UNIVERSITY of LIVERPOOL

PATTERN CLASSIFICATION USING ENHANCED MACHINE LEARNING

Thesis submitted in accordance with the
requirements of the University of Liverpool
for the degree of Doctor of Philosophy

in

Electrical Engineering and Electronics

by

Li MENG , B.Sc.(Eng.), M.Sc.(Eng.)

May 2002

PATTERN CLASSIFICATION
USING
ENHANCED MACHINE LEARNING

by

Li MENG

Dedicated to:

Li Fengmei, my mother
Meng Qingru, my father

Acknowledgements

I would like to thank, first of all, my supervisor Professor Q. H. Wu for his invaluable support and intellectual guidance, both academically and personally, during my Ph.D. research. He has made a great contribution to this thesis.

Thanks also is owed to Prof. Z. Z. Yong for many helpful discussions and much valuable advice in the investigation of genetic guided clustering; also to the members of the Intelligence Engineering and Automation group, especially Mr. K. W. Lau for our valuable discussion in the investigation of fast training for support vector machines.

I am grateful to the Department of Electrical Engineering and Electronics, the University of Liverpool, for providing the research facilities, making it possible for me to conduct this research. I am indebted to both the University of Liverpool for the studentship (1998-2001) and the Committee of Vice-Chancellors and Principals for the 'Overseas Research Students Awards Scheme' (1998-2001). Thanks is also extended to British Federation of Women Graduates for funding my third year research.

Finally, I am greatly indebted to my parents, for their patience, understanding, encouragement, and love through the whole period of my postgraduate life.

Abstract

PATTERN CLASSIFICATION
USING
ENHANCED MACHINE LEARNING

by

Li MENG

According to the learning approach involved in the problem solving, pattern classification problems can be divided into categories: supervised and unsupervised. Unsupervised pattern classification is also referred to as clustering.

In the first part of the thesis, support vector machines (SVMs) are investigated for supervised pattern classification problems. Based on recent advances in statistical learning theory, SVMs comprise a new class of generation learning systems and have become one of the standard techniques for pattern classification.

To ease the separation of classes in a training set, SVMs map training examples from the input space to the feature space defined by a kernel function. Moreover, to overcome the problem of noise and non-separability, a parameter C has been introduced to allow training errors. The effects of different kernel functions and parameter C have been investigated in this thesis.

A major concern in the SVM is the issue of training, which amounts to solving a quadratic programming (QP) problem with a dense matrix. The present SVM training algorithms have been studied here. In addition, two new algorithms have been proposed for fast training of SVMs. Both of them train a SVM based on the cluster centres. In our first attempt, the set of cluster centres corresponds to partition of the full training set. The method used for clustering the training set is c-means. Despite its wide application,

it is well known that the c-means algorithm for clustering problems is fairly sensitive to initial conditions and can be easily trapped into different local extrema. As a result, different trained machines are obtained after different runs. In spite of this, the largely reduced training time encouraged the usage of a centre-based training method for the support vector machines. In our second attempt, the set of cluster centres corresponds to partition of the errors caused by the current machine. Results of the error-centre-based algorithm show that its computation time scales almost linear in the training set size and thus may be applied to much larger training sets, in comparison with the standard QP techniques.

The second part of the thesis is devoted to a discussion on the problem of clustering. Clustering is concerned with the discovery of natural groups in a population of data. It is useful for data exploration, versions of which are likely to be met, either explicitly or implicitly, in many real-world problems. Clustering problems are NP complete, whose solution space is huge. It is therefore not surprising that nearly all major "modern" techniques have been tried for solving it: artificial neural networks, fuzzy sets, evolutionary algorithms, simulated annealing, etc. Of course, more "classical" analytical and statistical approaches are also being used.

This thesis presents a new method for clustering, which combines the conventional hard c-means with the advanced genetic algorithm (GA) and is thus called genetic hard c-means clustering algorithm (GHCMCA). Experiments on GHCMCA show that a genetic approach is able to overcome the inevitable drawbacks of a hill-climbing technique such as c-means. Vector quantisation is an important application of c-means clustering and images are real-world domains of significant complexity. Inspired by this, the new GHCMCA has been tested on different image data sets, in comparison with the conventional c-means as well as a previous genetic clustering algorithm.

The application of GA to NP-hard problems has been extended to job-shop

problems (JSPs), where a new genetic scheduling algorithm GSA has been developed. A new GA crossover has been designed specially for JSPs to avoid infeasible solutions. Experimental results demonstrate that GSA is simple yet effective.

In this thesis, description of experiments and analysis of simulation results have been included to support the conclusions drawn in each work.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1 Pattern Recognition

The ease with which we recognise a face, understand spoken words, read handwritten characters, identify our car keys in our pocket by feel, and decide whether an apple is ripe by its smell, belies the astoundingly complex processes that underlie these acts of pattern recognition. *Pattern recognition* - the act of taking in raw data and taking an action based on the *category* of the pattern - has been crucial for our survival, and over the past tens of millions of years we have evolved highly sophisticated neural and cognitive systems for such tasks.

It is natural that we should seek to design and build intelligent machines that can recognise patterns. From automated speech recognition, fingerprint identification, optical character recognition, DNA sequence identification, and much more, it is clear that reliable, accurate pattern recognition by machine would be immensely useful. There are a very large number of reviews and books that are devoted to pattern recognition by intelligent machines [1, 2].

The utility of classes and categories is obvious: any object that has been recognised as a member of a certain category inherits the general properties of that category. For example, being told that a horse is a mammal, we im-

mediately know whether the animal lays eggs, whether it can fly, or whether its skin is covered with fur or feathers. On the other hand, categories in the data of a particular area contain knowledge about that area and recognition of them will thus lead to discoveries of associations and cause-effect relationships. The associations between different categories and their causes are in turn the bricks from which the wall of scientific knowledge is built.

Typical pattern recognition system by an example

To illustrate the types of problems involved in pattern recognition, let us consider the following imaginary example. Suppose that a fruit packing plant wants to automate the process of sorting incoming fruit on a conveyor belt according to types. As a pilot project it is decided to try to separate oranges from apples using optical sensing. We set up a camera, take some sample images, and begin to note some physical differences between the two types of fruit - peel colour, peel texture, shape, lightness, and so on - and these suggest *features* to explore for use in our classifier. We also notice noise or variations in the images - variations in lighting, position of the fruit on the conveyor, even the static noise due to the electronics of the camera itself. Given that there truly are differences between the population of oranges and that of apples, we view them as having different models - different descriptions. Usually, we would like to represent these models in a mathematical form. The goal and approach in pattern regression is to hypothesise the class of these models, process the sensed data to eliminate noise (not due to the models), and for any sensed pattern choose the model that corresponds best.

A typical pattern recognition system for performing this specific task might have the form shown in Figure 1.1. First, the camera (sensor) captures an image of the fruit. Next, the camera's signals are preprocessed to simplify subsequent operations without losing relevant information. In particular, we might use a

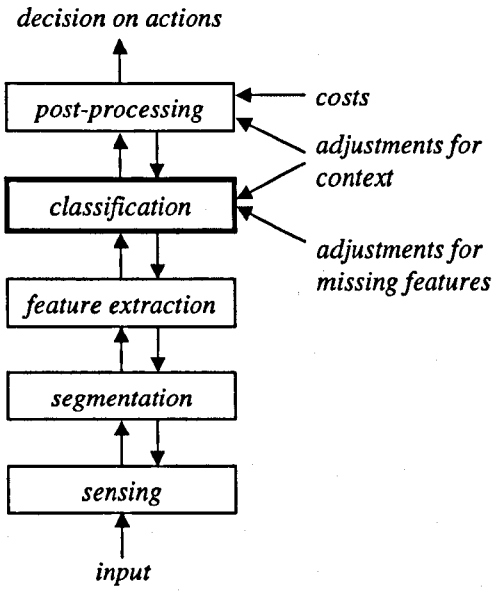


Figure 1.1: Components of a typical pattern recognition system (cited from [3], p. 10). Although this description stresses a one-way or "bottom-up" flow of data, some systems employ feedback from higher levels back down to lower levels (downward arrows).

segmentation operation in which the images of different fruit are somehow isolated from one another and from the background. The information from a single fruit is then sent to a feature extractor, whose purpose is to reduce the data by measuring certain features. These features (or, more precisely, the values of these features) are then passed to a classifier that evaluates the evidence presented and makes a final judgement as to the types. A classifier rarely exists on its own. Instead, it is generally to be used to recommend actions (e.g. put this fruit in this bucket, put that fruit in that bucket), each action having an associated cost. The post-processor uses the output of the classifier to decide on the recommended action.

The performance of a classifier is heavily dependent on the choice of the features that are used for describing the objects. Considering this, the next subsection briefly addresses how the feature extractor should be designed. After that, the main concern of this thesis - the design of the classifier component -

is discussed.

Feature extraction

In pattern recognition, objects are characterised by values of some predefined set of *attributes*, such as shape and colour. Boolean, numeric, symbolic, or mixed-valued attributes can be considered, and the scope of their values is often constrained by background knowledge. A *feature* is either an attribute or a function of one or more attributes. Ideally, the set of features used in a classification decision should be statistically independent, i.e. none of the features can be determined by a function of other features in the set, or estimated from them because of correlations. For instance, if the peel texture of all fruit correlated perfectly with peel colour, then even when we include peel texture as a feature in addition to peel colour classification performance need not be improved. Besides, an ideal feature extractor should yield a representation in which the true (but unknown) model of the patterns can be expressed. Methods generally used for feature extraction are Principal Component Analysis (PCA) [4, 5] and Independent Component Analysis [6].

Classification

The task of the classifier component proper of a full system is to use the data provided by the feature extractor to assign the object to a category. The degree of difficulty of the classification problem depends on the variability in the feature values for objects in the same category relative to the difference between the feature values for objects in different categories. The variability of feature values for objects in the same category may be due to the variation within that category, and may be due to noise. We define noise in very general terms: any property of the sensed pattern which is not due to the true underlying model but instead to randomness in the world or the sensors. All nontrivial decision and pattern recognition problems involve noise in some form.

There are two modes that a classifier executes on the feature vector of an

object, corresponding to the two processes - learning and reasoning - of an intelligent system. One is the operational mode where the classifier maps each input feature vector onto an output vector that represents the class decision. This decisionmaking is often referred to as *recognition*. Before a system can do this, however, it must have first learned the categories of feature vectors through a process that partitions the set of feature vectors. This is the process of *classification*, which involves training or machine learning. Most of this thesis is concerned with the training of a classifier.

1.2 Problems and Learning Approaches

According to the learning approach involved in the problem solving, pattern classification problems can be divided into two categories: supervised and unsupervised. Unsupervised pattern classification is also referred to as clustering. The rest of this section addresses the problems of supervised pattern classification and clustering, respectively.

1.2.1 Supervised pattern classification

Learning becomes supervised when a pre-partitioned training set is available. For supervised pattern classification, a teacher provides a category label or cost for each pattern in a training set, and seeks to reduce the sum of the costs for these patterns. The category label can be either nominal- or numeric-valued.

A major concern in supervised learning is the issue of *generalisation*. If the model proposed is too simple, the resulting classifier may perform badly even on the training patterns. If our model is extremely complicated, the classifier may have a decision boundary more complex than the one obtained using a simple model and all the training patterns will be separated perfectly.

With such a solution, though, our satisfaction would be premature because the central aim of designing a classifier is to suggest actions when presented with novel patterns (that is, examples not yet seen). This is the need for good generalisation. It is unlikely that a extremely complex decision boundary would provide good generalisation - it seems to be tuned to the particular training examples, rather than some underlying characteristics or true model of all the individuals in each class that will have to be separated. But if designing a very complex classifier is unlikely to give a good generalisation, precisely how should we quantify and favour simpler classifiers and how our system would automatically detect noisy examples and determine that a decision boundary can generalise better than the simpler and more complicated ones? Assuming that we somehow manage to optimise this tradeoff, can we then *predict* how well our system will generalise to new patterns?

The above discussion outlines some of the key problems encountered in *statistical pattern classification.* In the book on statistical learning theory [7], a upper bound on the error in generalisation is presented. And based on it, the principle of *structure risk minimisation* (SRM) is established. Implementing this principle, support vector machines (SVMs) have become one of the standard techniques for pattern classification [8] - [14] and nonlinear regression [15] - [17]. The use of SVMs in pattern classification has been investigated in this thesis.

1.2.2 Clustering

The aim of clustering is to group individuals in a population such that, in some sense, the individuals within a group are close or similar to one another, but dissimilar from the individuals in other groups. Clustering is a form of *unsupervised learning*, where no explicit teacher is available and the system forms clusters or *natural* groupings of the input examples. "Natural" is always

defined explicitly or implicitly in the clustering system itself. Given a particular set of examples and objective function, different clustering algorithms lead to different partitions.

Clustering problem has been proven to be NP-complete and thus hard to be solved by general techniques as the number of input examples is increased. The space of all possible partitions of n examples, which is the search space to be scanned by the learning methods, is huge. For example, for a set of 50 examples, the order of magnitude is 10^{47}; and 100 examples can be partitioned into 5 classes in approximately 10^{68} different ways. In addition, most of the objective function involved in clustering problems are highly nonlinear and possess a number of local extrema. When the natural clusters are not "obvious" among the input examples, the problem becomes more difficult as both the nonlinearity of the objective function and the number of local extrema would increase dramatically.

When the clusters to be formed are required not to overlap with each other, the problem becomes a *hard clustering problem. Hard c-means clustering algorithm* (HCMCA) is a popular method for such problems. It is a sum-of-squares method, which partitions the examples into c clusters so that a defined within-group sum-of-squares is minimised. Starting from c initial cluster centres, HCMCA alternatively assigns each example in the training set to its closest centre and then updates the c centres according to the current partition. HCMCA is efficient since it is essentially a hill-climbing approach, guiding the search in the direction that the value of the cost function decreases most rapidly. However, it does have weaknesses:

- The way to initialise the searching is not specified by the algorithm. One popular way to start is to randomly choose c of the examples.
- The results produced depend on the initial values of the centres, and it frequently happens that suboptimal partitions are found. The standard

solution is to try a number of different starting points. However, it is obvious that the optimal partition can not be guaranteed no matter how many starting points would be tried.

- It can happen that the set of examples closest to a cluster centre is empty and consequently that the centre cannot be updated [18]. This is an annoyance that must be handled in an implementation.

The problem of how to conquer these weaknesses has become another main focuses of this thesis, besides the use of SVMs for supervised pattern classification.

1.3 Main Results/Algorithms

1.3.1 Statistical pattern classification using support vector machines

To ease the separation of the classes in a training set, SVMs map the training set from the input space to the feature space defined by a kernel function. Different kernel functions implement different mappings, which will in turn determine the separability of the mapped classes in the feature space. For different data sets, the highest separability in the feature space is achieved by different kernel functions. How to choose the best kernel function for a particular data set still remains an open question. In this thesis, the commonly used kernel functions are introduced and their effects are investigated empirically by comparing the decision boundaries that are obtained when applying different kernels to a same benchmark data set.

To solve the problem of noise and the non-separability caused by noisy examples, a parameter C has been introduced in SVMs to allow training errors. The effects of different values of parameter C have been investigated, again em-

pirically, by comparing the resulting decision boundaries on a same benchmark problem.

Training a SVM amounts to solving a quadratic programming (QP) problem. The standard QP techniques require a memory space growing quadratically with the problem size and take a training time that grows exponentially with the problem size. To tackle this problem, two centre-based algorithms have been proposed for a fast training of SVMs. Under these new algorithms, examples in a given training set are partitioned into clusters and the SVM is trained on the centres of these clusters. The partition of training examples is updated repeatedly for producing a better decision boundary. Under our first attempted algorithm - Centre-based optimisation (CO) [19], the partition is updated by splitting each current cluster into two new clusters. This is achieved by the classic HCMCA. And as a result, for a same training set different partitions and hence different decision boundaries might be obtained after independent runs of the algorithm. Despite this, the largely reduced training time encouraged the usage of a centre-based training method for the SVMs. To avoid the uncertainty of the results, a method rather than HCMCA is required for the construction of new clusters. In the training of a classifier, errors are those examples that are classified, according to the current decision boundary, to a class rather than the taught one. In our second attempt, errors in each of the current clusters compose a new cluster. By such means, the consistency of the obtained results is achieved. The procedure of updating the partition of the training examples, training the SVM on the set of cluster centres, and identifying errors caused by the current decision boundary iterates until no error example is found. Since only are the centres of error clusters involved in the training, this new algorithm is called *error-centre-based optimisation* (ECO) [20] - [22]. Under ECO, error examples are identified using the Karush-Kuhn-Tucker (KKT) conditions. The KKT conditions are the necessary and sufficient conditions for the optimal solution of a QP problem. Therefore, the optimality

of solutions produced by ECO is guaranteed. The great potential of ECO for large training sets has been demonstrated by the experimental results. While the optimal solution is guaranteed by ECO, the size of QP problems involved in the training is largely reduced and the training time grows almost linearly with the size of the training set. The experiment results obtained using ECO are presented and analysed in this thesis. So are the results obtained using CO.

1.3.2 *c*-Means clustering using a genetically guided learning approach

As already mentioned in section 1.2.2, by hill-climbing, HCMCA is found to be easily trapped into different local minima and fairly sensitive to the initial conditions. When the models are fairly simple and of low dimension, the standard *c*-means approach can still perform well. However, it becomes increasingly unsatisfactory as the models become more complicated. The more complicated the model, the less the prior knowledge, and the less the training data, the more we must rely on sophisticated search methods for constructing an acceptable model.

