

For parallelism, we choose several trials of the number of threads to test the effect of the number of the threads on the execution time. At the same time, the number of processors will be allocated with the requirement of the threads. As shown in the algorithms before, the step to find one data point's β neighbor is independent with that of any other point. When the program is run in the multi-thread environment, the only common writing operation of the different threads is to revise the status of each point in the dataset, from which we can judge whether the point is in the resulting edited set. Then we use a flag to record the status of each data point, and if it is in the resulting set, the corresponding flag will be set to 1. Furthermore, setting the bit to 1 repeatedly do not change the final result of the edited set. So it is not necessary for us to think about the communication among the threads and then the number of the threads will not affect the editing results.

According to the results of experiments, the conjecture above is approved again that the edited sets obtained are the same, although the number of threads are different. So we only record the running time of the algorithms.

Table 5.1: Time for Finding Gabriel Edited Set for Different Degree of Parallelization (in second)

Problem	Number of threads			
	1	3	6	8
f2	25	30	31	30.9
f6	40	46.2	48.3	48.13
10k.15d.2c	9137	9036	10343	11460
uniform.100.4d	0.2	0.3	0.28	0.26
uniform.1000.4d	26	32.4	33.1	33.48
uniform.10000.4d	3088	3798	3907	3923
segment	803	863	861	867
german	289	312	312	313.7
sat	40703	42367	21916	17066
glass	0.73	0.58	0.57	0.57
wine	1.12	0.88	0.89	0.93

From Table 5.1 and Table 5.2, we can obtain the following observations.

1. From Table 5.1, we can see that generally the Gabriel edited algorithm with parallelism do not get the time gain but increases the execution time. Since the Gabriel editing time for the datasets $10k.15d.4c$ (857547) and $10k.15d.8c$ (1250286) is huge, even if we make full use of the 8 threads, at least $\frac{1}{8}$ of the costing time obtained from the heuristic Gabriel algorithm is needed, which is still not demanding. Then for these two datasets we do not do the parallelism in the Gabriel editing procedure.

2. When the number of the thread is set to 3, the Relative editing time is reduced at most 20%, except for the larger datasets such as $10k.15d.2c$, $10k.15d.4c$, sat, $uniform.1000.4d$, $uniform.10000.4d$.

3. The parallel Relative neighbor edited algorithm behavior well on $10k.15d.4c$, sat, when the number of thread increases. For the other datasets, the increasing threads do not bring the performance gain. So $Pthread$ is not quite useful in the heuristic β-neighbor edited algorithm.

Table 5.2: Time for Finding Relative Neighbor Edited Set for Different Degree of Parallelization (in second)

Problem	Number of threads			
	1	3	6	8
f2	22.24	18.7	18.8	19.1
f6	43.18	33.8	33.9	34.4
10k.15d.2c	3233	2430	2348	2562
10k.15d.4c	54341	31642	16688	16404
uniform.100.4d	0.13	0.16	0.17	0.18
uniform.1000.4d	19.59	16.57	16.29	16.57
uniform.10000.4d	2464	1921	1933	1933
segment	537	476	477	475
german	141.5	112	110	112
sat	14240	11870	6212	5340
glass	0.71	0.73	0.7	0.72
wine	1.03	0.96	0.94	0.97

4. When the number of the threads increases, in theory the heuristic Relative editing time of $10k.15d.8c$ (446719) could be reduced to $\frac{1}{k}$ of the time, where k is the number of threads. Even though we can obtain that, it is wide of the training time of standard SVM. So we do not use the parallel Relative neighbor edited algorithm to deal with it.

5. To sum up, compared with the training of standard SVM, only with the parallel technique the performance of the β-neighbor edited algorithms is still not acceptable.

With the analysis in the previous section, the heuristic β-neighbor edited algorithms are not suitable for *shuttle* dataset, so we will not do parallelism on it. This condition also makes us think of other techniques to reduce the demand of the huge memory before parallelizing the searching procedure. Finding the edited subset of the dataset in multi-dimensional spaces can be regarded as a kind of searching, which targets to obtain the pairs of "close" points in the dataset. So the indexing structure can be an effective method to improve the performance of the β-neighbor edited algorithm further. Details will be shown

in next section.

5.2 Tree Indexing Structure

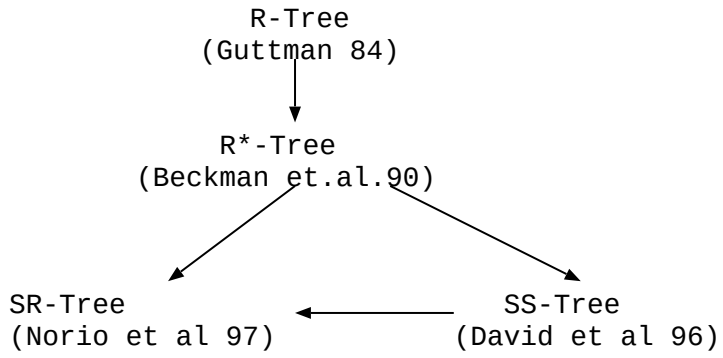

Figure 5.2: Tree Indexing Structure History

In this section, we will introduce and briefly discuss the important index structures for high-dimensional data spaces and the history of tree-indexing structure growing is showed in Fig 5.2. We first describe index structures using minimum bounding rectangles as page regions (after partitioning hierarchically the data space into a manageable number of smaller subspaces, the subspaces are called page regions) such as the R-tree, the $R^{\star}$-tree. We continue with the structures using bounding spheres such as the SS-tree and conclude with structures using combined regions: the SR-tree. Then we will describe the algorithms based on SR-tree structure in detail and improve the performance of the algorithm with pruning search place, which will be discussed later.

5.2.1 R-tree and $R^{\star}$-tree

The R-tree [41] family of index structures uses solid minimum bounding rectangles (MBR) as page regions. An MBR is a multidimensional interval of the

data space (i.e., axis-parallel minimal approximations of the enclosed point set) There exists no smaller axis-parallel rectangle also enclosing the complete point set. Therefore, every $(d-1)$ dimensional surface area must contain at least one datapoint. Space partitioning is neither complete nor disjoint. Parts of the data space may not be covered at all by data page regions. Overlapping between regions in different branches is allowed, although overlaps deteriorate the search performance especially for high-dimensional data spaces [7]. The region description of an MBR comprises for each dimension a lower and an upper bound. Thus, $2d$ floating point values are required.

The $R^\star$ [5] is an extension of the R-tree based on a careful study of R-tree algorithms under various data distributions. Beckmann et al. [5] identified several weaknesses of the original algorithms. In particular, they confirmed that the insertion phase is critical for good search performance. The design of the $R^\star$ therefore introduces a policy called $forced\ reinsert$: If a node overflows, it is not split right away. Rather, p entries are removed from the node and reinserted into the tree. The parameter p may vary; Beckmann et al. suggest it should be about 30% of the maximal number of entries per page.

Another issue investigated by Beckmann et al. concerns the node-splitting policy. Although Guttman's R-tree algorithms tried only to minimize the area covered by the bucket regions, the $R^\star$-tree algorithms also take the following optimization objectives into account.

- minimize overlap between page regions,
- minimize the surface of page regions,
- minimize the volume covered by internal nodes, and
- maximize the storage utilization.