Genetic algorithms (GAs) comprise a general class of such methods. The class of GAs is based on the concepts stemming from biology, specially the principles of natural evolution. A primary advantage of GAs is that they conduct a global search and thus can effectively overcome the inevitable drawbacks of a local search algorithm, such as HCMCA. Inspired by this, a genetically guided approach - *genetic hard c-means clustering algorithm* (GHCMCA) - has been proposed specially for hard clustering problems [23] - [25]. GHCMCA adopts the basic scheme of a genetic algorithm (GA) and performs a genetically guided search in order to optimise the objective function of a hard clustering problem. Unlike previous clustering algorithms with GAs, population members under

GHCMCA represent partition matrices instead of sets of cluster centres. A new genetic crossover operator has been introduced which effectively recombines important partition similarities between pairs of parents creating new solutions. GHCMCA has been evaluated and compared against the traditional clustering algorithm HCMCA and a previous genetic clustering algorithm which adopts the standard two-point crossover. Results from the comparative study show that a genetic approach is able to overcome the inevitable drawbacks of a hill-climbing technique. In experiments with image data sets, the proposed algorithm is superior to the previous genetic clustering algorithm in the sense that it converges much more quickly to the desired region in which the global optimum resides. Therefore in the cases where speed as well as performance is required, GHCMCA provides a solution to the dilemma where the classical HCMCA can be easily trapped in different local extrema and the conventional genetic approach is time consuming.

In this thesis, the application of GAs has been extended to another NP-complete problem - the job-shop problem. A new genetic scheduling algorithm GSA has been proposed. GSA employs a new solution representation scheme and a new crossover operator, both have been devised specially for the job-shop problems to avoid infeasible solutions. The power of GSA has been demonstrated by the experimental results. For each testing problem, GSA has found the optimal solution.

1.4 Outline of Thesis

This thesis is organised as follows.

We begin in Chapter 2 (Statistical Learning and Support Vector Machines) with an introduction to the structure risk minimisation principle in statistical learning theory. Then the statistical learning method - support vector machines (SVMs) is introduced. Both maximal-margin and soft-margin SVMs

are addressed. The effect of parameter C in a soft-margin SVM has been investigated. When the classes of points cannot be separated by a hyperplane, SVMs first map the points into a high-dimensional feature space defined by a kernel function. In Chapter 3 (Kernel-Induced Feature Space) the theorem for testing the validity of a kernel function and different types of kernel-induced feature spaces are discussed. Chapter 4 (Support Vector Machine Training and Its Implementation Algorithms) considers the issue of SVM training. Different training algorithms are presented, including the two new centre-based methods proposed by the author.

Then from Chapter 5 (Unsupervised Learning and Clustering) we move to the investigation of unsupervised pattern classification. We begin the chapter by formulating the problem of clustering and the unsupervised learning method HCMCA for solving it. Chapter 5 also demonstrates the inevitable drawbacks of c-means due to its hill-climbing nature and brings up the idea of using a genetic approach to overcomes these drawbacks. In Chapter 6 (Fundamentals of Genetic Algorithms) the genetic algorithms (GAs) as well as the relevant algorithms in the area of evolutionary computation are addressed. Alternative schemes of the GA operators are discussed and the basic theorems in the GA literature are described. In Chapter 7 (A Genetic Hard c-Means Clustering Algorithm) the new algorithm combining the traditional c-means and the advanced GA is presented. Comparative experiments are included to show the robustness of this hybrid algorithm. Chapter 8 (Application of GA to Job-Shop Problems) presents the new genetic scheduling algorithm GSA. Its power has been demonstrated by the experimental results.

This thesis is concluded in Chapter 9 (Conclusion) by giving a summary of the results obtained and also several suggestions for future work.

Chapter 2

STATISTICAL LEARNING AND SUPPORT VECTOR MACHINES

2.1 Introduction

Support vector machines (SVMs) based on recent advances in statistical learning theory [7, 26], compose a specific class of learning systems. SVMs were invented by Boser, Guyon and Vapnik. It were first introduced at the conference of Computational Learning Theory (COLT) 1992 with the paper [27]. To overcome the problem of noise and non-separability, the soft-margin version [8] was later introduced. SVMs were developed originally for solving the classification problems. While in 1997 the algorithm was extended for solving regression problems [28]. SVMs have now become one of the standard techniques for pattern classification and nonlinear regression. This chapter introduces SVMs in the setting of pattern classification. Many of the ideas may carry directly over to the case of regression estimation. Successful applications of SVM classifiers have been reported for various fields including isolated

handwritten digit recognition [8] - [11], text categorisation [12] - [14], speaker identification [29], gene expression profile analysis [30, 31], DNA and protein analysis [32] - [33], and many more. Many of these recent advances are reported in the collection [34]. Most of the new contributions are only available online and can be recently accessed via the website [35].

Each SVM can be used to solve a particular two-class classification problem. While the classification problem can be restricted to consideration of two-class problems without loss of generality. To get a k-class classifier ($k > 2$), we can just construct k two-class classifiers, each separating a certain class from the others, and combine them by doing the k-class classification according to the maximal output among the two-class classifiers. In a two-class classification problem, one possible formulation of the task is to separate the two classes by a function which is induced from some available examples. The goal is to produce a classifier that will work well on unseen examples, i.e. it generalises well. To achieve this, a SVM provides a decision boundary that separates a set of positive examples from a set of negative examples with the maximum margin. Although intuitively simple, this idea of maximum margin actually exploits the structural risk minimisation (SRM) principle in statistical learning theory. Thus the learned machine will not only have a minimal empirical risk but also good generalisation performance.

Before describing different types of SVMs, the next section is devoted to a brief introduction to some basic concepts in the theory of statistical learning.

2.2 Fundamentals of Statistical Learning Theory

Figure 2.1 shows a model of supervised learning process. The model consists of three interrelated components:

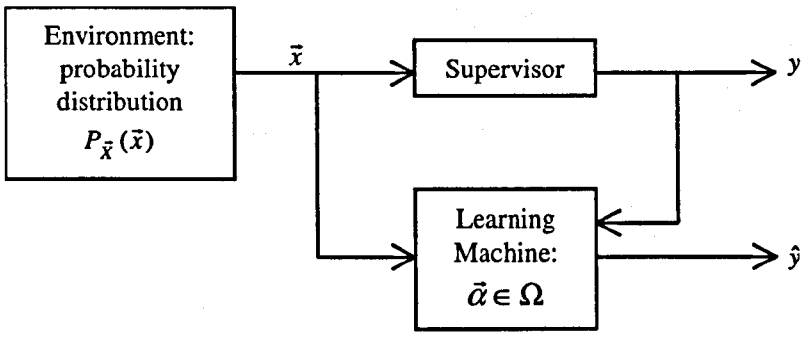


Figure 2.1: Model of the supervised learning process.

1. *Environment.* In our work, it is assumed that the environment is stationary, supplying the vectors ($\vec{x} \in \vec{X}$, $y \in Y$) *independently* and *identically distributed* (i.i.d.) according to a fixed but unknown probability distribution $P_{\vec{X},Y}(\vec{x}, y)$.

2. *Supervisor.* The supervisor or teacher provides a target response y for every input vector $\vec{x}$ received from the environment. The target function which transforms the vectors $\vec{x}$ into values y, is unknown, but exists and does not change.

3. *Learning machine.* The learning machine (e.g. a classifier) is capable of implementing a class of input-output mapping functions described by

$$\hat{y} = f(\vec{x}, \vec{\alpha})$$

where $\hat{y}$ is the actual response produced by the learning machine in response to an input $\vec{x}$, and $\vec{\alpha}$ is a set of free parameters. A particular choice of $\vec{\alpha}$ generates a "trained machine".

During the learning process, the learning machine observes a set of l pairs - the *training set*

$$(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots, (\vec{x}_l, y_l).$$

Each pair contains an input vector $\vec{x}$ and the target response y. After training, the machine must give, on any $\vec{x}$ generated by the environment according to the same probability distribution, a value $\hat{y}$. The best function f of a learning machine that one can obtain is the one minimising the *expected error* (or *risk*)

$$\begin{aligned} R(f(\alpha)) &= \int_{\vec{X},Y} \frac{1}{2}|y-\hat{y}|\, dP_{\vec{X},Y}(\vec{x},y) \\ &= \int_{\vec{X},Y} \frac{1}{2}|y-f(\vec{x},\vec{\alpha})|\, dP_{\vec{X},Y}(\vec{x},y) \end{aligned} \tag{2.2.1}$$

$R(f(\alpha))$ defines the frequency of errors made by a trained machine when it is tested with examples not seen before. $R(f(\alpha))$ is also called the *actual risk* to emphasise that it is the quantity that we are ultimately interested in. Unfortunately, the risk cannot be minimised directly since the underlying probability distribution $P_{\vec{X},Y}(\vec{x},y)$ is unknown. An approximation is therefore required.

According to the statistical learning theory [7, 26, 36], the following bound holds with probability $1-\xi$ $(0 \leq \xi \leq 1)$

$$R(f(\alpha)) \leq R_{\text{emp}}(f(\alpha)) + \sqrt{\frac{h(\log(2l/h)+1) - \log(\xi/4)}{l}}, \tag{2.2.2}$$

where h is a non-negative integer called the *Vapnik Chervonenkis (VC) dimension*, and $R_{\text{emp}}(f(\alpha))$ is the *empirical risk* defined as

$$R_{\text{emp}}(f(\alpha)) = \frac{1}{2l}\sum_{i=1}^{l}|y_i - f(\vec{x}_i,\vec{\alpha})|, \tag{2.2.3}$$

i.e. the measured mean error rate made by the learning machine on the training set.

The difference between the expected risk and the empirical risk is bounded by the second term on the right hand side of inequality (2.2.2) which is called *confidence interval*. As the training set size $l \to \infty$, the empirical risk will converge toward the expected risk. The quantity $\frac{1}{2}|y_i - f(\vec{x}_i,\vec{\alpha})|$ in the definition

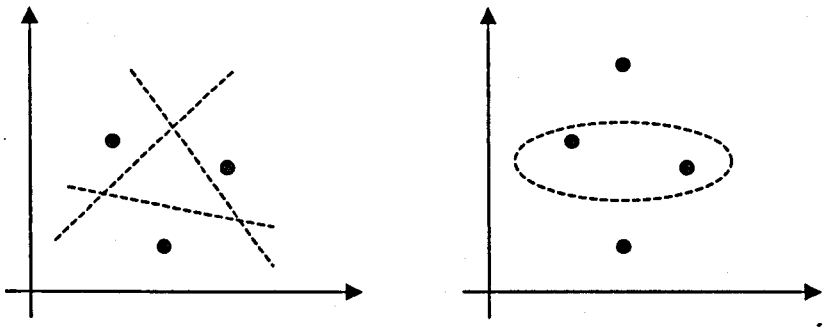


Figure 2.2: Illustration of VC dimension of the function class of lines.

of empirical risk is called the *loss*. To have equation (2.2.2) tenable, the loss at each example must be either 0 or 1, such as in the problem of two-class classification where $y_i \in \{-1, 1\}$.

2.2.1 VC dimension

VC dimension is a scalar value that measures the capacity/complexity of a function class F.

If a given set of l points can be labelled in all 2^l ways, and for each labelling, a function f in class F can be found which correctly assigns those labels, then that set of points is said to be *shattered* by function class F. The VC dimension for a function class F is defined as the maximum number of points that can be shattered by it. The VC dimension of the class of oriented hyperplanes in $\mathcal{R}^n$ is $n+1$. Figure 2.2 illustrates how three points on a plane can be shattered by the class of straight lines whereas four points cannot.

2.2.2 Structure risk minimisation (SRM)

As revealed by equation (2.2.2), for a given learning task, with a given finite amount of training data, the best generalisation performance will be achieved if right balance is reached between the empirical risk (i.e. accuracy attained

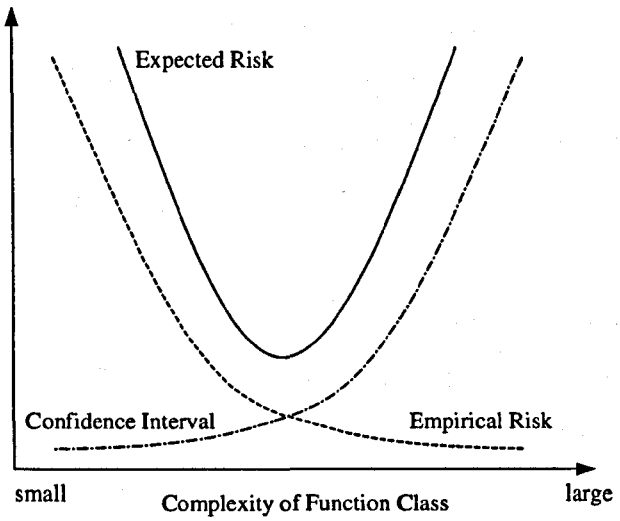


Figure 2.3: Illustration of the relationships among empirical risk, confidence interval and expected risk.

on that particular training set) and the capacity of machine (i.e. the ability of machine to learn *any* training set without error). A machine with too much capacity is like a botanist with a photographic memory who, when presented with a new tree, concludes that it is not a tree because it has a different number of leaves from any tree he has seen before; a machine with too little capacity is like the botanist's lazy brother, who declares that if it is green, it is a tree. Neither can generalise well. The exploration and formalisation of these concepts has resulted the *structural risk minimisation* (SRM) principle in the theory of statistical learning [36].

Constructing a nested family of function classes

$$F_1 \subset \ldots \subset F_h \subset \ldots \subset F_k \subset \ldots \quad (1 < h < k)$$

with nondecreasing VC dimensions, the SRM principle then consists of solving the following problem

$$\min_{\mathcal{F}_h} \left\{ R_{\text{emp}}\left(f(\alpha)\right) + \sqrt{\frac{h(\log(2l/h)+1) - \log(\eta/4)}{l}} \right\}, \tag{2.2.4}$$

i.e. finding the class of functions that minimises the bound on the expected

risk. Two extremes arise for equation (2.2.4): 1). a very small function class (like F_1) yields a vanishing square root term, but a large empirical error might remain; while 2). a huge function class (like F_k) may give a vanishing empirical error but a large square root term. The best class is usually in between (see Figure 2.3), as one would like to obtain a function that explains the data quite well (i.e. with a small empirical risk) *and* to have a small risk in obtaining that function (i.e. with small capacity in terms of VC dimension). This is much in analogy to the bias-variance dilemma scenario reported for neural networks [37].

In a later section, we will see how SRM principle has been successfully implemented by SVMs.

2.3 Maximal-Margin Optimisation: the Separable Case

2.3.1 SVMs with a maximal margin

Consider a training set or sample $\{(\vec{x}_i, y_i)\}_{i=1}^{l}$, where target response y_i is the label indicating which class the input example $\vec{x}_i$ belongs to. $\vec{x}_i \in \mathcal{R}^d$ and for two-class classification, $y_i \in \{-1, 1\}$. This section starts from the simplest case of pattern classification: a set of linearly separable data, i.e. there is a hyperplane which separates the positive examples from the negative. Points on the separating hyperplane satisfy

$$f(\vec{x}) = \vec{w} \cdot \vec{x} + b = 0, \tag{2.3.1}$$

where $\vec{w}$ is the normal to the separating hyperplane, and $|b|/\|\vec{w}\|$ the perpendicular distance from the hyperplane to the origin. $\vec{w}$ and b are the parameters to be optimised for a maximum margin. Let d^+ (d^-) be the distance from the separating hyperplane to the closest positive (negative) examples. See Figure

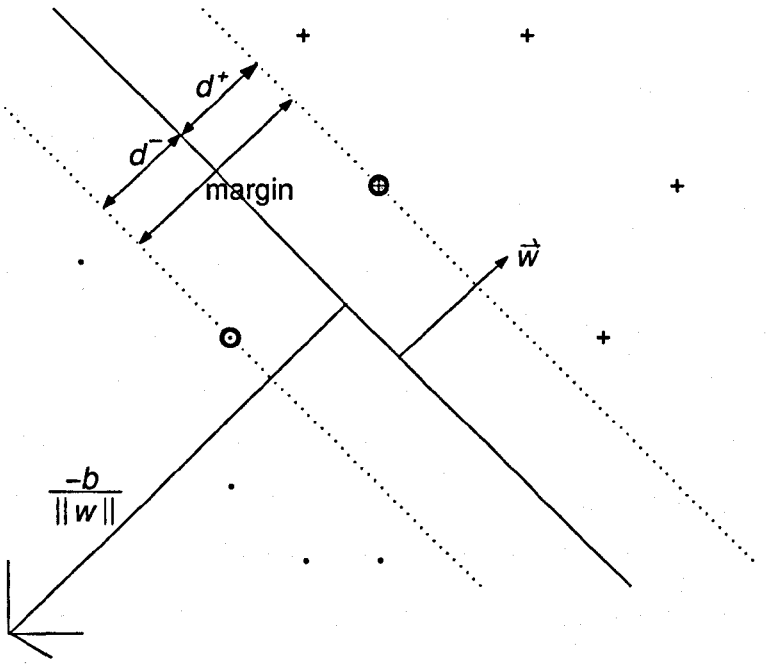


Figure 2.4: Illustration of a linear SVM with the maximal margin. Points closest to the separating hyperplane are marked with circles.

2.4 for an illustration. Define the *geometric margin* or simply the *margin* of a separating hyperplane to be the area that is bounded by the set of closest examples. It is possible to make all the closest positive and negative points satisfy $y_i f(\vec{x}_i) = 1$, i.e. the closest examples that lie on one of the two hyperplanes parallel to but apart from the separating hyperplane by 1.

In such a case, the following constraints hold

$$y_i(\vec{w} \cdot \vec{x}_i + b) - 1 \geq 0, \ i = 1, \ldots, l. \tag{2.3.2}$$

Consider the positive points for which the equality of equation (2.3.2) holds. These points lie on the hyperplane $H_1 : \ \vec{w} \cdot \vec{x}_i + b = 1$ with normal $\vec{w}$ and perpendicular distance from the origin $|1 - b|/\|\vec{w}\|$. Similarly, the negative points for which the equality of equation (2.3.2) holds lie on the hyperplane $H_2 : \ \vec{w} \cdot \vec{x}_i + b = -1$ with normal $\vec{w}$ and perpendicular distance from the origin $|-1 - b|/\|\vec{w}\|$. Note that H_1 and H_2 are parallel and that no training examples may fall between them. Hence, under constraints (2.3.2), $d^+ = d^- = 1/\|\vec{w}\|$

and the width of the margin is simply $2/\|\vec{w}\|$. We can therefore find the separating hyperplane with the maximum margin by minimising $\|\vec{w}\|^2$ subject to constraints (2.3.2).

The optimal separating hyperplane is independent of b because that provided constraints (2.3.2) are satisfied (i.e. it is a separating hyperplane) changing b will move the hyperplane in the direction of its normal. Accordingly the margin remains unchanged but the constraints (2.3.2) is no longer satisfied in that the separating hyperplane will be nearer to one class than the other.

As observed in many practical cases, the decision boundary between two classes can not be defined by a linear function. Then how can the above method be generalised to the nonlinear cases. This has been accomplished by Boser, Guyon and Vapnik [27] in an astonishingly straightforward way. For nonlinearly separable problems, a nonlinear mapping is introduced before the construction of the separating hyperplane, which transforms the training examples from the input space to a higher-dimensional *feature space.* Let Φ denote this nonlinear mapping

$$\Phi : \mathcal{R}^d \mapsto \mathcal{F}. \tag{2.3.3}$$

The separating hyperplane is then constructed in the feature space $\mathcal{F}$. This yields a nonlinear decision boundary in the input space $\mathcal{R}^d$, which is composed of the points whose mapped points in the feature space are on the separating hyperplane there. The nonlinear mapping is performed in accordance with Cover's theorem on the separability of patterns.

Cover's theorem [38]: A complex pattern classification problem cast in a high-dimensional space nonlinearly is more likely to be linearly separable than in a low-dimensional space.

The idea of mapping from the input space to a feature space has also been adopted in the theory of neural networks, where the mapping is implemented by the hidden layer(s) between the input and output layers.

The optimisation problem associated with the SVM training, for both linear and nonlinear cases, is summarised as follows

$$\text{OP1}: \quad \min_{\vec{w},b} \quad \frac{1}{2}\|\vec{w}\|^2, \tag{2.3.4}$$

$$\text{subject to} \quad y_i(\vec{w}\cdot\Phi(\vec{x}_i)+b) \geq 1, \ i=1,\ldots,l. \tag{2.3.5}$$

The constraints enforce that no training examples may fall inside the margin. The training of SVM on a linearly separable data set is a special case of OP1, in which $\Phi(\vec{x}) = \vec{x}$. For other data sets, we are still doing a linear separation, but in a different space - the high-dimensional feature space.

2.3.2 Solving the SVM optimisation problem

The constrained optimisation problem OP1 is solved by introducing Lagrange multipliers $\vec{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_l\}$ and a Lagrangian

$$L(\vec{w}, b, \vec{\alpha}) = \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{l} \alpha_i[y_i(\vec{w}\cdot\Phi(\vec{x}_i)+b) - 1], \tag{2.3.6}$$

where the Lagrangian $L(\vec{w}, b, \vec{\alpha})$ must be minimised with respect to the *primal variables* $\vec{w} = \{w_1, w_2, \ldots, w_d\}$ and b and maximised with respect to the *dual variables* $\vec{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_l\}$. That is to say, a saddle point must be found.

The minimisation of $L(\vec{w}, b, \vec{\alpha})$ with respect to the primal variables requires that at the saddle point

$$\frac{\partial}{\partial \vec{w}} L(\vec{w}, b, \vec{\alpha}) = 0 \quad \text{and} \quad \frac{\partial}{\partial b} L(\vec{w}, b, \vec{\alpha}) = 0.$$

These lead to

$$\vec{w} = \sum_{i=1}^{l} y_i \alpha_i \Phi(\vec{x}_i) \tag{2.3.7}$$

and

$$\sum_{i=1}^{l} y_i \alpha_i = 0 \tag{2.3.8}$$

Substituting these equations back into the primal problem OP1, we have the dual problem

$$\text{OP2}: \quad \min_{\vec{\alpha}} \quad -\sum_{i=1}^{l}\alpha_i + \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} y_i y_j \alpha_i \alpha_j \left(\Phi(\vec{x}_i)\cdot\Phi(\vec{x}_j)\right), \tag{2.3.9}$$

$$\text{subject to} \quad \sum_{i=1}^{l} y_i \alpha_i = 0, \tag{2.3.10}$$

$$\alpha_i \geq 0, \; i = 1, \ldots, l. \tag{2.3.11}$$

With the optimal values of $\vec{\alpha}$, the decision boundary in the input space is given by

$$f(\vec{x}) = \sum_{i=1}^{l} \alpha_i^* y_i \Phi(\vec{x}_i)\cdot\Phi(\vec{x}) + b^* = 0, \tag{2.3.12}$$

where b^* is found by employing the primal constraints (2.3.2)

$$b^* = -\frac{\max_{y_i=-1}(\vec{w}^*\cdot\Phi(\vec{x}_i)) + \min_{y_i=1}(\vec{w}^*\cdot\Phi(\vec{x}_i))}{2}. \tag{2.3.13}$$

The decision function for classifying new examples is defined as

$$\text{sgn}(f(\vec{x})). \tag{2.3.14}$$

Notice that the only way in which the data appears in problem OP2 (equations (2.3.9) - (2.3.11)) is in the form of inner products - $(\Phi(\vec{x}_i)\cdot\Phi(\vec{x}_j))$. So does in equations (2.3.12) and (2.3.13) for the resulting decision boundary. This allows a SVM, without ever representing the feature space explicitly, to locate a separating hyperplane in the feature space and classify vectors in that space by simply specifying a *kernel function*, $K(\vec{x}_i, \vec{x}_j) = \Phi(\vec{x}_i)\cdot\Phi(\vec{x}_j)$. One example of kernel function is

$$K(\vec{x}_i, \vec{x}_j) = e^{-\|\vec{x}_i - \vec{x}_j\|^2/2\sigma^2}$$

In this particular example, $\mathcal{F}$ is infinite dimensional, so it would not be very easy to work with $\Phi(\vec{x})$ explicitly. However, if one replaces $(\Phi(\vec{x}_i)\cdot\Phi(\vec{x}))$ by $K(\vec{x}_i, \vec{x}_j)$ everywhere in the training, an algorithm would easily produce a

SVM which lives in an infinite dimensional space, while using roughly the same amount of time that it would take to train on the un-mapped examples.

Rewriting OP2 in terms of $K(\vec{x}_i, \vec{x}_j)$, we have

$$\begin{aligned} \text{OP3}: \quad \min_{\vec{\alpha}} \quad & -\sum_{i=1}^{l} \alpha_i + \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} y_i y_j \alpha_i \alpha_j K(\vec{x}_i, \vec{x}_j), \quad (2.3.15) \\ \text{subject to} \quad & \sum_{i=1}^{l} y_i \alpha_i = 0, \\ & \alpha_i \geq 0, \ i = 1, \ldots, l. \end{aligned}$$

And the decision boundary becomes

$$f(\vec{x}) = \sum_{i=1}^{l} \alpha_i^* y_i K(\vec{x}_i, \vec{x}) + b^* = 0 \quad (2.3.16)$$

2.3.3 Karush-Kuhn-Tucker (KKT) conditions

The Karush-Kuhn-Tucker (KKT) conditions [39, 40] play a central role in both the theory and practice of constrained optimisation, which define the necessary and sufficient conditions for a set of variables to be optimal for an optimisation problem and thus provide a mathematical characterisation of the solutions.

Applying the KKT conditions to problem OP1, we know that the optimal solution $\vec{\alpha}^*, (\vec{w}^*, b^*)$ must satisfy [41]

$$\alpha_i^* \left[y_i (\vec{w}^* \cdot \Phi(\vec{x}_i) + b^*) - 1 \right] = 0, \ i = 1, \ldots, l. \quad (2.3.17)$$

This implies that

$$\alpha_i^* = 0 \ \Leftrightarrow \ y_i f(\vec{x}_i) \geq 1; \quad (2.3.18)$$

$$\alpha_i^* > 0 \ \Leftrightarrow \ y_i f(\vec{x}_i) = 1, \quad (2.3.19)$$

i.e. only for the examples for which $y_i f(\vec{x}) = 1$ and that hence lie on the margin boundaries are the corresponding α_i^* non-zero. These examples are termed

support vectors. Equations (2.3.18) and (2.3.19) indicate that only part of examples are involved in the expression for the separating hyperplane (2.3.12). Even if the other examples are removed from the training set, recalculating the hyperplane would produce the same answer. This can also be seen from the dual problem, since removing the items containing non-support vectors leaves the same optimisation problem. This implies that SVMs can be used to summarise the information contained in a data set by the set of support vectors produced. Moreover, in many practical applications, only a small percentage of the training examples are support vectors. This feature is referred to as the *sparseness* of the solution to SVM optimisation. It is this feature that makes applying a SVM obtained with large training sets to new testing data computationally tractable. And it also is this feature that has inspired a series of fast training algorithms for SVMs.

2.4 Soft-Margin Optimisation: the Nonseparable Case

Of course, not all data sets are separable even in the high-dimensional feature space, e.g. when a high level of noise causes a large overlap of the classes of patterns. In the problem formulation above, the non-separable case would result in an infinite solution caused by overfitting, where α_i $(i = 1, \ldots, l)$ are infinitely large and margin is consequently infinitely small. In 1995, Cortes and Vapnik [8] introduced *soft-margin* SVMs which allow, but penalise, the failure of examples to keep the margin constraints (2.3.5). The primal problem in the soft-margin version of SVM optimisation is a modification of OP1 and is defined as follows

$$\text{OP4}: \quad \min_{\vec{w},b,\vec{\xi}} \quad \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{l}\xi_i, \tag{2.4.1}$$

$$\text{subject to} \quad y_i(\vec{w}\cdot\Phi(\vec{x}_i)+b) \geq 1-\xi_i, \; i = 1,\ldots,l, \tag{2.4.2}$$

where $\xi_i \geq 0$ $(i = 1, \ldots, l)$ are the slack variables that allow margin failures and C is a parameter that trades off wide margin with a small number of margin failures. Corresponding to soft-margin SVMs, SVMs with a maximal margin are also called *hard-margin* SVMs since they allow no margin errors.

Introducing Lagrange multipliers $\vec{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_l\}$ and $\vec{\beta} = \{\beta_1, \beta_2, \ldots, \beta_l\}$, and a Lagrangian

$$L(\vec{w}, b, \vec{\xi}, \vec{\alpha}, \vec{\beta}) = \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{l}\xi_i - \sum_{i=1}^{l}\alpha_i[y_i(\vec{w}\cdot\Phi(\vec{x}_i) + b) - 1 + \xi_i] - \sum_{i=1}^{l}\beta_i\xi_i,$$

and then minimising the Lagrangian with respect to $\vec{w}, b, \vec{\xi}$ and maximising it with respect to $\vec{\alpha}$, $\vec{\beta}$, where $\alpha_i, \beta_i \geq 0, \forall i$, we have the dual form of OP4 as follows

$$\text{OP5}: \quad \min_{\vec{\alpha}} \quad -\sum_{i=1}^{l}\alpha_i + \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} y_i y_j \alpha_i \alpha_j K(\vec{x}_i, \vec{x}_j), \tag{2.4.3}$$

$$\text{subject to} \quad \sum_{i=1}^{l} y_i \alpha_i = 0, \tag{2.4.4}$$

$$0 \leq \alpha_i \leq C, \; i = 1, \ldots, l. \tag{2.4.5}$$

Note that the slack variables ξ_i do not appear in the objective function (2.4.3) at all and that problem OP5 simply changes the constraint (2.3.11) of separable case into a box constraint.

The KKT conditions of OP5 are

$$\alpha_i^* \left[y_i(\vec{w}^* \cdot \vec{x}_i + b^*) - 1 + \xi_i\right] = 0, \qquad i = 1, \ldots, l, \tag{2.4.6}$$

and

$$\xi_i(\alpha_i^* - C) = 0, \qquad i = 1, \ldots, l, \tag{2.4.7}$$

implying that

$$\alpha_i = 0 \;\Leftrightarrow\; y_i f(\vec{x}_i) \geq 1; \tag{2.4.8}$$

$$0 < \alpha_i < C \;\Leftrightarrow\; y_i f(\vec{x}_i) = 1; \tag{2.4.9}$$

$$\alpha_i = C \;\Leftrightarrow\; y_i f(\vec{x}_i) \leq 1. \tag{2.4.10}$$

i.e. only for the examples that lie on the margin boundaries are the corresponding α_i^* not at the bounds; all examples for which the corresponding α_i^* equal to zero must be correctly classified and lie outside the margin. Furthermore, equation (2.4.7) indicates that non-zero slack variables can only occur when $\alpha_i = C$ and all margin errors are therefore penalised.

2.5 Effect of Trade-off Parameter C

2.5.1 An artificial data set with linear Kernel

Figures 2.5 - 2.7 show the decision boundaries obtained for a linearly non-separable training set when trained with different values of the trade-off parameter C. This data set is artificial and contains a noisy example. The two classes are respectively denoted by red '.'s and blue '+'s while the noisy example is pointed out with an arrow. Each support vector is identified with an extra circle. The margin errors that violate constraints (2.3.5) are identified with a cross. The corresponding numerical results are listed in Table 2.1, along with the results for $C = \infty$.

As shown in Figures 2.5 - 2.7, examples are not longer forbidden to lie inside the margin and may even be classification errors (i.e. may be classified to the class rather than the taught one). For different values of C, the width of the margin are different. As the value of C increases, the width of the margin decreases. This is apparent since:

1). Equation (2.4.10) indicates that all margin errors have the corresponding α_i reach the upper bound, C;

2). From equation (2.3.7), $\|\vec{w}\|^2$ increases with $\vec{\alpha}$; while

3). Margin $= 2/\|\vec{w}\|^2$.

Figures 2.5 and 2.6 show the resulting decision boundaries for $C = 0.5$ and 1.0, respectively. In each case, C is small and the margin errors are not heavily

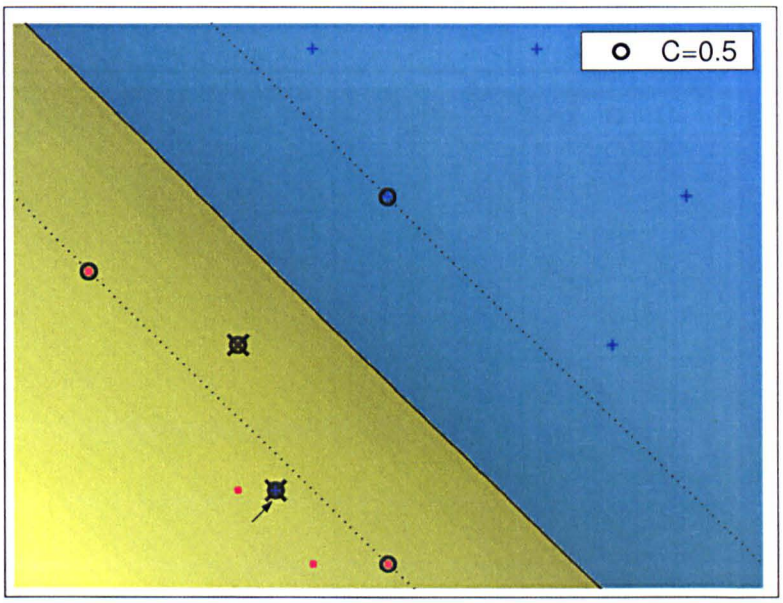


Figure 2.5: Example of a linearly non-separable training set ($C = 0.5$).

penalised. For both cases, two margin errors are generated during the training. As C increases the result converges towards the one shown in Figure 2.7, where all the training examples satisfy the constraints on margin (2.3.5) except the noisy example. Although wider margins can be obtained with smaller C, some points containing important classification information can be lost, as in the cases of $C = 0.5$ and $C = 1.0$ (refer to Figures 2.5 and 2.6), and consequently poorer generalisation performance would be expected.

In the limit as $C \to \infty$, no example is allowed to violate the constraints on margin. However, the training set is not linearly separable. When the solution tries to meet the constraints, the noisy example brings the corresponding α_i to the upper bound ∞, and some other α_i nearly to ∞ as well. This in turn leads to an infinit solution, as shown on the last row of Table 2.1.