In summary, the $R^\star$-tree differs from the R-tree mainly in the insertion algorithm; deletion and searching are essentially unchanged. Beckmann et al.

report performance improvements up to 50% compared to the basic R-tree. Their implementation also shows that reinsertion may improve storage utilization. In broader comparisons, however, Hoel and Samet [47] and *Günther* and Gaede [40] found that the CPU time overhead of reinsertion can be substantial, especially for large page sizes. The $R^\star$-tree and R-tree can ensure the minimum storage utilization, because they require no forced split.

5.2.2 SS-tree

The SS-tree [99] is an index structure designed for similarity indexing of multidimensional data point. It is an improvement of the $R^\star$-tree and enhances the performance of nearest neighbor queries by modifying the following respects.

Firstly, it employs bounding spheres rather than bounding rectangles for the region shape. The center of a sphere is the centroid of underlying points and the SS-tree permits to divide points into isotropic neighborhoods by utilizing centroids in the tree construction algorithms, i.e., the insertion algorithm and the split algorithm. Another advantage of using bounding spheres for the region shape is that it only requires nearly half storage compared to bounding rectangles. Since a sphere is determined by the center and the radius, it can be represented with as many parameters as the dimensionality plus one. Moreover, spheres are theoretically superior to volume equivalent MBRs because the Minkowski sum is smaller. MBRs have in general a smaller volume, whereas the advantage in the Minkowski sum is more than compensated. So the SS-tree outperforms the $R^\star$-tree.

Secondly, the SS-tree modifies the forced reinsertion mechanism of the $R^\star$-tree. When a node or a leaf is full, the $R^\star$-tree reinserts a portion of its entries rather than splits it, unless reinsertion has been made on the same tree level. However, the SS-tree reinserts entries unless reinsertion has been made at the same node or leaf. This promotes the dynamic reorganization of the tree

structure [54].

5.2.3 SR-tree

The SR-tree [54] can be regarded as the combination of the $R^\star$-tree and the SS-tree. It uses the intersection solid between a rectangle and a sphere as the page region. The rectangular part is, as in R-tree variants, the minimum bounding rectangle of all points stored in the corresponding subtree. The spherical part is, as in the SS-tree, the minimum sphere around the centroid point of the stored objects. Regions of SR-trees have the most complex description among all index structures presented in this section: they comprise $2d$ floating point value for the MBR and $d + 1$ floating point values for the sphere.

The motivation for using a combination of sphere and rectangle, presented by Katayama et al. [54] is that according to an analysis presented in White and Jain [99], spheres are basically better suited for processing nearest-neighbor and range queries using the L_2 metric. On the other hand, spheres are difficult to maintain and tend to produce much overlap in splitting. Katayama et al. believe therefore that a combination of R-tree and SS-tree will overcome both disadvantages.

The reported performance results, compared to the SS-tree and the $R^\star$-tree, suggest that the SR-tree outperforms both index structures [10].

In Summary, Table 5.3 shows the index structures described above and their most important properties. The first column contains the name of the index structure, the second shows which geometrical region is represented by a page. The last columns describe the used algorithms: what strategy is used to insert new data items (column 3), what criteria are used to determine the division of objects into sub-partitions in case of an overflow (column 4), and if the insert algorithm uses the concept of forced reinserts (column 5) [10].

Here, provide a comparison among the indexes listing only properties not

Table 5.3: High-Dimensional Index Structures and Properties

Name	Region	Criteria for Insert	Criteria for Split	Reinsert
R-tree	MBR	Volume enlargement volume	(Various algorithm)	No
$R^\star$-tree	MBR	Overlap enlargement Volume enlargement Volume	Surface area Overlap Dead space coverage	Yes
SS-tree	Sphere	Proximity to centroid	Variance	Yes
SR-tree	Intersect. sphere and MBR	Proximity to centroid	Variance	Yes

Table 5.4: Qualitative Comparison High-Dimensional Index Structures

Name	Problems in High-D	Supported Query Types	Locality of Node Splits	Storage Utilization
R-tree	Poor split algorithm leads to deteriorated directories	NN, Region, range	Yes	Poor
$R^\star$-tree	Dto.	NN, Region, range	Yes	Medium
SS-tree	High overlap in directory	NN	Yes	Medium
SR-tree	Very large directory size	NN	Yes	Medium

trying to say anything about the "overall" performance of a single index. In fact, most probably, there is no overall performance; rather, one index will outperform other indexes in a special situation whereas this index is quite useless for other configurations of the database. Table 5.4 shows such a comparison. The first column lists the name of the index, the second column explains the biggest problem of this index when the dimension increases. The third column lists the supported types of queries. In the fourth column, we show if a split in the directory causes "forced splits" on lower levels of the directory. The fifth column shows the storage utilization of the index, which is only a statistical value depending on the type of data and, sometimes, even on the order of insertion [10].

5.2.4 β-neighbor Algorithm Based on SR-tree Structure

Table 5.5 gives out all of the necessary notations used in the algorithms.

Table 5.5: Notations in the Algorithms

P	Point Set (Dataset)
C_i	class of P_i
E	the edited set
N_i	Potential β-neighbor of Point i
NP_i	β-neighbor of Point i
M	vector to store the pairwise information
V	vector to record the visited/non-visited information
attribute	flag to indicate non-leaf node/leaf-node
R	$\frac{\beta}{2}$dist(P_i, P_r)
$dist(P_i, P_r)$	the Euclidean distance between points P_i and P_r
$J(P_i, P_r)$	the interior region of the two circles as described before
rangesearch (SR, Q_1, Q_2, R)	nearest point returned by range search in tree SR with query Q_1, Q_2 and range R
$H\text{-}prune_i$	vector to store the pre-pruned subtree/neighbor information

With the SR-tree for indexing, the β-neighbor edited algorithm can be modified in the following fields. For each pair of points (P_i, P_r), according to the edited algorithm, two searching requests, S_1 and S_2, will be put forward. S_1 is the range search with the query point Q_1, which equals to $(1-\frac{\beta}{2})\times P_i+\frac{\beta}{2}\times P_r$. The query point of S_2 is Q_2 and it equals to $(1-\frac{\beta}{2})\times P_r+\frac{\beta}{2}\times P_i$. The range for both of the searches, S_1, S_2, is $\frac{\beta}{2}dist(P_i, P_r)$. Then we will traverse the tree built before once for both range searches and obtain the nearest neighbor, P_k, satisfying the conditions from the two range searches. If P_k is not in the joint section of the two range searches, (P_i, P_r) will be a pair in the result of β-neighbor edited set. The details of the algorithm can be found in Algorithm 2. The time complexity is $O(dn^2logn)$.

5.2.5 Pruning Search Space for β-neighbor Algorithm

Fist of all, we will introduce the basic concepts of the hyperplane and some properties of the β-neighbor pairs. Let P_i be the data point of the set whose β-neighbors' set is our interesting in computing. Consider a Point, $P_r \in P$.