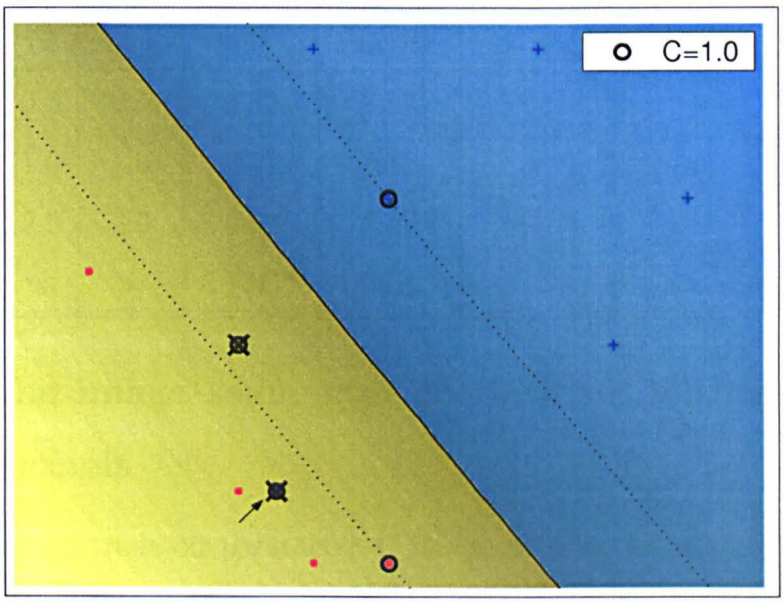


Figure 2.6: Example of a linearly non-separable training set ($C = 1.0$).

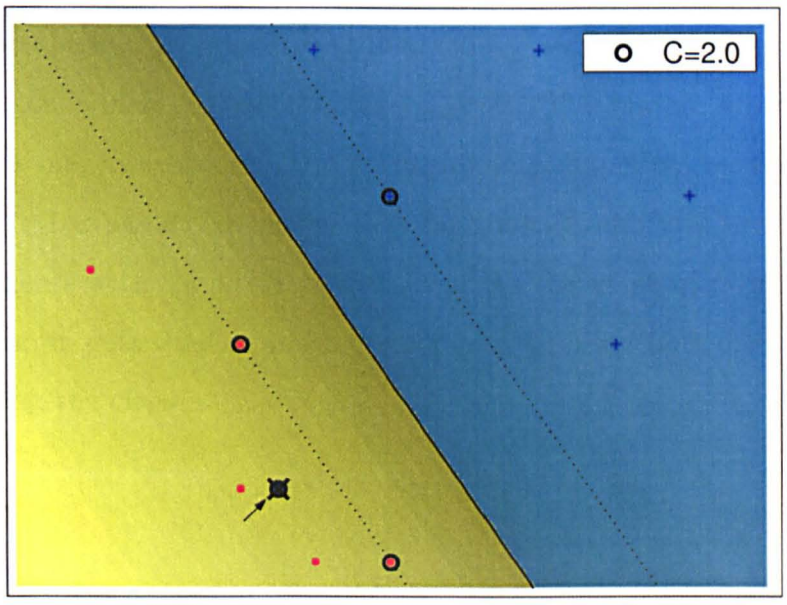


Figure 2.7: Example of a linearly non-separable training set ($C = 2.0$).

Table 2.1: Example of a linearly non-separable training set. Results are obtained with different values of trade-off parameter C.

C	α_{noise}	$\|\vec{w}\|^2$	margin	no. of margin errors
0.5	0.5	0.32	3.54	2
1.0	1.0	0.41	3.12	2
2.0	2.0	0.52	2.77	1
∞	∞	3.73E^{10}	1.03E^{-5}	8

2.5.2 The image segmentation data set with nonlinear Kernels

The effect of trade-off parameter C has also been investigated for a non-linear Kernel. The image segmentation data set [42] has been used for this investigation. The data set contains 7 classes of examples of 19 attributes. To allow the results to be visualised, two attributes have been used to classify a pair of two non-overlapped classes of 891 examples. Figures 2.8 - 2.12 show the decision boundaries obtained for different values of the trade-off parameter C. The two classes are respectively denoted by red '.'s and blue '+'s. Each support vector is identified with an extra circle. The margin errors are identified with a magenta square. The corresponding numerical results are listed in Table 2.2. The same conclusions may be drawn from these results, as after the experiments with the artificial data set. As the value of C increases, the width of margin gets smaller and more complex decision boundary is required for separating the classes.

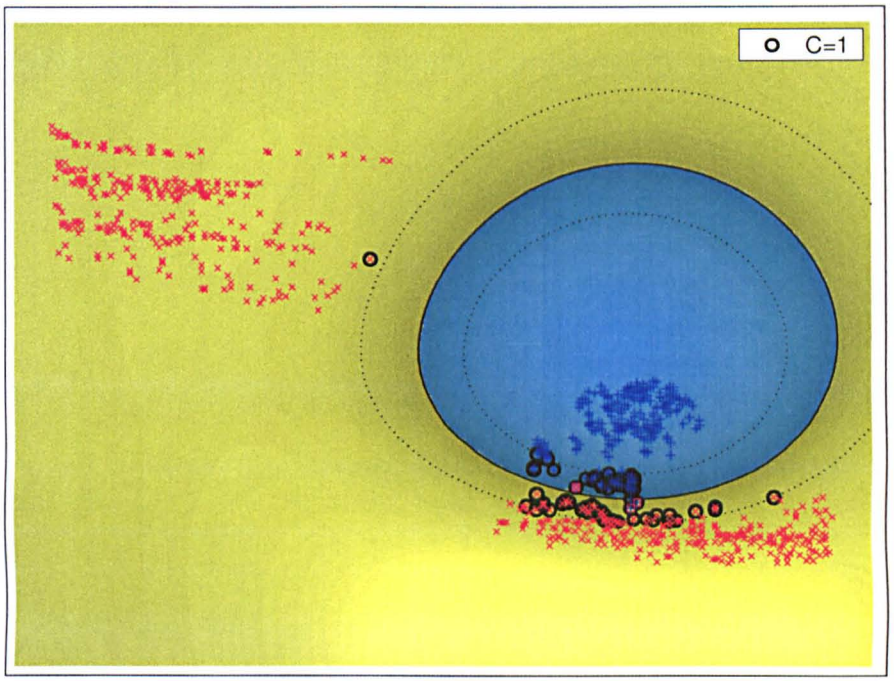


Figure 2.8: Example of image segmentation data set ($C = 1$).

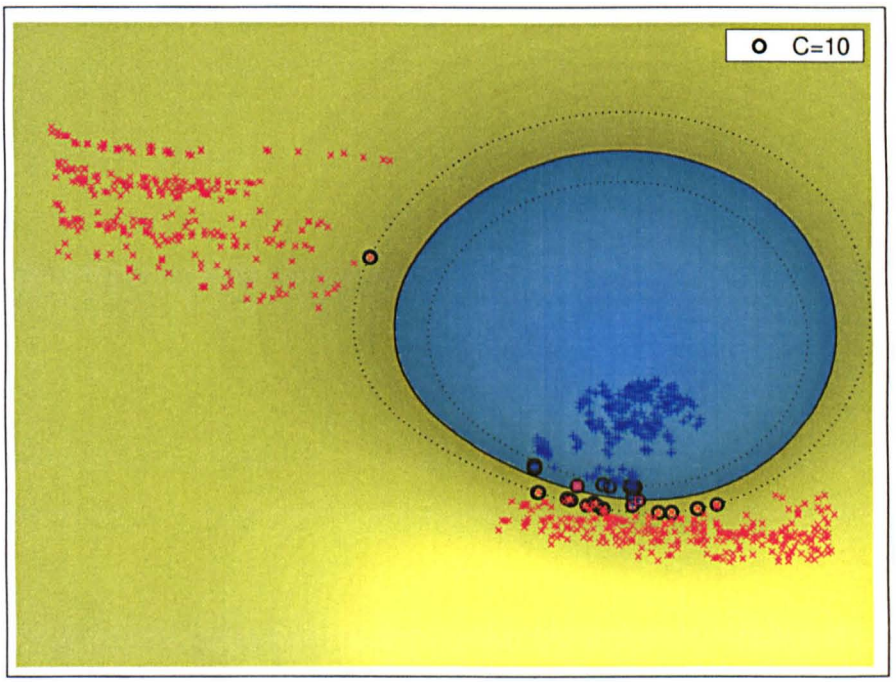


Figure 2.9: Example of image segmentation data set ($C = 10$).

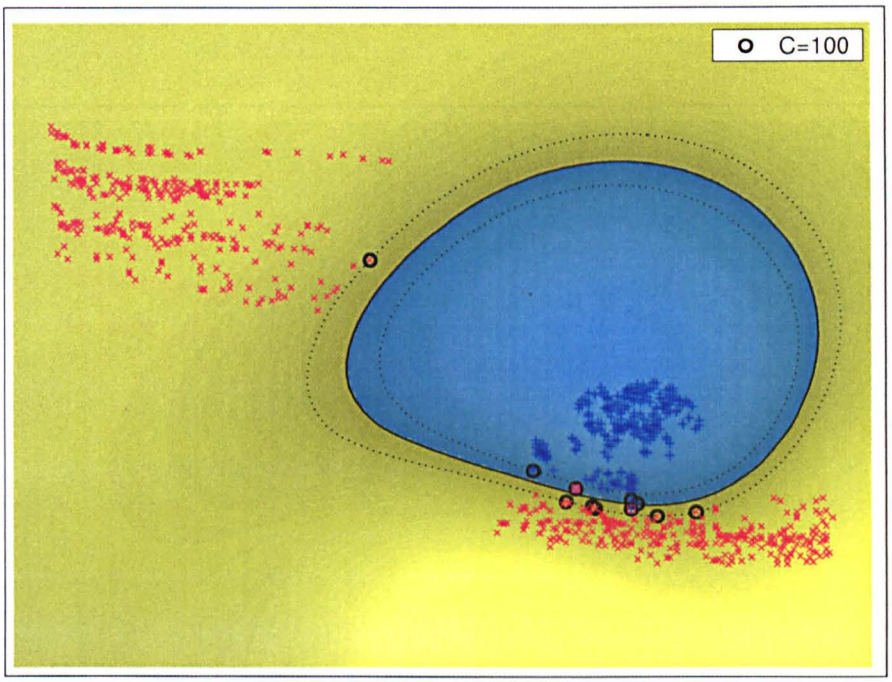


Figure 2.10: Example of image segmentation data set ($C = 100$).

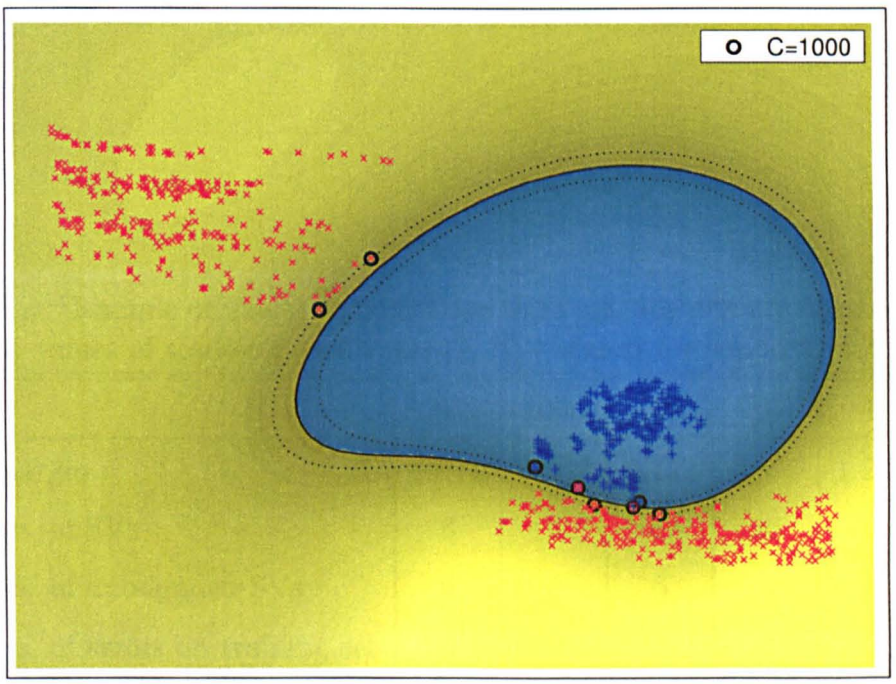


Figure 2.11: Example of image segmentation data set ($C = 1000$).

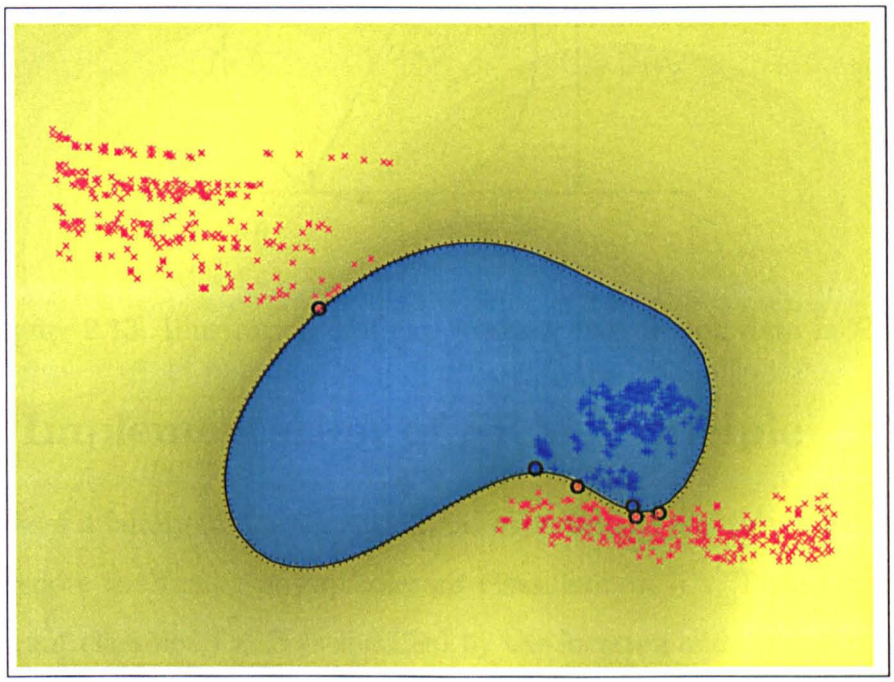


Figure 2.12: Example of image segmentation data set ($C = \infty$).

Table 2.2: Example of image segmentation data set. Results are obtained with different values of trade-off parameter C. SVs stands for support vectors.

C	∞	1000	100	10	1
margin	0.011	0.062	0.122	0.327	1.296
no. of SVs	6	8	11	26	80
no. of unbounded SVs	6	6	6	6	6
no. of errors on training set	0	2	2	3	3

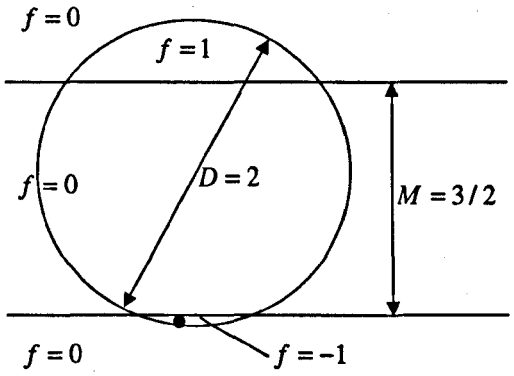


Figure 2.13: Illustration of a gap tolerant classifier on data in $\mathcal{R}^2$.

2.6 Implementation of SRM Principle

Consider a family of classifiers that has been termed *gap tolerant classifiers.* Let $\mathcal{F}$ denote the family of gap tolerant classifiers on $\mathcal{R}^d$. Then a particular gap tolerant classifier $f \in \mathcal{F}$ is specified by the location and diameter of a ball in $\mathcal{R}^d$, and by two hyperplanes, with parallel normals, also in $\mathcal{R}^d$. Call the set of points lying between, but not on, the hyperplanes the "margin set". The decision function f is defined as follows: points that lie inside the ball, but not in the margin set, are assigned class ± 1, depending on which side of the margin set they fall; all other points are simply defined to be "correct", that is, they are not assigned a class by the classifier, and do not contribute any risk. The situation is summarised, in Figure 2.13, for $d = 2$.

According to [36], the following Theorem holds for gap tolerant classifiers.

Theorem 2.1. For data in $\mathcal{R}^d$, the VC dimension h of the set of gap tolerant classifiers of minimum margin M_{min} is

$$h \leq \min\{D^2/M_{\text{min}}^2,\ d\} + 1, \tag{2.6.1}$$

where D is the diameter of a hypersphere enclosing all the data points.

For example, for the gap tolerant classifier in Figure 2.13, $D = 2, d = 2$

and hence VC dimension is $h = 3$ when $M \leq \sqrt{2}$ ($M_{\min} = 0$), $h = 2$ when $\sqrt{2} < M \leq 2$ ($M_{\min} > \sqrt{2}$), and $h = 1$ when $M > 2$ ($M_{\min} > 2$).

The gap tolerant classifier is in fact a special kind of SVM which simply does not count data falling outside the sphere containing all the training data, or inside the separating margin, as an error. Thus, equation (2.6.1) may also be applied to SVMs.

Recall that $\frac{1}{\text{margin}} = \frac{\|\vec{w}\|^2}{2}$. Hence from equation (2.6.1), maximising margin or minimising equation (2.3.4) is equivalent to minimising an upper bound on the VC dimension of a SVM and hence equivalent to implementing the SRM principle. The optimisation problem OP4 is posed so as to minimise the margin errors (i.e. the second term of equation (2.4.1)) as well as an upper bound on the VC dimension of the classifier (i.e. the first term of equation (2.4.1)).

2.7 A Mechanical Analogy

Suppose that the ith support vector exerts a force $\text{Forces}_i = \alpha_i y_i \vec{w}_0$ on the stiff sheet along the surface of the decision boundary (or decision sheet) in the feature space. Here $\vec{w}_0$ denotes the unit vector in the direction $\vec{w}$. Then according to equations (2.3.8) and (2.3.7) respectively, a resulting SVM satisfies the following conditions of mechanical equilibrium [43]

$$\sum \text{Forces}_i = \sum_i \alpha_i y_i \vec{w}_0 = \left(\sum_i \alpha_i y_i\right) \vec{w}_0 = 0, \tag{2.7.1}$$

$$\sum \text{Torques}_i = \sum_i \vec{s}_i \cdot (\alpha_i y_i \vec{w}_0) = \vec{w} \cdot \vec{w}_0 = 0. \tag{2.7.2}$$

In equation (2.7.2), $\vec{s}_i$ denotes the ith support vector and '$\cdot$' the inner product of two vectors.

This mechanical analogy depends only on equations (2.3.7) and (2.3.8) and therefore holds for both hard- and soft-margin SVMs. This analogy emphasises

the interesting point that the most important data points are the support vectors with highest values of α_i, since they exert the highest forces onto the decision sheet. For non-separable case, the upper bound C of α_i corresponds to an upper bound on the force any given point is allowed to exert onto the sheet. This analogy also provides a reason, as good as any other, to call these particular vectors the support vectors.

Recall the linearly nonseparable example in subsection 2.5. For different values of C, the orientations of the separating hyperplane are different. Figure 2.14 compares the decision boundaries obtained for $C = 0.5$, $C = 1.0$ and $C = 2.0$. And Table 2.3 gives, for each case of C, the amount of forces exerted by the four common support vectors (for $C = 0.5$, there is one more support vector). For all the cases, the force exerted by the noisy point (P_4) onto the decision sheet is bounded by $-C\vec{w}_0$. The last four rows of Table 2.3 are the forces exerted by each support vector divided by the force exerted by P_4 along the y-axis. As C increases, more and more amounts of force are exerted onto the decision sheet by P_2 while P_3 exerts less and less. The amount of force exerted by either P_1 or P_4 remains almost the same. As a result, the orientation of the decision boundary changes in the way as shown in Figure 2.14. And when C reaches 2.0, α_2 becomes smaller than C; the amount of force need to be exerted by P_2 for the equilibrium is no longer limited by the upper bound. P_2 is no longer a margin error. Yet it still lies on the margin boundary as a support vector. For higher C's, the same decision boundary is obtained as that for $C = 2.0$.

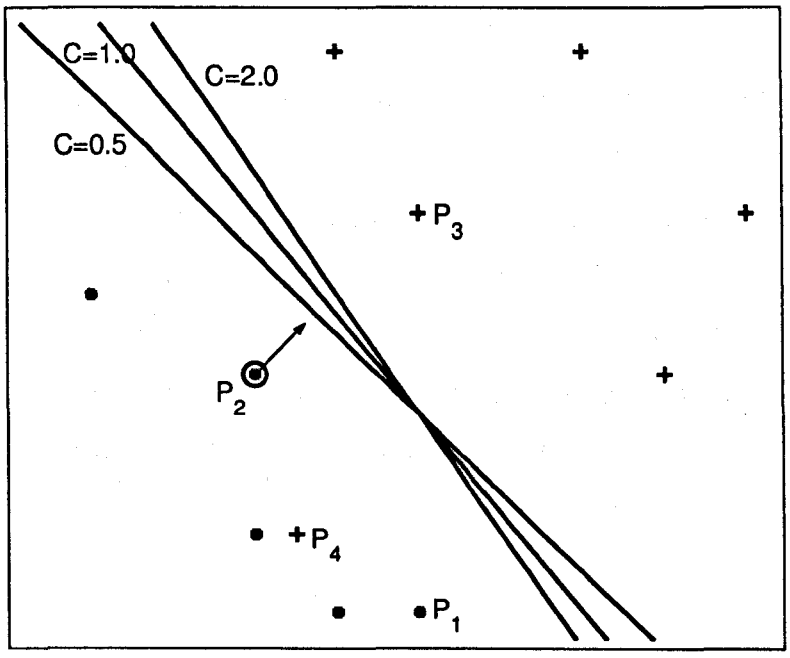


Figure 2.14: Comparison of decision boundaries obtained on a linearly nonseparable training set for different values of C.

Table 2.3: Comparison of SVM results obtained on a linearly nonseparable set for different values of C.

C		0.5	1.0	2.0	10	100
α_i	P_1	0.2725	0.4800	0.9600	4.9600	49.9600
	P_2	0.5000	1.0000	1.8000	7.8000	75.3000
	P_3	0.3100	0.4800	0.7600	2.7600	25.2600
	P_4	0.5000	1.0000	2.0000	10.000	100.000
$\vec{w}$		-0.4, -0.4	-0.5, -0.4	-0.6, -0.4	-0.6, -0.4	-0.6, -0.4
Force_i	P_1	-0.19, -0.19	-0.37, -0.30	-0.80, -0.53	-4.13, -2.75	-41.57, -27.71
	P_2	-0.35, -0.35	-0.78, -0.62	-1.50, -1.00	-6.49, -4.33	-62.65, -41.77
	P_3	0.22, 0.22	0.37, 0.30	0.63, 0.42	2.30, 1.53	21.02, 14.01
	P_4	0.35, 0.35	0.78, 0.62	1.66, 1.11	8.32, 5.55	83.21, 55.47
Force_i'	P_1	-0.55, -0.55	-0.60, -0.48	-0.72, -0.48	-0.74, -0.50	-0.75, -0.50
	P_2	-1.00, -1.00	-1.25, -1.00	-1.35, -0.90	-1.17, -0.78	-1.13, -0.75
	P_3	0.62, 0.62	0.60, 0.48	0.57, 0.38	0.41, 0.28	0.38, 0.25
	P_4	1.00, 1.00	1.25, 1.00	1.50, 1.00	1.50, 1.00	1.50, 1.00

2.8 Conclusions

The family of support vector machines has been introduced in this chapter. It is characterised by the use of kernel functions, the sparseness of solutions and the capacity control obtained by acting on the margin. These facts of SVMs mark a clear distinction between these systems and other pattern recognition algorithms. And more importantly, the implementation of SRM principle makes SVMs superior to the others that employ Empirical Risk Minimisation (ERM) principle during the training, such as neural networks.

Chapter 3

KERNEL-INDUCED FEATURE SPACE

As stated, the mapping $\Phi : \mathcal{R}^d \mapsto \mathcal{F}$ of nonlinear SVMs maps data from the input space $\mathcal{R}^d$ into a potentially high-dimensional feature space $\mathcal{F}$. And the training and using of a SVM only depends on the data through dot products in $\mathcal{F}$. This chapter discusses the method that can be used to construct the mapping by the use of reproducing kernels. The idea of using kernel functions is to enable operations to be performed in the input space rather than the high-dimensional feature space. Hence the inner product does not need to be evaluated in the feature space. This provides a way of addressing the curse of dimensionality.

3.1 Mercer's Theorem

For which kernels does there exist a pair $\{\mathcal{F}, \Phi\}$, with the properties described above, and for which does there not? The answer is given by Mercer's condition [7, 44], based upon Reproducing Kernel Hilbert Spaces (RKHS) [45, 46, 47].

Mercer's condition: If the K is a symmetric and positive definite function, which satisfies

$$K(\vec{x},\vec{y}) = \sum_{i=1}^{\infty} \alpha_i \, \phi_i(\vec{x}) \, \phi_i(\vec{y}), \qquad \text{where } \alpha_i \geq 0 \tag{3.1.1}$$

$$\text{and } \iint K(\vec{x},\vec{y}) \, g(\vec{x}) \, g(\vec{y}) \, d\vec{x} \, d\vec{y} > 0, \qquad \text{where } \int g^2(\vec{x}) \, d\vec{x} < \infty, \tag{3.1.2}$$

then the kernel function K represents a legitimate inner product in the feature space defined by $\Phi(\vec{x}) = \{\phi_1(\vec{x}), \ldots, \phi_i(\vec{x}), \ldots\}$.

3.2 Kernel Functions for Support Vector Machines

This section gives some examples of Kernel functions for SVMs, which unless stated are valid for all real $\vec{x}$ and $\vec{y}$.

Polynomial

Polynomial mapping is a popular method for non-linear modelling. A polynomial kernel of degree p is defined as

$$K(\vec{x},\vec{y}) = ((\vec{x} \cdot \vec{y}) + 1)^p . \tag{3.2.1}$$

The corresponding feature space $\mathcal{F}$ is a Euclidean space of dimension $\binom{d+p-1}{p}$, where d is the dimension of the input space.

Gaussian radial basis function

Radial basis functions have received significant attention, most commonly with a Gaussian of the form,

$$K(\vec{x},\vec{y}) = \exp\left(-\frac{(\vec{x}-\vec{y})^2}{2\sigma^2}\right), \tag{3.2.2}$$

Exponential radial basis function

A radial basis function of the form,

$$K(\vec{x},\vec{y}) = \exp\left(-\frac{\|\vec{x}-\vec{y}\|}{2\sigma^2}\right), \tag{3.2.3}$$

produces a piecewise linear solution which can be attractive when discontinuities are acceptable.

The radial basis function kernels are translation invariant. Moreover, as each of them satisfies $K(\vec{x},\vec{x}) = 1$ for all $\vec{x} \in \mathcal{R}^d$, each mapped example has unit length, $\|\Phi(\vec{x})\| = 1$. In addition, for radial basis function SVMs, the number of centres (the number of support vectors), the centres themselves (the support vectors), the weights (α_i) and the threshold (b) are all produced *automatically* by the SVM training and give excellent results compared to classical radial basis functions, for the case of Gaussian radial basis functions [48].

Multi-layer perceptron

The long-established multi-layer perceptron, with a single hidden layer, also has a valid kernel representation,

$$K(\vec{x},\vec{y}) = \tanh(scale \cdot (\vec{x}\cdot\vec{y}) - offset) \tag{3.2.4}$$

for certain values of the *scale* and *offset* parameters (first noticed experimentally [7]). Here the support vectors correspond to the nodes on the first layer and the Lagrange multipliers to the weights. Thus, the architecture (number of weights) is determined automatically by SVM training.

Fourier series

A Fourier series can be considered an expansion in the following $2N+1$ dimensional feature space. The kernel is defined on the interval $[-\frac{\pi}{2},\frac{\pi}{2}]$,

$$K(\vec{x},\vec{y}) = \frac{\sin(N+1/2)(\vec{x}-\vec{y})}{2\sin((\vec{x}-\vec{y})/2)}. \tag{3.2.5}$$

Linear spline

Splines are a popular choice for modelling due to their flexibility. Linear spline

is defined as

$$K(x_i, y_j) = 1 + x_i y_j + x_i y_j \min(x_i, y_j) - \frac{(x_i + y_j)}{2} (\min(x_i, y_j))^2 + \frac{1}{3} (\min(x_i, y_j))^3, \quad \forall i = 1, \ldots, d. \tag{3.2.6}$$

Additive kernels

More complicated kernels can be obtained by forming summing kernels, since the sum of two positive definite functions is still positive definite.

$$K(\vec{x}, \vec{y}) = \sum_i K_i(\vec{x}, \vec{y}). \tag{3.2.7}$$

Tensor product kernels

Multidimensional kernels can be obtained by forming tensor products of kernels [45], where

$$K(\vec{x}, \vec{y}) = \Pi_{m=1}^{n} K_m(\vec{x}_m, \vec{y}_m). \tag{3.2.8}$$

3.3 Some Notes on Φ and $\mathcal{F}$

Polynomial kernel of degree p constructs a feature space of dimension $\binom{d+p-1}{p}$. While the feature space defined by a radial basis function kernel has an infinite dimension. Usually, mapping the data to a "feature space" with an enormous number of dimensions would bode ill for the generalisation performance of the resulting machine. After all, the set of all hyperplanes $\{\vec{w}, b\}$ in a feature space $\mathcal{F}$ are parameterised by $\dim(\mathcal{F}) + 1$ numbers. Most pattern recognition systems with billions, or even an infinite, number of parameters would not make it past the start gate due to the curse of dimensionality. How come SVMs do so well? One might argue that, given the form of solution and the size of training set l, there are at most $l + 1$ adjustable parameters in a SVM. However, the real reason lies in the use of maximum margin as described in section 2.6.

As for the enormous calculational load of an infinite-dimensional space, it has been avoided by the use of kernels. The following uses the example of a Fourier kernel to show that the inner product of two vectors in an infinite-dimensional space can be calculated in closed form. Suppose $x \in \mathcal{R}^1$. Then a Fourier expansion in the data x, cut off after N terms, has the form

$$f(x) = \frac{a_0}{2} + \sum_{r=1}^{N} (a_{1r}\cos(rx) + a_{2r}\sin(rx)). \tag{3.3.1}$$

This can be viewed as a dot product of two vectors in $\mathcal{R}^{2N+1}$:

$$\vec{a} = \{a_0/\sqrt{2}, a_{11}, \ldots, a_{21}, \ldots\}$$

$$\text{and the mapped } \Phi(x) = \{1/\sqrt{2}, \cos(x), \cos(2x), \ldots, \sin(x), \sin(2x), \ldots\}.$$

Then letting $\delta = x_i - x_j$,

$$\begin{aligned}
\Phi(x_i)\cdot\Phi(x_j) &= \frac{1}{2} + \sum_{r=1}^{N} \cos(rx_i)\cos(rx_j) + \sin(rx_i)\sin(rx_j) \\
&= -\frac{1}{2} + \sum_{r=0}^{N} \cos(r\delta) = -\frac{1}{2} + Re\{\sum_{r=0}^{N} e^{(ir\delta)}\} \\
&= -\frac{1}{2} + Re\{(1 - e^{i(N+1)\delta})/(1 - e^{i\delta})\} \\
&= \frac{\sin((N+1/2)\delta)}{2\sin(\delta/2)}.
\end{aligned} \tag{3.3.2}$$

Hence, the kernel can be computed in closed form by equation (3.3.2) in terms of x_i and x_j.

3.4 Kernel Selection

The obvious question that arises is that with so many different mappings to choose from, which is the best for a particular problem? This is not a new question, but with the inclusion of many mappings within one framework it is easier to make a comparison. The rest of this section illustrates the effect

Table 3.1: Summary of statistics of iris data set

	Min	Max	Mean	Standard Deviation	Class Correlation
sepal length:	4.3	7.9	5.84	0.83	0.7826
sepal width:	2.0	4.4	3.05	0.43	-0.4194
petal length:	1.0	6.9	3.76	1.76	0.9490
petal width:	0.1	2.5	1.20	0.76	0.9565

of different kernels by comparing the experimental results on a benchmark problem - the classification of iris data.

The iris data set is a benchmark data set used for demonstrating the performance of classification algorithms. The data set contains three classes of iris examples. Each example is defined by four attributes. The goal is to classify the class of iris based on these four attributes. The summary statistics of iris data set is listed in Table 3.1. The last column shows the correlation coefficient between each attribute and the class label. The higher the correlation coefficient is the more classification information an attribute contains. To visualise the problem we restrict ourselves to the two attributes that contain the most information about the classes, namely the petal length and the petal width. According to these two attributes, the distribution of iris data set is illustrated in Figure 3.1. The Versilcolor example that overlaps with a Virginica example (according to petal length and petal width) has been removed.

The Setosa and Versilcolor classes are easily separated with a linear boundary and the resulting decision boundary using a linear kernel is illustrated in Figure 3.2, with the two support vectors circled. Figure 3.3 shows the results for the same pattern recognition problem, but where the kernel was chosen to be a polynomial of degree 2. Notice that, even though the number of degrees of freedom is higher, for the linearly separable case, the solution given by the

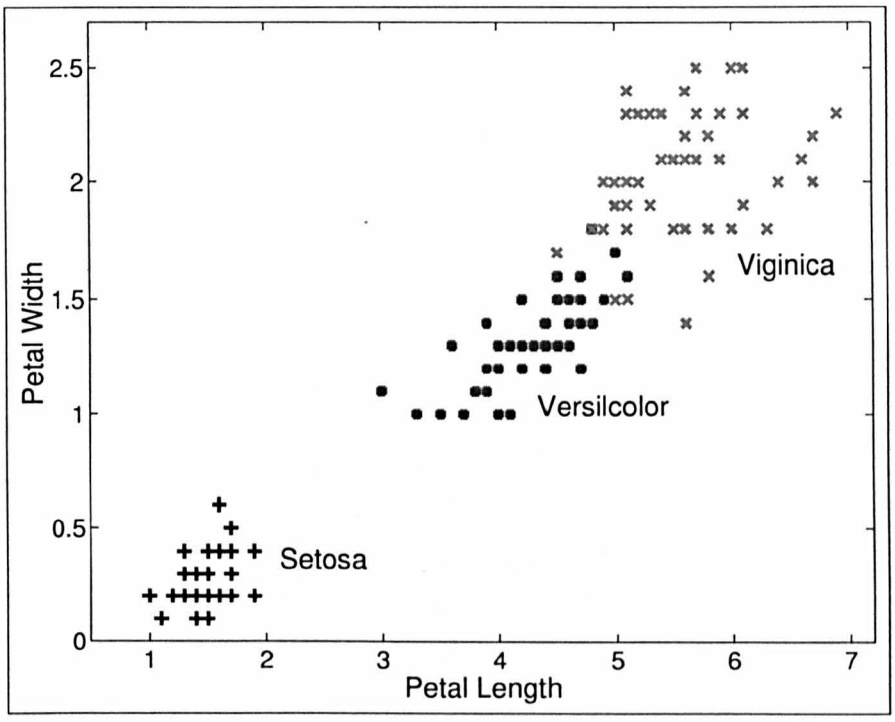


Figure 3.1: Iris data set with two attributes.

polynomial SVM is roughly linear, indicating that the capacity is being well controlled.