Algorithm 2: $\beta-$neighbor Edited Algorithm Based on SR-tree

Data : $P = \{P_1, P_2, \ldots, P_n\}$.
Result : E = the edited set.
begin
Let E be {}
Build a SR-tree SR, to store P
foreach P_i **do**
Use a Vector M to store the pairwise information
end
for $\forall P_i \in P$ **do**
traverse the tree from the left-most leaf node to the right-most leaf node
1 **foreach** P_r **do**
Initialize $M[r]$
if *P_i and P_r are not in the same class, and P_r is on the right side of P_i* **then**
$M[r]$ = CANDIDATE
else $M[r]$ = NOT-CANDIDATE
end
2 Initialize $V[T]$ = NOT-VISITED, where T represents the subtree of SR.
3 Traverse SR in post-order
for *T being visited and $V[T]$ = NOT-VISITED* **do**
if *T is a leaf node* **then**
for *$\forall P_i$ in T and $M[r]$ = CANDIDATE* **do**
consider $J(P_i, P_r)$
Let R be $\frac{\beta}{2}$dist(P_i, P_r)
Let Q_1 and Q_2 equal to $(1-\frac{\beta}{2})P_i+\frac{\beta}{2}P_r$, $(1-\frac{\beta}{2})P_r+\frac{\beta}{2}P_i$ respectively.
Let P_k=rangesearch(SR, Q_1, Q_2, R)
if *dist$(P_k, Q_1) \leq R$ and dist$(P_k, Q_2) \leq R$* **then**
$M[r]$ = NOT-CANDIDATE.
else $M[r]$ = SELECTED;
$E = E + \{P_i, P_r\}$
end
end
if *T is a internal node* **then**
foreach *subtree of T from the left-most to the right-most subtree* **do**
Repeat step **3**.
V[T]=VISITED.
end
end
end
end
end

Draw out the intersection region of the two circles and the hyperplane i and hyperplane r which are perpendicular to the line segment (P_i, P_r). From the Fig. 5.3, we have the following definitions and lemmas.

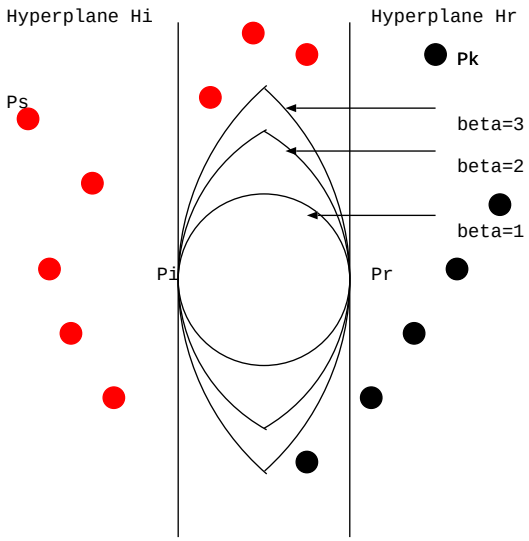

Figure 5.3: Testing $\beta-$neighbor Pairs and Plane Pruned

Definition 2 Pruned β-neighbor

Points in the dataset behind the hyperplane P_r are the pruned β-neighbor of point i.

Hence the similar definition of **β-neighbor** is easily obtained as follows:

Definition 3 β-neighbor

In the Euclidean space, point P_i and P_r are β-neighbors if there is no point $P_k \in P \backslash \{P_i, P_r\}$, which makes $\angle P_i P_k P_r > \frac{\pi}{2}$.

Secondly, it is obvious that if by some means we can reduce the number of pairs to be tested for β-neighbor, the algorithm will be more efficient. So we define the pairs of points, which can be pruned in advance. Given two points P_i and P_r, which are a distinct β-neighbor pair, we construct $S(P_i, P_r)$. Then

we construct a hyper-plane H_i containing the tangent to the sphere touching P_i, and another one H_r touching P_r. Let point P_k be on the **negative side** of H_i, i.e. the side that is opposite to the normal of H_i, as shown in Fig. 5.4. From definition 3, we can infer that the angle $\angle P_r P_i P_k$ must be less than 90°. Therefore, P_k must not be a (distinct) β- neighbor of P_i. Similarly, P_s is not a (distinct) β-neighbor of P_r. Using such property, we can prune the searching space for each member in the distinct β-neighbor pair by half.

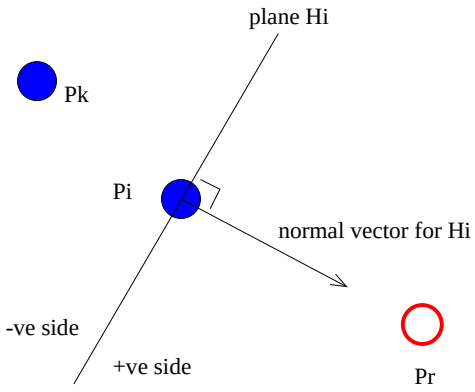

Figure 5.4: Negative and Positive Side of a Hyperplane

Thirdly, in the SR-tree indexing structure, the basic units are the bounding sphere and bounding rectangle. As known before, spheres are basically better suited for processing nearest-neighbor and range queries using the L_2 metric. Moreover, from Table 5.3 the main criterion for node insertion and split is the bounding sphere. Furthermore, from the viewpoint of calculation, the hyper-sphere is the better choice, since the number of the parameters of hyper-sphere is almost half of that of the hyper-rectangle, used to record the necessary information. Then with SR-tree indexing structure, we can prune the search space as the following description.

Lemma 4 Consider a hyper-sphere B_s with radius R and center C, we can

test if B_s is completely on the negative side of hyper-plane H_i by:

$$\vec{n} \cdot (\vec{p_i} - \vec{c}) \leq -R$$

where $\vec{n}$ is the normal vector of H_i, $\vec{p_i}$ is the spatial vector of P_i, and $\vec{c}$ is the spatial vector of C. If the above inequality holds, then B_s is completely on the negative side of H_i. Using such property, any points inside B_s can be pruned when we search (distinct) β-neighbors for P_r. Similarly, we can do such pruning for P_i.

Definition 5 Hyper plane

Hyper-plane (P_i, P_r) : Hyper plane that contains the tangent touching $S(P_i, P_r)$ at P_i, P_r (See Fig. 5.3).

Lemma 6 Any point inside a hyper-sphere B_s with radius R and center C is not a (distinct) β-neighbor for point P_r if B_s is on the negative side of hyper-plane (P_i, P_r), and B_s does not intersect hyper-plane (P_i, P_r), where P_i and P_r are a (distinct) β-neighbor pair.

According to the definitions and lemmas above, we make use of SR-tree for this pruning since for each node of a SR-tree, there is a bounding sphere. If the bounding sphere of a subtree is completely on the negative side of a hyper-plane during the searching of β-neighbors for a point, then we can prune the whole subtree. Note that it is faster to use hyper-sphere for the negative side testing than using hyper-rectangle especially when the dimension is high because, for the later, we need to test every vertex of the hyper-rectangle which results in many distance computation.

The following paragraphs present our proposed algorithm in more details.

Algo. 3 shows the proposed algorithm using the SR-tree structure. First, we build a SR-tree for the given dataset as shown in step (1). Second, we traverse the tree to visit each point in **post-order**, i.e. from top to bottom

Algorithm 3: $\beta-$neighbor Edited Algorithm Based on SR-tree with Plane Pruning

Data : $P = \{P_1, P_2, \ldots, P_n\}$.
Result : $E =$ the edited set.
begin
  Let E be {}
1 Build a SR-tree SR, to store P
  foreach P_i **do**
    Use a Vector M to store the pairwise information
  end
  foreach P_i **do**
    Use a Vector $H\text{-}prune_i$ to store the pruned node information
  end
  for $\forall P_i \in P$ **do**
    traverse the tree from the left-most to the right-most leaf node
2   **foreach** P_r **do**
      Initialize $M[r]$
      if *P_i and P_r are not in the same class, and P_r is on the right side of P_i* **then** $M[r]$ = CANDIDATE
      else $M[r]$ = NOT-CANDIDATE
    end
3   Initialize $V[T]$ = NOT-VISITED, where T represents the subtree of SR-tree with root T.
4   $V[T]$ = PRUNED if T is completely on negative side of any $H \in H - prune_i$.
5   ***Find-β-neighbor***(SR, SR, M, V, E, P_i, $H\text{-}prune_1$, ...,$H\text{-}prune_n$)
  end
end

Function `Find-`β`-neighbor`(*SR, T, M, V, E,* P_i, *H-prune*$_1$, ..., *H-prune*$_n$)

for *T being visited and* $V[T]$ = *NOT-VISITED* **do**
  Traverse T in post-order
  if *T is a leaf node* **then**
    for $\forall P_i$ *in T and* $M[r]$ = *CANDIDATE* **do**
      consider $J(P_i, P_r)$
      Let R be $\frac{\beta}{2}$dist(P_i, P_r)
      Let Q_1 and Q_2 equal to $(1 - \frac{\beta}{2})P_i + \frac{\beta}{2}P_r$, $(1 - \frac{\beta}{2})P_r + \frac{\beta}{2}P_i$ respectively.
      Let P_k=rangesearch(SR, Q_1, Q_2, R)
      if *dist*$(P_k, Q_1) \leq R$ *and dist*$(P_k, Q_2) \leq R$ **then**
      $M[r]$ = NOT-CANDIDATE.
      else $M[r]$ = SELECTED;
      $E = E + \{P_i, P_r\}$
      Let H_r = hyper-plane(P_r, P_i)
      Visit SR and prune subtree T' that is on the negative side of H_r
      Set $V[T']$= PRUNED,
      Let H_i = hyper-plane(P_i, P_r)
      H-prune$_r$ = *H-prune*$_r$ + H_i.
    end
  end
  if *T is an internal node* **then**
    foreach *subtree of T from the left-most to the right-most subtree* **do**
      Find-*β*-neighbor(SR, T', M, V, E, P_i, *H-prune*$_1$, ..., *H-prune*$_n$).
    end
  end
  V[T]=VISITED.
end

and from left to right. For each point P_i in the leaf node N, we consider any other point P_j that is on the right hand side of P_i as shown in step (2). (P_r is either in N or a leaf node on the right hand side of N.) Based on symmetry, we do not need to visit point P'_r that is on the left hand side of P_i. This is because the pair (P'_r, P_i) has already been tested (β-neighbor pair testing) during the searching of β-neighbors for P'_r, and the effect of testing (P'_r, P_i) is the same as that of testing (P_i, P'_r). Note that this left-to-right searching order is based on the fact that points in the same node are usually closer to each other than points in different nodes, and therefore points in the same node are more likely β-neighbors of each other. With this ordering, we can construct hyper-planes and perform pruning earlier. Another advantage of this ordering is the reduction of memory swapping because points in the same node are usually inclined to be loaded to the same memory page simultaneously. Experiments in [22] show that using such ordering can reduce 10% in searching time when compared to random ordering.

If P_i and P_r are of different classes, we test whether there is any other point inside $J(P_i, P_r)$ by performing the nearest neighbor search, which refers to step (5) of the $find\text{-}\beta\text{-}neighbor()$ function in Algo. 3. If no such point exists, then P_i and P_r are a distinct β-neighbor pair. We add them to the resulting edited set.

Once we find out such neighbor for P_i, we construct pruning planes H_i for P_i and H_r for P_r. Then we visit the tree for P_i, and prune all the subtrees with bounding spheres completely on the negative side of H_r. For the remaining unpruned subtrees, we test for the "not yet visited" points under these subtrees in the left-to-right order again. Once we find out another neighbor for P_i, we repeat the plane construction and pruning step. For P_r, we add H_i to the hyper-plane list for pre-pruning, $H\text{-}prune_r$, so that we can use planes in $H\text{-}prune_r$ for pruning when finding the β-neighbors of P_r as shown in step (4).

Table 5.6: Notation and Description Used in this Section

Gab-srtree	the Gabriel editing algorithm based on the SR-tree
Rel-srtree	the Relative editing algorithm based on the SR-tree
Gab-sr-prune	the Gabriel editing algorithm based on the SR-tree with pruning
Rel-sr-prune	the Relative editing algorithm based on the SR-tree with pruning

5.2.6 Comparing Results of Non-index Methods with Those of Methods with Indexing

In this section, we describe the application of our proposed algorithms 2 and Algo. 3 to the synthetic and real datasets. These two algorithms are coded by Ling [22] and me. First, we extract the edited set with heuristic algorithm 1 described in Chapter 3. Then another algorithm, the β-neighbor edited algorithm based on the SR-tree, is also used to obtain the edited set. Finally, after pruning the searchings space, the edited algorithm with SR-tree indexing structure is applied to extract the points from the training dataset to the resulting edited set. The edited sets obtained in the three methods are the same, so we only consider the performance of the methods from the viewpoint of efficiency. Table 5.6 shows the notations and descriptions used in the experiments.

From table 5.7 5.8 it is easy to see that:

1. Our SR-tree methods are much faster than the regular editing algorithm in general, especially for the Gabriel editing procedure.

2. Furthermore, with the aid of pruning, we can further speed up the whole procedure of finding Gabriel edited set. When we regard the time cost by Heuristic Gabriel editing method as the standard value, the time taken by the other two methods are normalized based on the standard value.

Table 5.7: Time to Find Distinct Gabriel Edited Sets (in second)

data	Gab-heurist	Gab-srtree	Gab-sr-prune
f2	25	17	6.19
f6	40	19	9.17
10k.15d.2c	9137	6712	6776
10k.15d.4c	857547	10914	10894
10k.15d.8c	1250286	24205	23888
uniform.100	0.2	0.09	0.06
uniform.1000	26	16	7.12
uniform.10000	3088	3264	2647
segment	803	547	480
german	289	49	41
sat	40703	4158	3374
shuttle	-	104062	100592
glass	0.73	0.99	1.8
wine	1.12	0.51	0.68

From Fig. 5.5 we can find SR-tree method can save the execution time at least 30%. For the Relative editing procedure, the pruning step does not reduce the execution time too much, since the relative editing reduces the candidate points quickly enough.

3. For dataset $10k.15d.4c$, $10k.15d.8c$, $german$, and $satimage$ applying SR-tree indexing to the β-neighbor edited algorithm can even save the editing time more than 60%. Especially for the $shuttle$ dataset, the heuristic β-neighbor edited algorithm is inefficient to handle the large dataset and using SR-tree indexing, we obtain the satisfiable result.

4. Nevertheless, SR-tree indexing structure does not perform well for data set $glass$ and $uniform.10000$ where the cost time to find β-neighbor edited set based on SR-tree is more than that of the heuristic algorithm.