It is obvious that Virginica is not linearly separable. A nonlinear decision boundary is expected and thus a nonlinear kernel is needed to separate the classes completely. Different kernels have been tried for separating Virginica from Versilcolor plus Setosa. Figures 3.4 - 3.10 show the results, accordingly. And table 3.2 gives the width of the margin and the number of support vectors.

The results given by the 2nd-degree polynomial SVM is not satisfying. The margin is too small. As higher degree polynomials are used, the result becomes worse. Figure 3.5 shows the result of 5th-degree polynomial. As we can see, the margin becomes extremely small. And there is evidence of overfitting due to the high dimensional nature of the kernel, which is emphasised by the unnecessarily large number of support vectors. Furthermore, note that the decision boundaries given by both polynomial SVMs contain a disjoint region

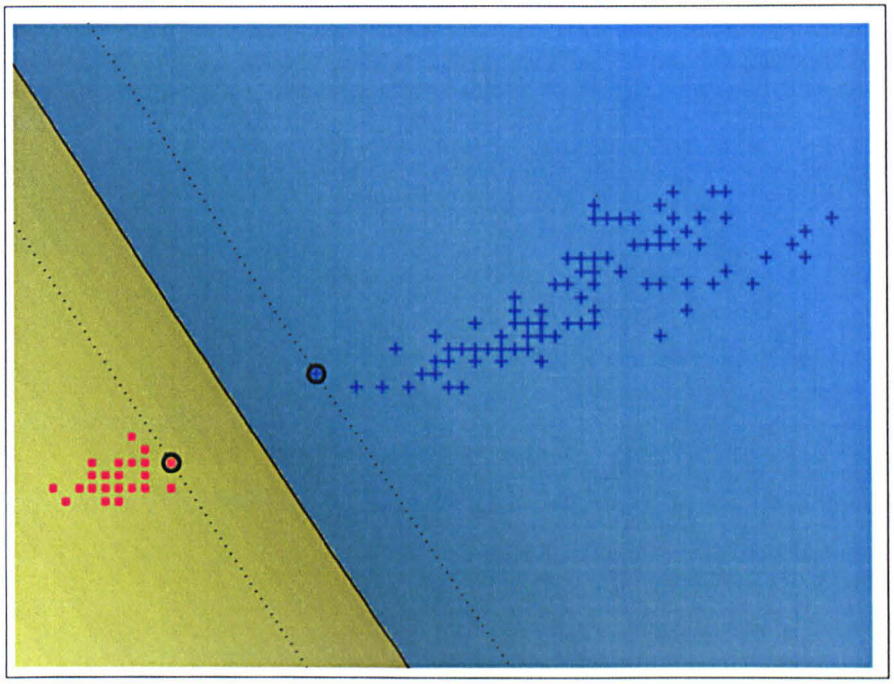


Figure 3.2: Separating Setosa with a linear SVM ($C = \infty$).

in the top of the illustration, which is another evidence of overfitting.

Figures 3.6 - 3.9 show the results given by the radial basis function SVMs with different pre-specified variances. As we can see, more complicated boundaries have been provided by radial basis function kernels. And roughly same set of support vectors has been found with different variances. However, smaller the variance allows more shape bending. Amazingly, when variance is set to 0.6, the resulting decision boundary separates the class of Setosa from Versilcolor and identifies Setosa as a separate class.

Finally, a linear spline SVM has been used. The resulting decision boundary is similar to that with a high variance radial basis function. However, a disjoint region is observed.

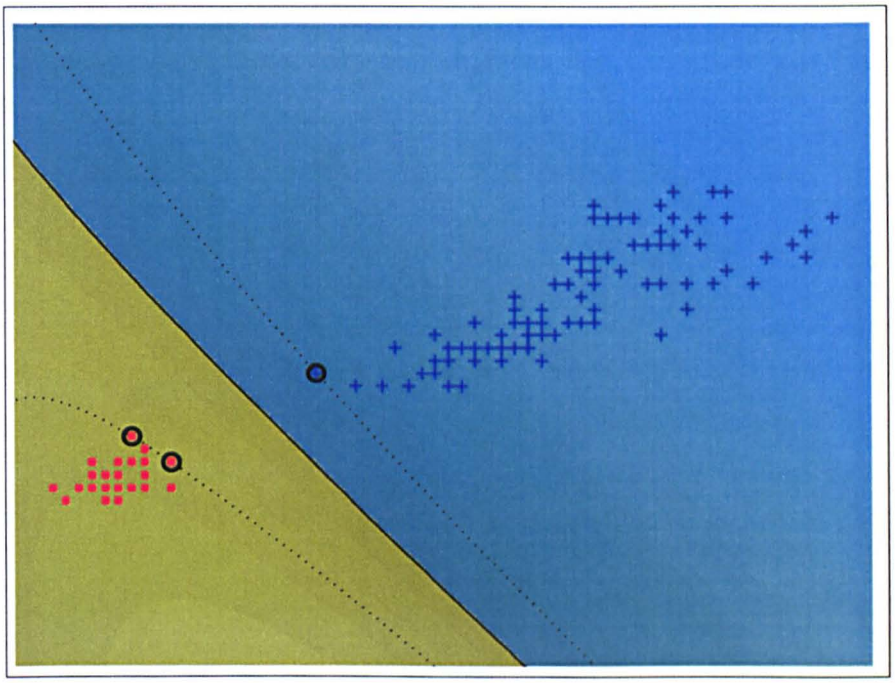


Figure 3.3: Separating Setosa with a polynomial SVM (degree 2, $C = \infty$).

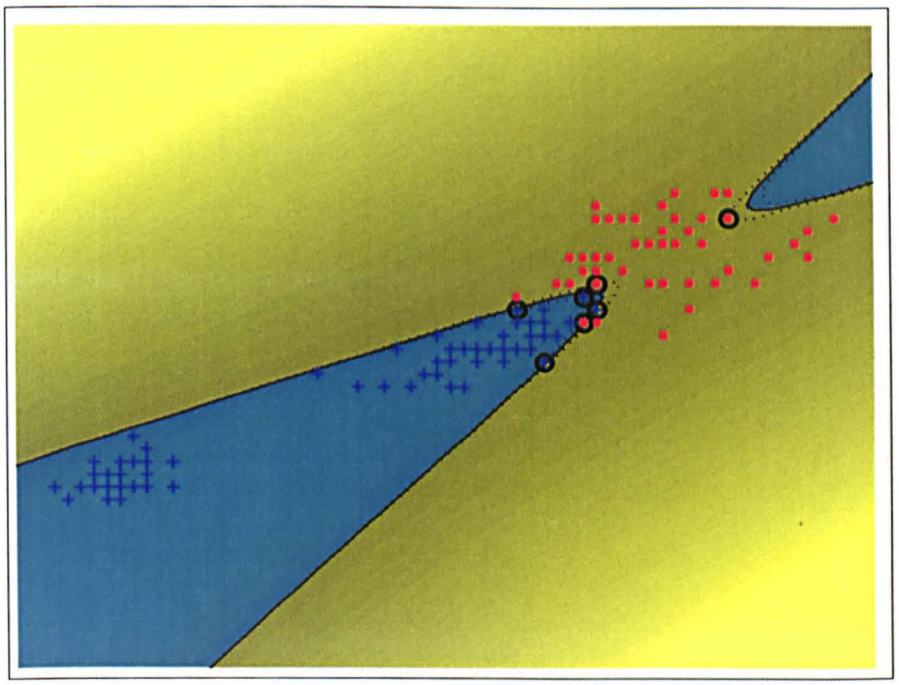


Figure 3.4: Separating Virginica with a polynomial SVM (degree 2, $C = \infty$).

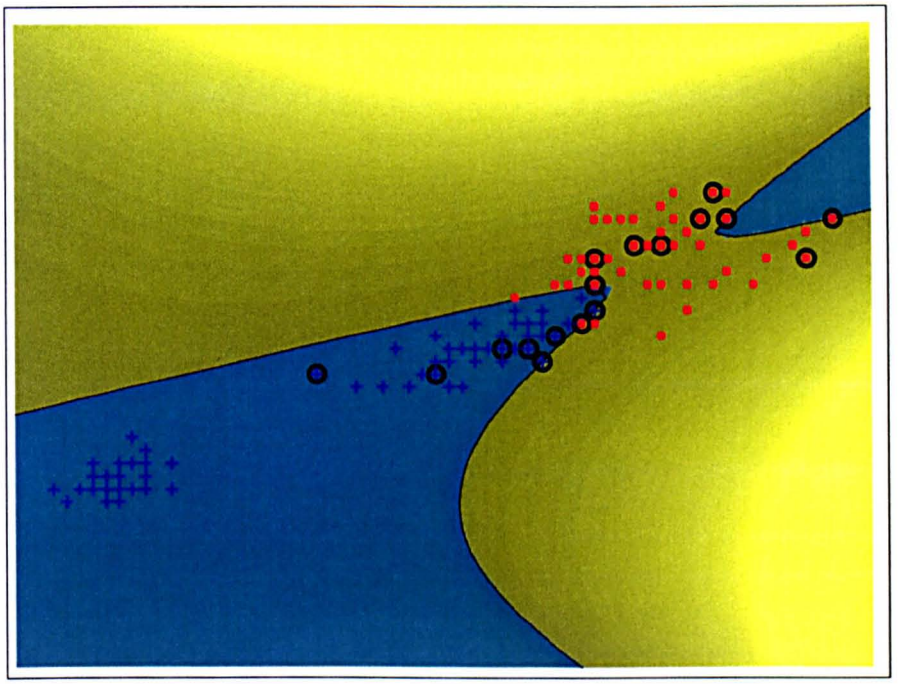


Figure 3.5: Separating Virginica with a polynomial SVM (degree 5, $C = \infty$).

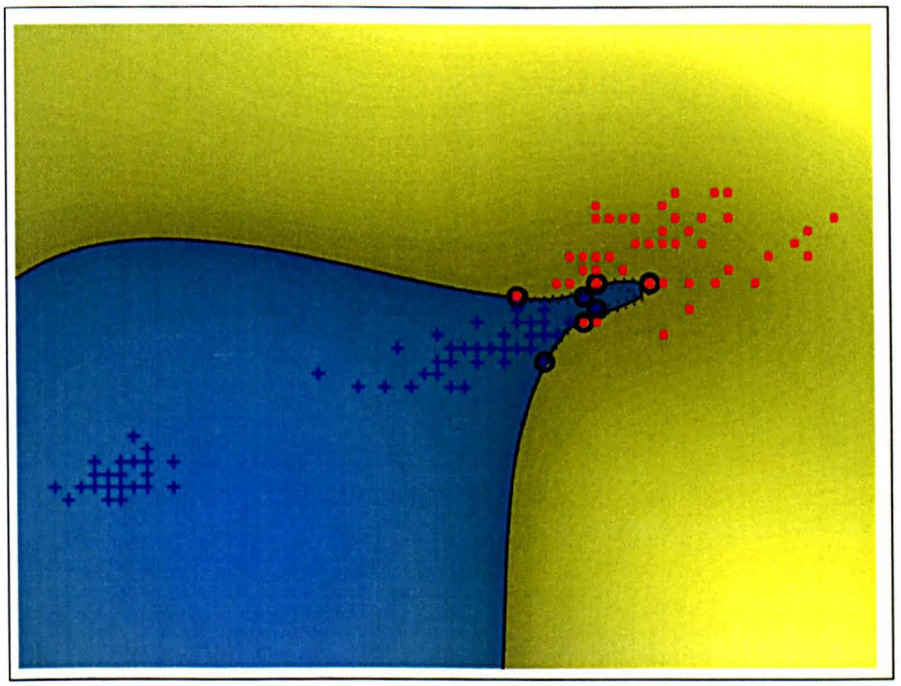


Figure 3.6: Separating Virginica with a Gaussian radial basis function SVM ($\sigma = 2.0$, $C = \infty$).

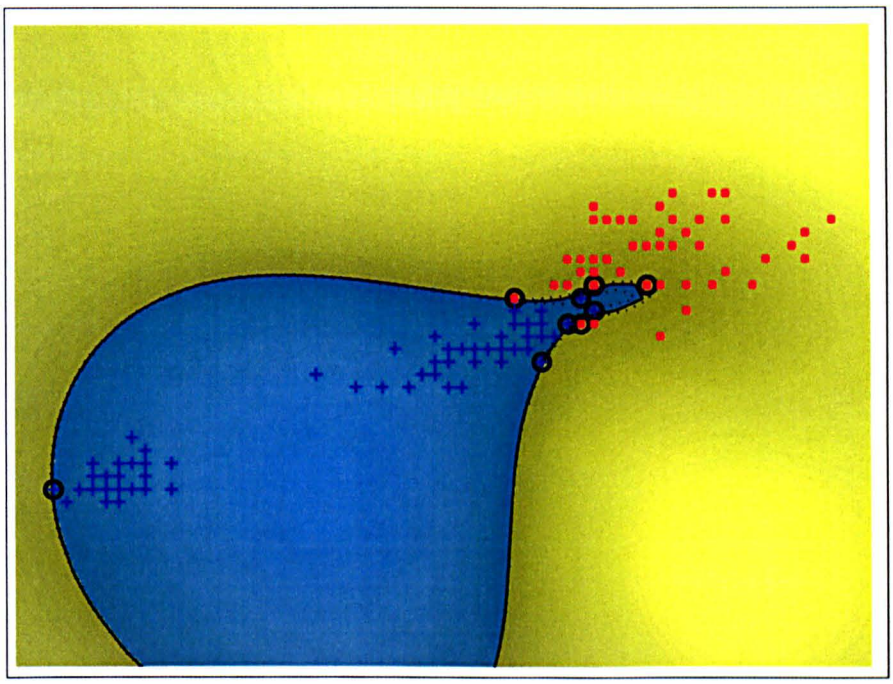


Figure 3.7: Separating Virginica with a Gaussian radial basis function SVM ($\sigma = 1.5,\ C = \infty$).

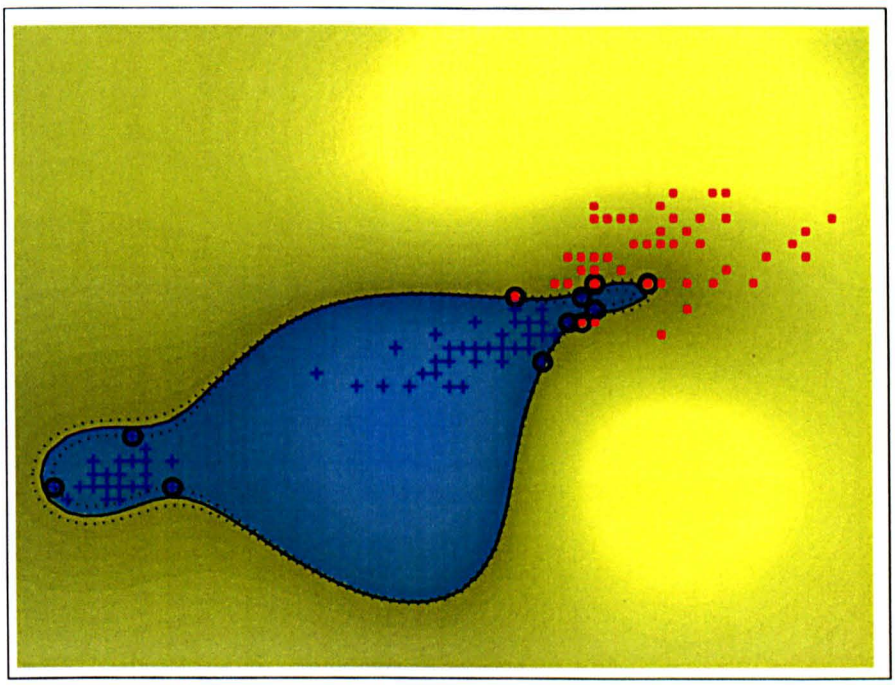


Figure 3.8: Separating Virginica with a Gaussian radial basis function SVM ($\sigma = 1.0,\ C = \infty$).

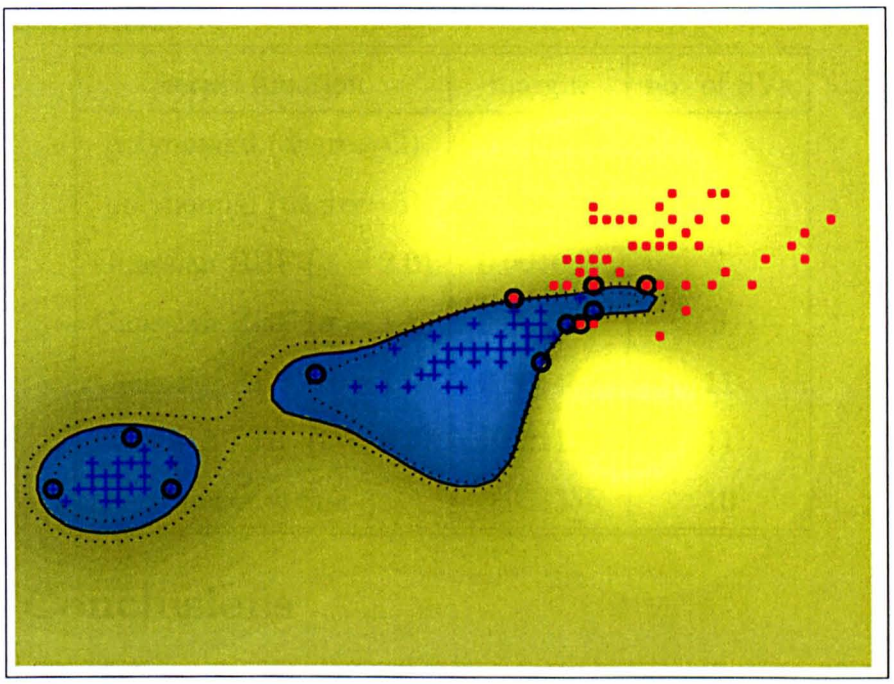


Figure 3.9: Separating Virginica with a Gaussian radial basis function SVM ($\sigma = 0.6,\ C = \infty$).