5. For the dataset $10k.15d.2c$, we can find that the cost time with the heuristic algorithm is the least, since comparing with the original size of dataset, the size of Relative edited set is so small that during the

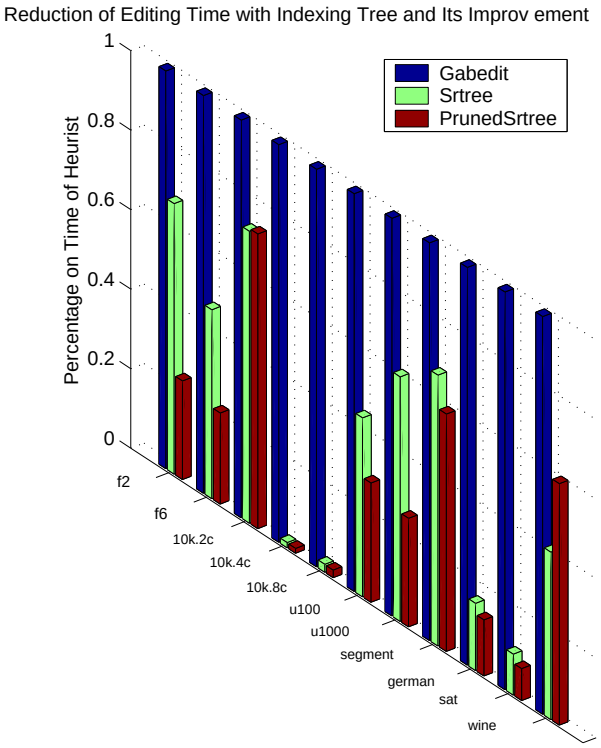

Figure 5.5: Reduction of Editing Time with Indexing Tree and Its Improvement

searching procedure, the speed of discarding the candidate points is quick enough. Building the SR-tree and doing the pruning operation can not reduce the execution time but expend more time.

6. Moreover, the experimental results show that pruning steps may effect on most of the datasets but not all of them. The exceptions are its use on the *glass* in β-editing procedures and *wine* datasets in the Gabriel editing procedure. The values in Table 5.7 and Table 5.7 present that using SR-tree indexing structure and pruning step degenerates much the performance of the algorithm on *glass* dataset.

7. In general cases, the method based on SR-tree and using plane-pruning can reduce the running time up to 60% compared to that of using SR-tree indexing only.

Table 5.8: Time to Find Distinct Relative Neighbor Edited Sets (in second)

dataset	Rel-heurist	Rel-srtree	Rel-sr-prune
f2	22.24	19.38	17.9
f6	43.18	26.4	23.3
10k.15d.2c	3233	6159	6142
10k.15d.4c	54341	9144	9160
10k.15d.8c	446719	11438	11397
uniform.100.4d	0.13	0.11	0.08
uniform.1000.4d	19.59	19.08	18.77
uniform.10000.4d	2464	4280	3998
segment	537	565	548
german	141.5	56.5	53
sat	14240	4793	4731
shuttle	-	254243	252968
glass	0.71	1.33	1.36
wine	1.03	0.61	0.62

8. If the dataset is large such as $uniform$.10000, SVM (8213 seconds) is much slower than the Gabriel editing (2647 seconds) and the Relative editing (2464 seconds). The training and testing time for edited sets are much less than the time for the corresponding whole datasets. This time reduction is significant for real time processing.

9. Note that for *shuttle*, although the training time of SVM with the original set is fast, we need to have a good SVM parameter setting, otherwise there are some cases where the training time exceeds 100,000 seconds.

5.3 Combination of Parallelism and SR-tree Indexing Structure

Based on the SR-tree structure, suppose $P = \{P_1, P_2, P_3, P_4\}$, and the left-to-right order for the points stored in the SR-tree is $P_2 - > P_1 - > P_4 - > P_3 - > P_5$. In our algorithm, the visiting order is $(a)P_2$, verifying P_1, P_4, P_3, P_5; $(b)P_1$, verifying P_4, P_3, P_5; $(c)P_4$, verifying P_3, P_5; $(d)P_3$,verifying P_5. This

visiting sequence can be executed simultaneously by using the multi-thread processing [22].

We can see that the nodes on the left will test more potential neighbors than the nodes on the right, so instead of randomly allotting the points to the processor, we adopt the left-to-right order to visit the points. Furthermore, if a processor is idle, we will assign the most left and unvisited point to the processor, which can keep that every processor has the similarly equal loading. For example, if we have only two processors, points 2, 1 will be visited first. If Processor 2 is idle, point 4, not point 3, will be visited.

5.3.1 Comparing Results of Both Techniques Applied

Table 5.9: Time for Finding Gabriel Edited Set Using SR-tree (with pruning) for Different Degree of Parallelization (in second)

Problem	Number of threads			
	1	3	6	8
f2	6.19	8.48	8.1	10.69
f6	9.17	11.7	8.5	11.88
10k.15d.2c	6776	2882	1983	2310
10k.15d.4c	10894	9374	3967	3649
10k.15d.8c	23888	19343	19071	19515
uniform.100	0.06	0.08	0.07	0.08
uniform.1000	7.12	15.61	8.33	10.4
uniform.10000	2647	2047	1421	1321
segment	480	363	173	290
german	41	87.33	14.48	24.85
sat	3374	3425	1305	2079
shuttle	100592	58649	39095	29320
glass	1.8	1.22	0.5	0.76
wine	0.68	0.35	0.25	0.33

In the experiments, 1,3,6,8 are chosen as the several trials of the number of threads to watch the effect of the amount of threads on the performance.

We scale the editing time in Table 5.9 into a suitable interval and according to the values' interval, illustrate them into two figures and present the scaling

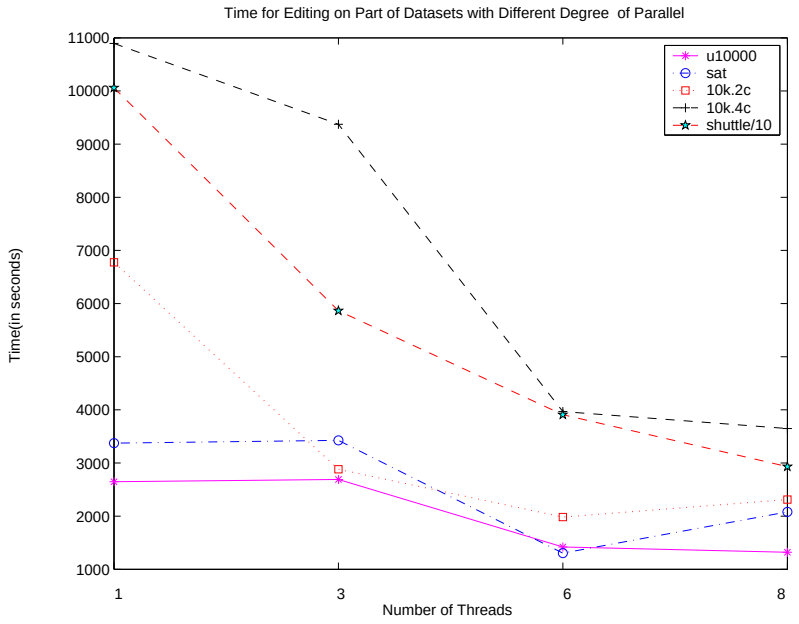

Figure 5.6: Time for Editing with SR-tree on Part of Datasets in Different Degree of Parallel

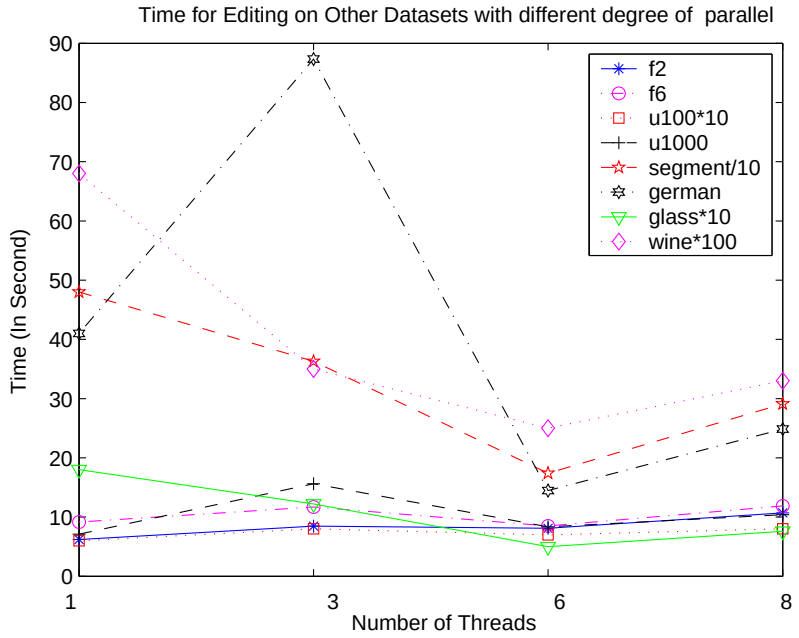

Figure 5.7: Time for Editing with SR-tree on Remaining Part of Datasets in Different Degree of Parallel

Table 5.10: Time for Finding Relative Neighbor Edited Set Using SR-tree (with pruning) for Different Degree of Parallelization (in second)

Problem	Number of threads			
	1	3	6	8
f2	17.9	11.89	9.92	10.7
f6	23.3	16.54	11.92	12
10k.15d.2c	6142	2833	1887	1635
10k.15d.4c	9160	4055	2740	2455
10k.15d.8c	11397	5350	3392	3052
uniform.100.4d	0.08	0.09	0.14	0.12
uniform.1000.4d	18.77	11.9	11	12.1
uniform.10000.4d	3998	2591	1946	1882
segment	548	249	140.9	123.7
german	53	30.5	18.65	16.2
sat	4731	1991	1050	856
shuttle	252968	131298	84234	78007
glass	1.36	0.66	0.48	0.43
wine	0.62	0.35	0.3	0.26

coefficient in the Fig. 5.6 and Fig. 5.7.

1. When the number of threads increases to 3, the running time of the Gabriel editing is not reduced; on the contrary, the execution time increases for most of the datasets except *segment* and *wine* data and the large datasets with the size larger than 10,000.

2. For most datasets, using 6 threads achieves the quickest Gabriel editing time of all the trials. Along with the amount of threads allocated increasing to 8, the running time of $10k.15d.4c$, $uniform1000$, and *shuttle* datasets continues decreasing (See Fig. 5.6).

3. In the Relative editing procedure, the execution time of the datasets is reduced along with the increasing threads, except for the datasets, $f2$, $f6$, $uniform1000$ and $uniform100$. Nearly all of these datasets gain the least editing time when the number of the thread is equal to 6.

4. Comparing with the training time of SVM (See Table 4.10), even with

parallel technique used to find β-neighbor edited set, the costing time can not be reduced too much in common cases.

5. Nevertheless for large datasets, such as $uniform10000$, *shuttle*, the β-neighbor edited algorithm with parallelism outperform the performance of the standard SVM on efficiency.

So before we assign the threads to the program, we need consider the scale of the problem, watch the load of CPU and choose a suitable number of threads so as to avoid overloading of the system. In sum, for the large datasets our β-neighbor edited algorithm with parallelism can preserve the high accuracy and reduce the training time of SVM.

5.4 Summary

Along with the techniques used, the heuristic β-neighbor edited algorithm, the parallel heuristic β-neighbor edited algorithm, the β-neighbor edited algorithm based on SR-tree index structure, and the parallel β-neighbor edited algorithm based on SR-tree index structure are proposed to edit the original dataset (See Table 1.1). After that, we use do SVM training on the edited set, compare the sum of the editing time and the training time with the original SVM training time and then have some observations as follows from the experiments conducted before.

First, the heuristic method is not suitable to deal with the huge datasets, since the matrix is used to store the β-neighbor pair information and the necessarily allocated space for the matrix may overflow. For the common-size datasets, the heuristic method can not outperform SVM training on efficiency.

Secondly, we apply parallel technique in the program to speed up the editing procedure, which can separate the whole editing procedure into several independent parts. However, the results show that the parallelism used in the

heuristic algorithm does not improve much.

Thirdly, the result in the second trial makes us think about other ways to improve the performance of the editing algorithm. The spatial indexing structure is chosen as our testing technique employed in the editing algorithm. The experimental result verify our expectation. Even though the SR-tree indexing reduces the editing time much, compared to the SVM training, the editing algorithm based on SR-tree indexing structure does not behave better on the common-size dataset. Only when applied to the large dataset, the editing algorithm does well.

Finally, we combine the SR-tree indexing structure and parallel computing technique together. The performance of the editing algorithm on large datasets is improved much and the editing time becomes more comparable to the SVM training time. So we can edit the huge dataset with the parallel β-neighbor edited algorithm, which is also based on the SR-tree indexing structure, before we do SVM training. Then this editing step can speed up the original training procedure of SVM.

Chapter 6

Conclusion

Support Vector Machine is a new and promising approach to pattern classification. It promises to give good generalization and has been applied to various tasks.

The formulation of SVM is elegant in that it is simplified to a convex quadratic programming (QP) problem. But the number of iterations required to solve a QP problem is not in control. Along with the size of dataset increasing, the difficulty of the convergence of SVM increases. Hence, when the number of training data points exceeds a few hundred, the computation cost for the SVM training will be not satisfiable. There are two kinds of methods to clear the obstacle. One is to improve the algorithm from the interior, such as RSVM. The other is to preprocess the dataset and discard the unimportant points, which is the goal of our proposed method. The relationship among SVM, Convex hull, and Voronoi diagram inspires us to employ computational geometric algorithms to locate plausible support vectors and reduce the size of the training data. In particular, we present the β-neighbor edited algorithm.

With the obtained β-neighbor edited set as the training set, we can speed up the convergence of SVM, while preserves the high performance. When we compare the training time of SVM to the combination of the editing time and the training time of SVM with the edited set as input data, we find that the latter is not less than the former one. So we use some other effective

technologies, such as parallel and SR-tree indexing, to improve the efficiency of the edited algorithm. Experiments indicate that when the size of the training data is large enough, such as larger than 10,000, the performance of the editing algorithm and the training of SVM using the edited set as input is acceptable.

We also test the performances of the other two different classifiers from SVM, which are kNN and $C4.5$. Generally with the edited set (no matter Gab-edited set or Rel-edited set) as the training set, the classifiers can preserve high accuracy with edited set. Furthermore, compared to the performance of RSVM, we find that the accuracy results from SVM with edited set are similar to those of RSVM.

Bibliography

[1] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Database mining: A performance perspective. In Nick Cercone and Mas Tsuchiya, editors, *Special Issue on Learning and Discovery in Knowledge-Based Databases*, number 5(6), pages 914–925. Institute of Electrical and Electronics Engineers, Washington, U.S.A., 1993.

[2] F. Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys, ACM*, 23(3):345–405, 1991.

[3] F. Aurenhammer and R. Klein. Voronoi diagrams. In J. Sack and G. Urrutia, editors, *Handbook of Computational Geometry*, chapter V, pages 201–290. Elsevier Science Publishing, 2000. [SFB Report F003-092, TU Graz, Austria, 1996].

[4] C. Bradford Barber, David P. Dobkin, and Hannu Huhanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, 1996.

[5] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 322–331, May 1990.