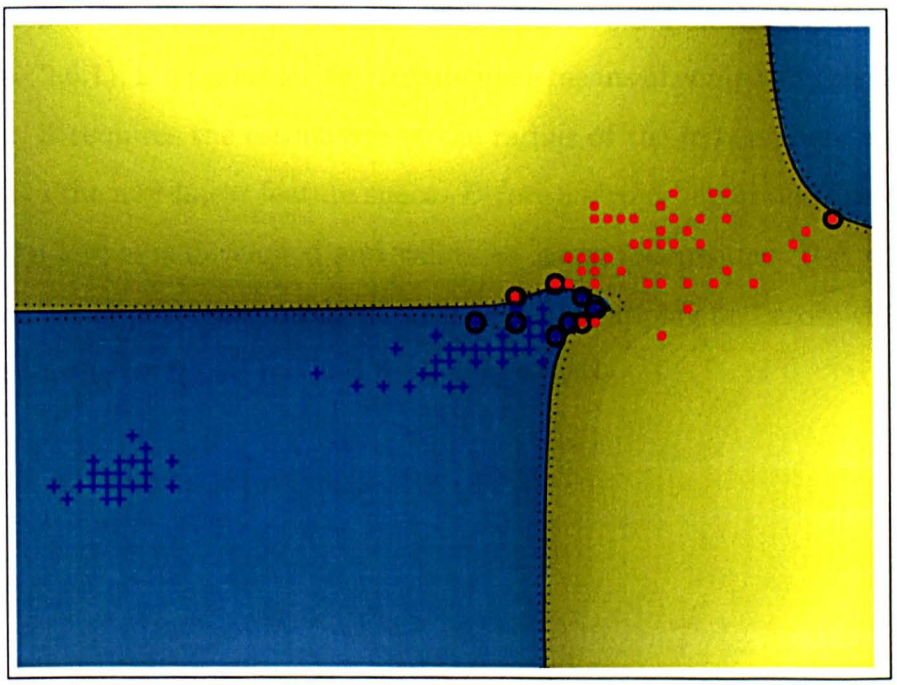


Figure 3.10: Separating Virginica with a linear spline SVM ($C = \infty$).

Table 3.2: Results when separating Virginica with different kernels. SVs stands for support vectors.

kernel function	margin	no. of SVs
polynomial (degree=2)	$1.200e^{-5}$	7
polynomial (degree=5)	$1.0085e^{-35}$	17
Gaussian RBF ($\sigma = 2.0$)	0.001085	7
Gaussian RBF ($\sigma = 1.5$)	0.002345	9
Gaussian RBF ($\sigma = 1.0$)	0.006462	11
Gaussian RBF ($\sigma = 0.6$)	0.021119	11
linear spline	0.038355	10

3.5 Conclusions

Mercer's condition tells us whether or not a prospective kernel is actually a dot product in some space. Examples of kernels that satisfy Mercer's condition have been given. The issue of how to choose an appropriate kernel for a particular classification problem is difficult. The upper bound on the VC dimension, equation (2.6.1), is a potential way to provide a means of comparing the kernels. However, it requires the estimation of the radius of the hypersphere enclosing the data in the non-linear feature space. Before a strong theoretical method for selecting a kernel is developed and validated using independent test sets on a large number of problems, methods such as bootstrapping and cross-validation will remain the preferred method for kernel selection.

Chapter 4

SUPPORT VECTOR MACHINE TRAINING AND ITS IMPLEMENTATION ALGORITHMS

4.1 Introduction

Problem OP5 in Chapter 2 (equations (2.4.3) - (2.4.5)) defines the optimisation problem associated with the training of a SVM. This problem is actually a quadratic programming (QP) problem since it has the form

$$\min_{\vec{\alpha}} \quad -\vec{\alpha}^T \vec{1} + \frac{1}{2}\vec{\alpha}^T Q \vec{\alpha}, \tag{4.1.1}$$

$$\text{subject to} \quad \vec{\alpha}^T \vec{y} = 0, \tag{4.1.2}$$

$$0 \leq \vec{\alpha} \leq C\vec{1}, \tag{4.1.3}$$

where the quadratic term Q in SVM training is a positive semidefinite matrix and

$$Q_{ij} = y_i y_j K(\vec{x}_i, \vec{x}_j), \ i, j = 1, \ldots, l. \tag{4.1.4}$$

As the objective function (4.1.1) is convex every (local) maximum is already a global maximum. However, there can be several optimal solutions (in terms of the variables α_i) when matrix Q is not positive definite, i.e. when objective function is not strictly convex. For example, consider the problem of four separable points on a square in $\mathcal{R}^2$: $\vec{x}_1 = [1,1]$, $\vec{x}_2 = [-1,1]$, $\vec{x}_3 = [-1,-1]$, and $\vec{x}_4 = [1,-1]$, with polarities $[+,-,-,+]$ respectively. One solution is $\vec{w} = [1,0], b = 0, \vec{\alpha} = [0.25, 0.25, 0.25, 0.25]$; another has the same $\vec{w}$ and b, but $\vec{\alpha} = [0.5, 0.5, 0, 0]$. Note that both solutions satisfy the constraints $\alpha_i \geq 0$ and $\sum_i \alpha_i y_i = 0$.

The problem of optimising a quadratic function of many variables has been widely studied. The book [49] discusses general algorithms and techniques for convex optimisation. Early implementations of SVMs have been based on optimisation packages such as MINO [50], LOQO [51], MATLAB optimisation package [52], etc. Chapter 1 of the book [34] contains a useful survey of different implementations. However most of the standard QP techniques require full storage of the quadratic term in the objective function. They are either suitable only for small problems or assume that the quadratic term is very sparse, i.e. most elements of this matrix are zero. Unfortunately this is not true for the SVM optimisation problem, where quadratic term Q is not only dense but also has a size of l^2, i.e. a size growing quadratically with the number of data points in the training set. For training tasks with 10,000 examples and more, the memory requirement will exceed hundreds of Megabytes and the training time will be enormous. These facts prohibit the application of standard QP techniques to the problems with large training sets and, on the other hand, have urged the design of a number of algorithms for fast SVM training.

4.2 General Considerations

The size of matrix Q, the structure of it, the nature of the solution, and the structure of the constraints all should be considered when choosing or developing a QP programming package to solve equations (4.1.1) - (4.1.3). Underneath lists a few properties of some of the problems we have seen which have influenced our thinking.

1. For postprocessing it is important to determine which elements of the solution are 0 or at the upper bound C and which are not. Some QP packages, particularly interior point implementations, may return values close to the machine precision rather than exact zero. This may lead to a higher detected number of support vectors because "small" values were misinterpreted.

2. The constraints are upper and lower bound constraints plus one general equality constraint. Codes that do not make a special provision for bound constraints are probably not useful.

3. For most problems most elements of the solution $\vec{\alpha}$ are 0. There will be fewer changes in the solution if one begins with a "zero" solution. And in the next section, we will see this sparseness feature of the solution of SVM optimisation has inspired a series of fast algorithms for SVM training.

4. When the upper bound C in equation (4.1.3) is small, the number of nonzero elements in $\vec{\alpha}$ increases, but many of them, will be at C. Then when doing vector-matrix multiplication $\vec{\alpha}Q$ one needs only the sum of the columns of the Q matrix corresponding to the elements of $\vec{\alpha}$ that equal C, not the elements themselves. For example,

$$\begin{aligned} &\text{if} && \vec{\alpha} = (0, C, C, \alpha_4, C, 0, \ldots, 0) \\ &\text{then} && \vec{\alpha}Q = \alpha_4 \vec{q}_4 + C(\vec{q}_2 + \vec{q}_3 + \vec{q}_5). \end{aligned}$$

It will not be unusual if on subsequent iterations, the 2nd, 3rd and 5th elements of $\vec{\alpha}$ will still be at C. Thus if one keep a running total of the sum of the columns of Q at bound, then one will save work on the vector-matrix multiplication and also require less space.

4.3 Present Fast Algorithms for SVM Training

This section introduces briefly three different algorithms for fast SVM training, including Chunking, Decomposition and Sequential Minimal Optimisation (SMO). They are in turn derived from the former algorithm.

4.3.1 Chunking

As discussed in last section, a key observation in solving large-scale SVM problems is the sparseness of the solution $\vec{\alpha}$. Depending on the problem, many of the optimal Lagrange multipliers α_i^* will either be zero or on the upper bound C. If one knew beforehand which α_i^* were zero, the corresponding rows and columns could be removed from the matrix Q without changing the value of the objective function. Furthermore, a particular $\vec{\alpha}$ can only be optimal for OP5 if and only if it satisfies the KKT conditions (equations (2.4.8) - (2.4.10)). In [36], a method called chunking is described, making use of the solution sparseness and the KKT conditions. It starts with an arbitrary subset or 'chunk' of the data, and trains an SVM using a generic optimiser on that portion of the data. The algorithm then retains the support vectors from the chunk while discarding the other points and then it uses a particular hypothesis to test and choose the points in remaining part of the data. In the original work reported in [36] the M points that most violate the KKT conditions (where M is a parameter of the system) are chosen to add to the support vectors of the previous training, to form a new chunk for next iteration. The chunk of data being optimised

at each stage varies but is finally equal to the number of nonzero Lagrange multipliers (i.e. the number of support vectors). A free implementation can be found, e.g. in [53].

4.3.2 Decomposition methods

Decomposition methods are similar in spirit to chunking as they solve a sequence of small QPs as well. But here the size of the subproblems is fixed. They are based on the observations that a sequence of QPs which at least always contains one example violating the KKT conditions will eventually converge to the optimal solution [54, 55]. Osuna [55] suggested keeping a constant-size matrix for every QP sub-problem, which implies adding and deleting the same number of examples in each iteration. Using a constant-size matrix allows the training of arbitrary large data sets. The algorithm given in [55] added and deleted one example in each iteration. In practice, however, the convergence of such an approach is very slow. Practical implementations use sophisticated heuristics to select several patterns to add and remove from the subproblem plus efficient caching methods. They usually achieve fast convergence even on large data sets with up to several thousands of support vectors. A good quality implementation is SVMlight [56]. Alternatively, a decomposition variant is contained in the package [53].

4.3.3 Sequential minimal optimisation (SMO)

Sequential Minimal Optimisation (SMO) algorithm is derived by taking the idea of the decomposition methods to its extreme and optimising a minimal subset of just two points at each iteration (to keep constraint (4.1.2) satisfied at least two elements of $\vec{\alpha}$ need to change simultaneously). The power of SMO resides in the fact that the optimisation problem for two data points admits an analytical solution, eliminating the need to use an iterative QP solver as part

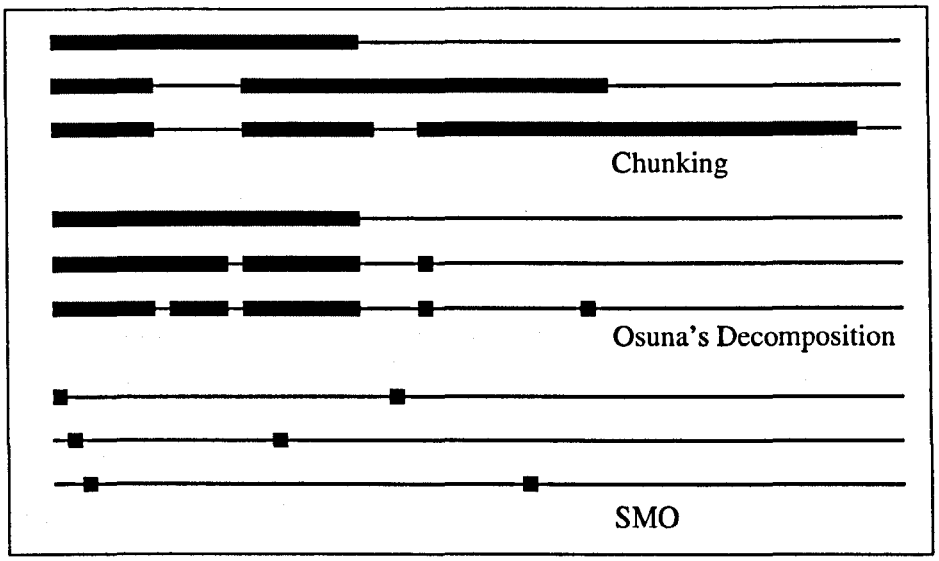


Figure 4.1: Three alternative algorithms for training SVMs: Chunking, Osuna's Decomposition and SMO. For each algorithm, three iterations are illustrated. (Cited from paper [58].)

of the algorithm. In each iteration SMO chooses two elements α_i and α_j to jointly optimise, finds the optimal values for those two parameters given that all the others are fixed, and updates the $\vec{\alpha}$ accordingly. Here the main problem remains to choose a good pair of parameters to optimise in each iteration. The heuristics presented in the original papers [57, 58] are based on the KKT conditions and there has been some work (e.g. [59]) to improve them. The implementation of the SMO approach is straightforward. The pseudocode of it may be found in [57, 58].

Figure 4.1 illustrates the difference between the three alternative algorithms for training SVMs: Chunking, Osuna's Decomposition and SMO. For Chunking, a fixed number of examples are added every iteration, while the zero Lagrange multipliers are discarded in every iteration. Thus, the number of examples trained per iteration tends to grow. For Osuna's Decomposition, a fixed number of examples are optimised every iteration. For SMO, only two examples are analytically optimised every iteration.

Chunking techniques in SVMs were already used by Vapnik and Chervo-

nenkis, and were improved, generalised and discussed in a number of papers among others, e.g. [55, 60, 61] by Osuna and Girosi, [56] by Joachims, [57, 58] by Platt and [62] by Kaufmann. The work of Osuna and Girosi inspired the subsequent work on data selection, which ultimately led to systems like SMO.

The SMO algorithm was devised by Platt [57, 58] and applied to text categorisation problems. An extension of SMO, differing in the way it calculates the bias, has been proposed in [63] and shown to be faster. Alex Smola has generalised SMO for the case of regression [64] and the code can be found at the website [65].

4.3.4 Other algorithms

Further algorithms have been proposed to solve the SVM problem or a close approximation. For instance, the Kernel-Adatron [66] is derived from the Adatron algorithm [67] which was proposed originally in a statistical mechanics setting. It constructs a large margin hyperplane using online learning. Its implementation is very simple. However, its drawback is that it does not allow for training errors, i.e. it is only valid for separable data sets, to hard-margin SVMs.

Keerthy et al. proposed a very elegant algorithm for SVMs [68] that does not maximise the margin by minimising the norm of the weight vector. Rather, they note that the distance between the nearest points of the convex hulls of the positive and negative data uniquely determines the maximal margin hyperplane. Based on the same approach, Kowalczyk [69] has proved a convergence rate for a new iterative algorithm that can be applied to problems with hard or soft margins, as well as experimental comparison with other iterative algorithms.

4.4 Centre-Based Optimisation

In this section, a series of centre-based algorithms that have been proposed by us are presented. Like most existing algorithms for fast training of SVMs, these algorithms are iterative and train a sequence of small QP problems. While unlike the others, these centre-based algorithms fasten the training of SVMs via a different way rather than selecting a particular subset of examples and then restricting the training to that subset. These new algorithms extract classification information contained in the full training data set and perform training on the compressed version of the full training set. The basic implementation scheme is to compress the original training set and then train the machine on the compressed training set. As stated in previous chapter, the typical way to compress a data set takes two steps: 1) divide the original data set into clusters; 2) represent object points in each of these clusters with the centre of that cluster. Our new methods follow this procedure and also use the set of cluster centres to represent the original data set, where the centre of the rth cluster $\mathbf{C}_r$ is defined as follows

$$\vec{c}_r = \frac{\sum_{\vec{x}_j \in \mathbf{C}_r} \vec{x}_j}{\sum_{\vec{x}_j \in \mathbf{C}_r} 1}. \tag{4.4.1}$$

But what is the desired clustering that would bring us the optimal decision boundary? Recall that the set of support vectors is all we need to construct a correct decision boundary. This implies that, in order to make use of all the classification information contained in the original training set, each support vector must appear in the compressed training set and thus must become a cluster centre. Therefore, the desired compression should assign each support vector to a cluster and the remaining examples to another cluster so as to reduce redundance. However it is impossible to achieve this in a straightforward way since we do not know beforehand which training examples would turn out to be support vectors. What we have tried is firstly including some redundance

in the clustering (i.e., more clusters than that in the ideal clustering) and then discarding this redundance from training, i.e. reducing the size of the QP problem at each training step. Attempts have been made to develop a heuristic approach for this.

4.4.1 Centre-based optimisation (CO)

The algorithm

Table 4.1: Implementation steps of the centre-based optimisation (CO) algorithm.