[6] Kristin P. Bennett and Erin J. Bredensteiner. Duality and geometry in SVM classifiers. In *Proc. 17th International Conf. on Machine Learning*, pages 57–64. Morgan Kaufmann, San Francisco, CA, 2000.

[7] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree: An index structure for high-dimensional data. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *Proceedings of the 22nd International Conference on Very Large Databases*, pages 28–39, San Francisco, U.S.A., 1996. Morgan Kaufmann Publishers.

[8] M. De Berg, M. Van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin, 1997.

[9] Binay K. Bhattacharya, Ronald S. Poulsen, and Godfried T. Toussaint. *Application of Proximity Graphs to Editing Nearest Neighbor Decision Rule.* International Symposium on Information Theory, Santa Monica, 1981.

[10] Christian Böhm, Stefan Berchtold, and Daniel A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys (CSUR)*, 33(3):322–373, 2001.

[11] Jean-Daniel Boissonnat and Mariette Yvinec. *Algorithmic Geometry.* Cambridge University Press, Cambridge, U.K., 1998.

[12] Prosenjit Bose, Luc Devroye, William S. Evans, and David G. Kirkpatrick. On the spanning ratio of gabriel graphs and beta-skeletons. In *Proc. of the 5th Latin American Symposium on Theoretical Informatics (LATIN'02)*, pages 479–493, Cancun, Mexico, 2002.

[13] B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, 1992.

[14] Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learing Theory*, pages 144–152, Pittsburgh, PA, July 1992. ACM Press.

[15] P. S. Bradley and O. L. Mangasarian. Massive data discrimination via linear support vector machines. *Optimization Methods and Software*, 13(1):1–10, 2000.

[16] P. S. Bradley, O. L. Mangasarian, and W. N. Street. Feature selection via mathematical programming. *INFORMS Journal on Computing*, 10:209–217, 1998.

[17] Erin J. Bredensteiner and Kristin P. Bennett. Multicategory classification by support vector machines. *Computational Optimizations and Applications*, 12:53–79, 1999.

[18] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth Inc., Belmont,CA, 1984.

[19] D.S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[20] J. Christopher Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[21] D. Butenhof. *Programming with Posix Threads*. Addison-Wesley, 1997.

[22] Yin-Ling Cheung, Ada Wai chee Fu, Irwin King, and Wan Zhang. An sr-tree approach to editing nearest neighbor decision rules. *To be submitted*, 2002.

[23] Chang Chih-Chung and Lin Chih-Jen. *LIBSVM: a Library for Support Vector Machines.* 2001. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[24] P. Chou. Optimal partitioning for classification and regression trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):340–354, 1991.

[25] C. Cortes and V. Vapnik. Support-vector network. *Machine Learning*, 20:273–297, 1995.

[26] B.V. Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, Las Alamitos, California, 1991.

[27] B.N. Delaunay. Sur la sphere vide. *Bull. Acad. Science USSR VII: Class. Sci. Math.*, 7:793–800, 1934.

[28] V. Susheela Devi and M. Narasimha Murty. An incremental prototype set building technique. *Pattern Recognition*, 35:505–513, 2002.

[29] H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. *Disc. and Comp. Geom.*, 8(1):25–44, 1986.

[30] Floriana Esposito, Donato Malerba, and Giovanni Semeraro. A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491, 1997.

[31] R.A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.

[32] E. Fix and J. L. Hodges. Discriminatory analysis: Nonparametric discrimination: consistency properties. Technical Report 4, USAF School of Aviation Medicine, Randolf Field, Texas, 1951.

[33] J. Friedman. Another approach to polychotomous classification. Technical report, 1996.

[34] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2-3):131–163, 1997.

[35] T.T. Friess. Support vector neural networks: The kernel adatron with bias and soft margin. Tech report, Dept of Automatic Control and Systems Engineering, University of Sheffield, Sheffield, England, 1998.

[36] T.T. Friess, N. Cristianini, and I. C. G. Campbell. The kernel-adatron: A fast and simple learning procedure for support vector machines. In J. Shavlik, editor, *Proceedings of the 15th International Conference on Machine Learning(ICML'98)*, pages 188–196, San Fancisco, California, 1998. Morgan Kaufmann.

[37] K.R. Gabriel and R.R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18:259–278, 1969.

[38] W. Gates. The reduced nearest neighbor rule. *IEEE Transatctions on Information Theory*, 18:431–433, 1972.

[39] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.

[40] Oliver Gnther and Volker Gaede. Oversize shelves: A storage management technique for large spatial data objects. *International Journal of Geographical Information Science*, 11(1):5–32, 1997.

[41] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 47–57, June 1984.

[42] Isabelle Guyon, Vladimir N. Vapnik, Bernhardt E. Boser, Léon Bottou, and Sara A. Solla. Structural risk minimization for character recognition. In *Advances in Neural Information Processing Systems*, volume 4, Denver, 1992. Morgan Kaufman.

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

[44] David J. Hand, Heikki Mannila, and Padhraic Smyth. *Principles of Data Mining.* The MIT Press, 2001.

[45] Peter E. Hart. The condensed nearest neighbor rule. *IEEE Transatctions on Information Theory*, 14:515–516, 1968.

[46] S. Haykin. *Neural networks: a comprehensive foundation.* Prentice-Hall, Upper Saddle River, NJ, 1994.

[47] Erik G. Hoel and Hanan Samet. A qualitative comparison study of data structures for large line segment databases. *ACM SIGMOD Record*, 21(2):205–214, 1992.

[48] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002.

[49] Chih-Wei Hsu and Chih-Jen Lin. A simple decomposition method for support vector machines. *Machine Learning*, 46:291–314, 2002. Implementation available in bsvm.

[50] Chih-Wei Hsu and Chih-Jen Lin. A simple decomposition method for support vector machines. *Machine Learning*, 46:291–314, 2002.

[51] Don Hush and Clint Scovel. Polynomial-time decomposition algorithms for support vector machines. Technical report, 2000.

[52] J.W. Jaromczyk and G.T. Toussaint. Relative neighborhood graphs and their relatives. *Proceedings IEEE*, 80(9):1502–1517, 1992.

[53] T. Joachims. Making large-scale support vector machine learning practical. In C. Burges B. Scholkopf and A. Smola, editors, *Advances in Kernel Methods: Support Vector Machines*, Cambridge, MA, December 1998. MIT Press.

[54] Norio Katayama and Shin'ichi Satoh. The SR-tree: an index structure for high-dimensional nearest neighbor queries. In Joan Peckham, editor, *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pages 369–380. ACM Press, 1997.

[55] L. Kaufman. Solving the quadratic programming problem arising in support vector classification. In C. J. C. Burges B. Schokopf and A. J. Smola, editors, *Advances in Kernel Methods Support Vector Learning*, number 14167. MIT Press, 1999.

[56] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. *A fast iterative nearest point algorithm for support vector machine classifier design.*, volume 11. 2000.

[57] D.G. Kirkpatrick and J. D. Radke. *A Framework for computational morphology.* In G. T. Toussaint, editor, Computational Geometry, North-Holland, Amsterdam, Netherlands, 1985.

[58] Stefan Knerr, Leon Personnaz, and Gerard Dreyfus. *Single-Layer learning revisited: a stepwise procedure for building and training a neural network.* In J. Fogelman, editor, Neurocomputing: Algorithms, Architecture and Applications., Springer-Verlag, 1990.

[59] J.H. Lee and Chih-Jen Lin. Automatic model selection for support vector machines. 2000.

[60] Yuh-Jye Lee and O. L. Mangasarian. Rsvm : Reduced support vector machines. In *Proceedings of the First SIAM International Conference on Data Mining*, 2001.

[61] Yuh-Jye Lee and O. L. Mangasarian. SSVM: A smooth support vector machine. *Computational Optimization and Applications*, 20:5–22, 2001.

[62] David D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, number 1398, pages 4–15, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE.

[63] Chih-Jen Lin. Linear convergence of a decomposition method for support vector machines. 2001.

[64] K.M. Lin and C.J. Lin. A study on reduced support vector machines. Technical report, Computer Science and Information Engineering, National Taiwan University, February, 2002.

[65] O. Mangasarian. Generalized support vector machines. *Proceedings of NIPS*98 Workshop on Large Margin Classifiers*, 1998. Technical Report 98-14, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin.

[66] O. L. Mangasarian. Mathematical programming in data mining. *Data Mining and Knowledge Discovery*, 1(2):183–201, 1997.

[67] O. L. Mangasarian and D. R. Musicant. Successive overrelaxation for support vector machines. *IEEE-NN*, 10(5):1032, September 1999.

[68] D.W. Matula and R.R. Sokal. Properties of gabriel graphs relevant to geographic variation research and the clustering of points in the plane. *Geographical Analysis*, 12(3):205–222, 1980.

[69] Donald Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, 1994.

[70] Douglas C. Montgomery and E.A. Peck. *Introduction to Linear Regression Analysis.* John Wiley and Sons, 2nd edition, 1992.

[71] Bradford Nichols, Dick Buttlar, and Jacqueline Proulx Farrell. *Pthreads Programming: A POSIX Standard for Better Multiprocessing.* O'Reilly and Associates, 1st edition, 1996.

[72] Atsuyuki Okabe, Barry Boots, and Kokichi Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams(Wiley Series in Probability and Mathematical Statistics).* John Wiley and Son Ltd, Chichester,West Sussex, England, 1992.

[73] Edgar Osuna, Robert Freund, and Federico Girosi. Support vector machines: Training and applications. Technical Report AIM-1602, MIT AI LAB. CBCL, 1997.

[74] J. Park and I. W. Sandberg. Universal approximation using radial basis function networks. *Neural Computation*, 3(2):246–257, 1991.

[75] Y.T. Park. A comparison of neural net classifiers and linear tree classifiers - their similarities and differences. *Pattern Recognition*, 27(11):1493–1503, 1994.

[76] J. Platt. Fast training of support vector machines using sequential minimal optimization. In A. Smola B. Scholkopf, C. Burges, editor, *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, December 1998.

[77] Poggio and Girosi. Networks for approximation and learning. *Proc. IEEE*, 78(9):1481–1496, September 1990.

[78] M. J. D. Powell. The theory of radial basis function approximation in 1990. In W. A. Light, editor, *Advances in Numerical Analysis*, volume 2 of *Wavelets, Subdivision, and Radial Basis Functions*, pages 105–210. Clarendon Press, Oxford, 1992.

[79] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, 1985.

[80] J.R. Quinlan. C4.5 release 8.

[81] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[82] S. V. Rao. *Some Studies on Beta-Skeletons*. PhD thesis, Department of Computer Science and Engineering,Indian Institute of Technology Kanpur, India, August 1999.

[83] G.L. Ritter, H.B. Woodruff, S.R. Lowry, and T.L. Isenhour. An algorithm for a selective nearest neighbor decision rule. *IEEE Transatctions on Information Theory*, 21:665–669, 1975.

[84] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, et al., editors, *Parallel Distributed Processing: Volume 1: Foundations*, pages 318–362. MIT Press, Cambridge, 1986.

[85] Ishwar K. Sethi. Entropy nets: from decision trees to neural networks. *Proceedings of the IEEE*, 78(10):1605–1613, October 1990.

[86] Ishwar K. Sethi. Decision tree performance enhancement using an artificial neural network interpretation. *Artificial Neural Networks and Statistical Pattern recognition*, 1991.

[87] S. Theodoridis and K. Koutroumbas. *Pattern Recognition.* Academic Press, San Diego, CA, 1999.

[88] Godfried T. Toussaint. The relative neighborhood graph of a finite planar set. *Pattern Recognition*, 12(4):261–268, 1980.

[89] Godfried T. Toussaint. Proximity graphs for nearest neighbor decision rules: recent progress. *Interface-2002, 34th Symposium on Computing and Statistics (theme: Geoscience and Remote Sensing)*, April 17-20 2002.

[90] Godfried T. Toussaint, Binay K. Bhattacharya, and Ronald S. Poulsen. The application of voronoi diagrams to nonparametric decision rules. In *Computer Science and Statistics: The Interface*, pages 97–108, Atlanta, 1985.

[91] Godfried T. Toussaint and Ronald S. Poulsen. Some new algorithms and software implementation methods for pattern recognition research. In *Proc. IEEE International Computer Software Applications Conference*, pages 55–63, Chicago, 1979.

[92] R. J. Vanderbei. LOQO: An interior point code for quadratic programming. *Optimization Methods and Software*, 11:451–484, 1999.

[93] Vladimir N. Vapnik. *Estimation of dependencies based on empirical Data.* (in Russian), Nauka, Moscow, 1979.

[94] Vladimir N. Vapnik. *The nature of statistical learning theory.* Springer Verlag, Heidelberg, Germany, 1995.

[95] Vladimir N. Vapnik. *Statistical Learning Theory.* Wiley, New York, 1998.

[96] M. G. Voronoi. Nouvelles applications des parametres continus a la theorie des formes quadratiques. *J. Reine Angew, Math*, 134:198–287, 1908.

[97] J. Weston and C. Watkins. Multi-class support vector machines. In M. Verleysen, editor, *Proceedings of ESANN99*, Brussels, 1999. D. Facto Press.

[98] R. Clint Whaley, A. Petitet, and Jack J. Dongarra. Automatically tuned linear algebra software and the ATLAS project. Technical report, Department of Computer Sciences, University of Tennessee, 2000.

[99] David A. White and Ramesh Jain. Similarity indexing with the ss-tree. In *Proc. 12th IEEE International Conference on Data Engineering*, pages 516–523, New Orleans, Louisiana, February 1996.

[100] C. Xavier and S. S. Iyengar. *Introduction to Parallel Algorithms.* John Wiley and Sons, INC, New York, 1998.

[101] Ming-Hsuan Yang and Narendra Ahuja. A geometric approach to train support vector machines. In *Proceedings of the 2000 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2000)*, volume 1, pages 430–437, Hilton Head Island, June 2000.

[102] Wan Zhang, Yin-Ling Cheung, Irwin King, and Ada Wai chee Fu. Training of SVM speeded up by computational geometric methods. *To be submitted*, 2003.

[103] Wan Zhang and Irwin King. Locating support vectors via β-skeleton technique. In Lipo Wang, Jagath C. Rajapakse, Kunihiko Fukushima, Soo-Young Lee, and Xi Yao, editors, *Proceedings to the International Conference on Neural Information Processing (ICONIP2002)*, Orchid Country Club, Singapore, November 2002.

[104] Wan Zhang and Irwin King. A study of the relationship between support vector machine and gabriel graph. In *In Proceedings of IEEE World Congress on Computational Intelligence–International Joint Conference on Neural Networks*, Honolulu, Hawaii, May 2002.