Given a training set **S**, split each class of **S** into two subclusters.

Initialise the working set $\hat{\mathbf{S}}$ to the centres of these four subclusters.

REPEAT

Train SVM on $\hat{\mathbf{S}}$.

FOR each support vector $(\vec{v}_c)_i$ in $\hat{\mathbf{S}}$

find the cluster $\mathbf{C}_r$ that has the current support vector $(\vec{v}_c)_i$ as its centre.

split cluster $\mathbf{C}_r$ into two subclusters.

add the centres of these two subclusters into $\hat{\mathbf{S}}$ and delete $(\vec{v}_c)_i$ from $\hat{\mathbf{S}}$.

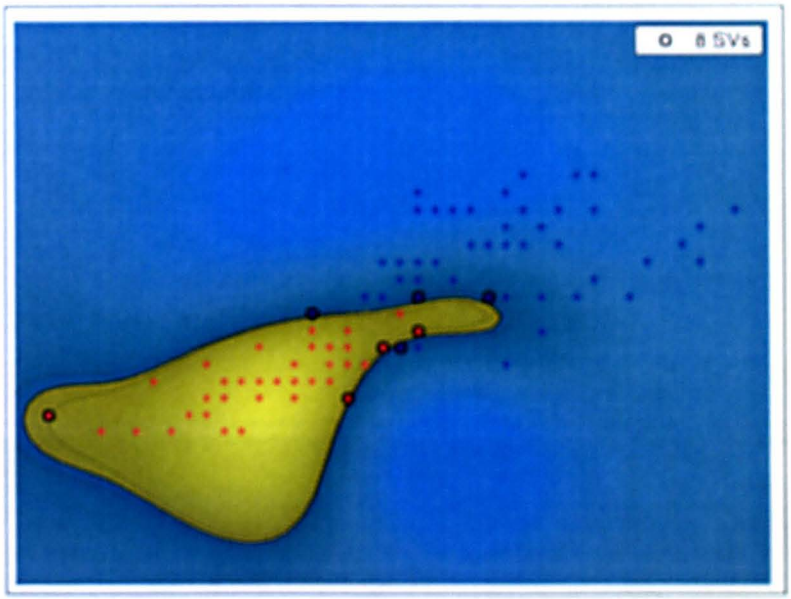
UNTIL no further splitting is possible.

The implementation steps of our first attempt is given in Table 4.1. The set of the cluster centres being optimised in a particular iteration is referred to as the *working set*. Let $\{\vec{v}_i\}$ denote the set of support vectors found using full training set and $\{(\vec{v}_c)_i\}$ the set found using a set of cluster centres. Examples in a cluster whose centre is in $\{(\vec{v}_c)_i\}$ are more likely to contain important classification information than other examples. In other words, examples in a cluster whose centre is in $\{(\vec{v}_c)_i\}$ are more likely to be in $\{\vec{v}_i\}$. And recall that to have the optimal decision boundary each member of $\{\vec{v}_i\}$

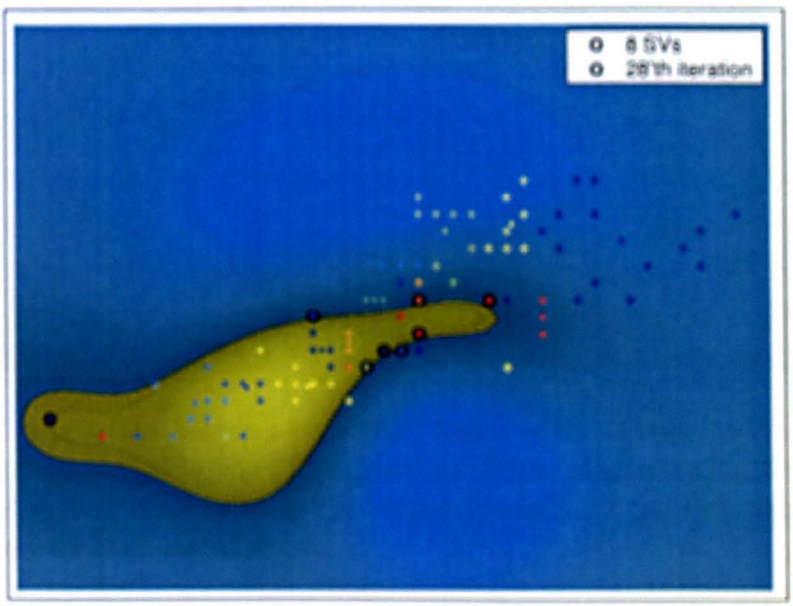
should occupy a cluster while to achieve the minimum redundance all the other examples should take as few clusters as possible. Considering these, in each iteration, each of the clusters whose centres are in $\{(\vec{v}_c)_i\}$ is split further into two new subclusters and then the working set is updated to include the centres of these new clusters. This procedure is iterated, splitting each support-vector clusters into two subclusters, restricting the training to the current set of cluster centres, and finally halting when no further splitting is possible (i.e. when each support-vector cluster contains only that support vector itself). Since this new algorithm extracts classification information from the compressed data set that is composed of cluster centres, it is called centre-based optimisation (CO) algorithm [19]. The CO algorithm has been implemented in MATLAB. The quadratic programming subroutine provided in MATLAB optimisation toolbox has been used as the standard technique to compare with. The QP problem in each iteration of the CO is also solved using this subroutine.

Experiment and results

The CO has been tested on the Iris data set described in section 3.4. Again to visualise the problem, experiments were conducted to separate the classes of Iris Versicolour and Iris Virginica according to the two attributes - petal length and petal width. Figures 4.2(a) and 4.2(b) show the decision boundaries obtained for these data sets using the standard QP technique and our CO algorithm, respectively. Comparing to the optimal decision boundary found by the standard QP technique, the decision boundary has been well approximated by the CO algorithm in terms of most support vectors have been identified. However, differences do exist. In Figure 4.2(b), one support vector has been lost. Moreover, different runs of the CO may result in different decision boundaries (see Figure 4.3 for another decision boundary that may possibly be generated by the CO algorithm). The reason for multiple solutions is that the c-means



(a) decision boundaries found using standard technique



(b) decision boundaries found using the CO algorithm

Figure 4.2: The decision boundaries found with a Gaussian SVM for two-feature Iris data set using (a) the standard technique and (b) the CO algorithm, respectively. Positive examples and negative examples are marked with 'x's and '+'s, respectively. Support vectors are marked with dark circles. The solid line denotes the decision boundary. The area between the dotted lines shows the margin. In Figure (b), different clusters are indicated by different colours. Each cluster center in the working set is marked with a dot with the same colour used for the members of that cluster.

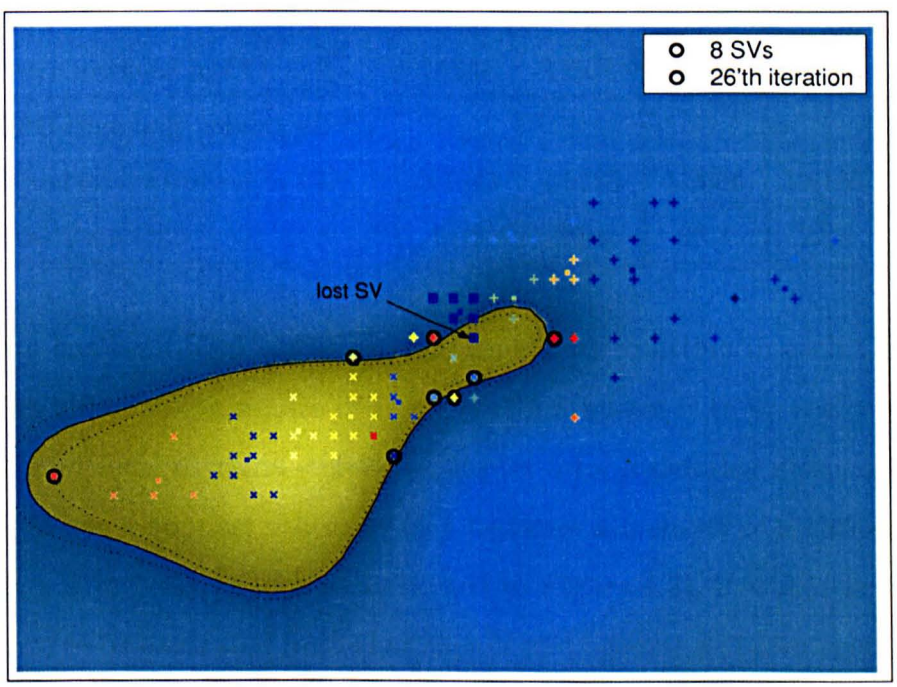


Figure 4.3: A possible decision boundary found with a Gaussian SVM for two-feature Iris data set using the CO algorithm. Same markers as in Figure 4.2(b) are used. Moreover, examples in the cluster containing the lost support vector are indicated with an extra square.

Table 4.2: Comparison of the standard QP technique and the CO algorithm for a Gaussian SVM on image segmentation subsets.

problem size	891	800	400	200	100
CPU time of standard algorithm	3920.5	2264.4	180.3	15.7	2.5
CPU time of CO	3.8567	3.3275	2.0084	1.2917	0.9882
size of Q involved in standard training	793881	640000	160000	40000	10000
total size of all the Q matrices involved in CO	14238	12928	8908.7	6835.1	5119.8
total CPU time only solving all the QP subproblems involved in CO	2.1043	1.9185	1.3666	1.0166	0.8201
no. of CO iterations	31.2	29.4	25.8	22.8	21

algorithm has been used as the method for cluster splitting in each CO iteration. The hill-climbing nature of this algorithm makes it to be easily trapped into different local extrema (refer to Chapter 5 for a detailed discussion on c-means). When a support vector in $\{\vec{v}_i\}$ becomes a member of a cluster whose centre is not in $\{(\vec{v}_c)_i\}$ (such as indicated in Figure 4.3), it would never have the chance to be separated from that cluster and become a cluster centre in the working set. Thus, the classification information contained in such a support vector would be lost in such case.

Despite the inaccuracy and multiplicity of the resulting decision boundaries, the CO is very fast. To investigate the increase of training time with the size of the full training set, the image segmentation data set described in section 2.5 has been used in the experiment and the size of the data set was varied by randomly taking subsets of the complete training set. Table 4.2 compares the performance of the CO algorithm with the standard QP technique. Due to the uncertainty caused by the c-means algorithm in a CO run, the results of the CO algorithm are averaged over 100 independent runs. So is the CPU time of a

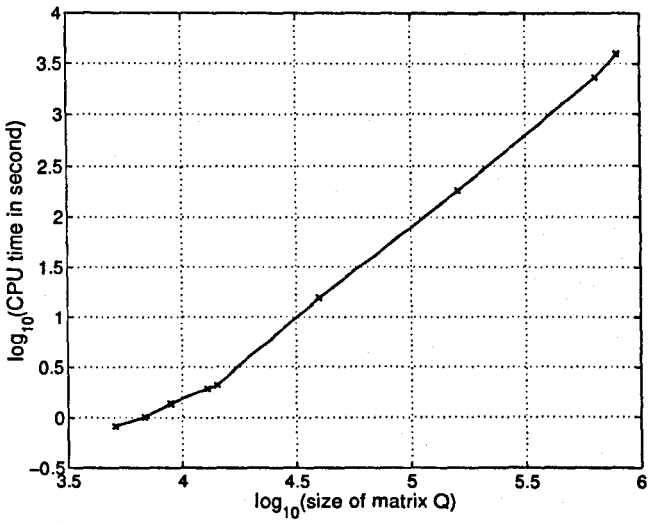


Figure 4.4: The log-log plot of solving time versus the size of a QP problem.

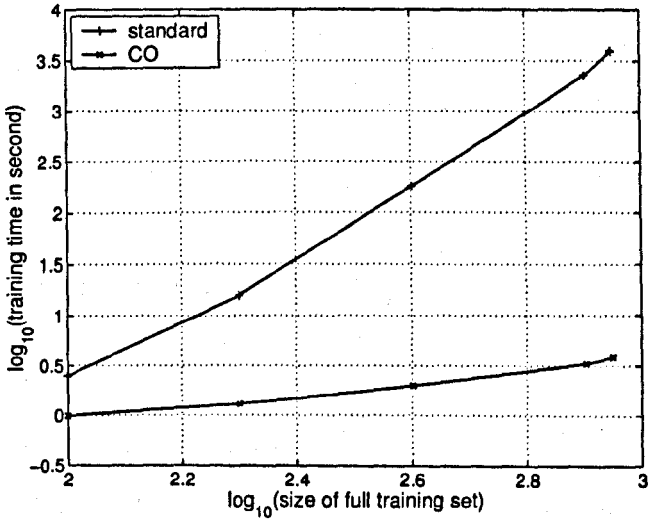


Figure 4.5: The log-log plot of training time versus the size of full training set for the CO algorithm and the standard technique on image segmentation data set.

standard run. By the CO, the training time is reduced dramatically, especially for large training sets. And for the largest (complete) training set, the training time has been reduced to the $(1/10^3)$'th of that cost by the standard technique. This significant improvement is due to the following facts:

1) as indicated in Table 4.2, the full QP problem has been converted to a sequence of small QP problems and the total size of these small QP problems is much smaller than the size of the corresponding full QP problem;

2) as shown in Figure 4.4, the CPU time for solving a QP problem grows exponentially with the size of the matrix Q;

3) as indicated in Table 4.2, the CPU time of a CO run is dominated by the solving time for the sequence of small QP problems.

Figure 4.5 shows the log-log plot of training time in seconds versus the size of the full training set for the CO and the standard technique on the image segmentation data set. By fitting a line to the plot and then working out the gradient of the line, an empirical scaling for the algorithm can be derived. The training time of the standard technique scales $l^{3.27}$, where l denotes the size of the full training set, while the CO time scales $l^{0.68}$. This indicates the great potential of centre-based algorithms for fast solving of SVM optimisation problems with large training sets.

4.4.2 Error-centre-based optimisation (ECO)

The algorithm

As stated, the optimality of the resulting decision boundary can not be guaranteed by the CO. Some support vectors may be missed. Observing Figures 4.2(b) and 4.3, we can see that these missed support vectors lie either inside or on the wrong side of the margin. And since they were not involved in last training their corresponding α_i are zero. Remind that the KKT conditions are the necessary and sufficient conditions for the optimal solution. Equation

(2.3.18) for hard-margin SVMs and equation (2.4.8) for soft-margin SVMs indicate that examples with zero α_i must be correctly classified and lie outside the margin. Inspired by this, modification has been made to the CO. In our second attempt, each cluster is split into two sub-clusters by separating those examples that satisfy the KKT conditions and thus lie outside or on the current margin from those that violate the KKT conditions and thus lie inside or on the wrong side of the current margin. On the one hand, as long as there are examples in the original training set violate KKT condition at least one cluster would be split. On the other hand, the procedure iterates until no example in the original training set violates the KKT conditions. Hence, the optimality of the solution found by this technique is guaranteed. Again, this new algorithm builds SVMs using a set of cluster centres. Here, we refer examples that violate the KKT conditions as margin errors. To further reduce the size of the QP problem in each iteration, only are the clusters of the margin errors involved in the SVM training. The rest clusters are represented by the support vectors found in the previous iteration. Moreover, it has been proved by Osuna [55] that the large QP problem can be broken down into a series of smaller QP sub-problems. As long as at least one example that violates the KKT conditions is added to the examples for the previous sub-problem, each step will reduce the overall objective function and maintain a feasible solution that obeys all of the constraints. Therefore, a sequence of QP sub-problems that always add at least one violator will be guaranteed to converge. Considering this, in order to ensure a strict improvement in the objective function and hence convergence, the new algorithm inserts an error centre into the working set only if it violates the KKT conditions. Otherwise, the example in that cluster that most violates the KKT conditions will be inserted into the working set as the representative of its cluster. Since most examples of the working set are the centres of error clusters (support vectors of previous iteration must have been centres of error clusters), this new algorithm is called error-centre-based optimisation (ECO).

Table 4.3: Implementation steps of the error-center-based optimization (ECO) algorithm.

Given a training set $\mathbf{S}$, treat each class of $\mathbf{S}$ as a cluster.

Initialize the working set $\hat{\mathbf{S}}$ to the centers of these two clusters.

REPEAT

Train SVM on $\hat{\mathbf{S}}$.

Set $\hat{\mathbf{S}}$ to the support vectors.

FOR each cluster $\mathbf{C}_r$ of $\mathbf{S}$

split the current cluster $\mathbf{C}_r$ into two subclusters by identifying the margin errors, i.e. those that violate the KKT conditions.

IF center of the error cluster violates the KKT conditions

add the center into $\hat{\mathbf{S}}$.

ELSE

add the example, the worst point violating the KKT conditions in $\mathbf{C}_r$, into $\hat{\mathbf{S}}$.

UNTIL no new margin error is found.

The implementation steps of ECO are listed in Table 4.3. When examples are tested against the KKT conditions, function $f(\vec{x})$ is calculated (see equations (2.3.18) and (2.4.8)). To reduce the computational time, only support vectors are involved in the calculation of $f(\vec{x})$ (equation (2.3.16)).

Experiments and results

The ECO algorithm has been implemented in MATLAB. And again, the quadratic programming subroutine provided in MATLAB optimisation toolbox have been used as the standard technique to compare with. The QP problem in each iteration of the ECO is also solved by this subroutine.

Like CO, the ECO has been tested on Iris and image segmentation data sets. Figures 4.6 and 4.7 show the decision boundaries obtained using the ECO for

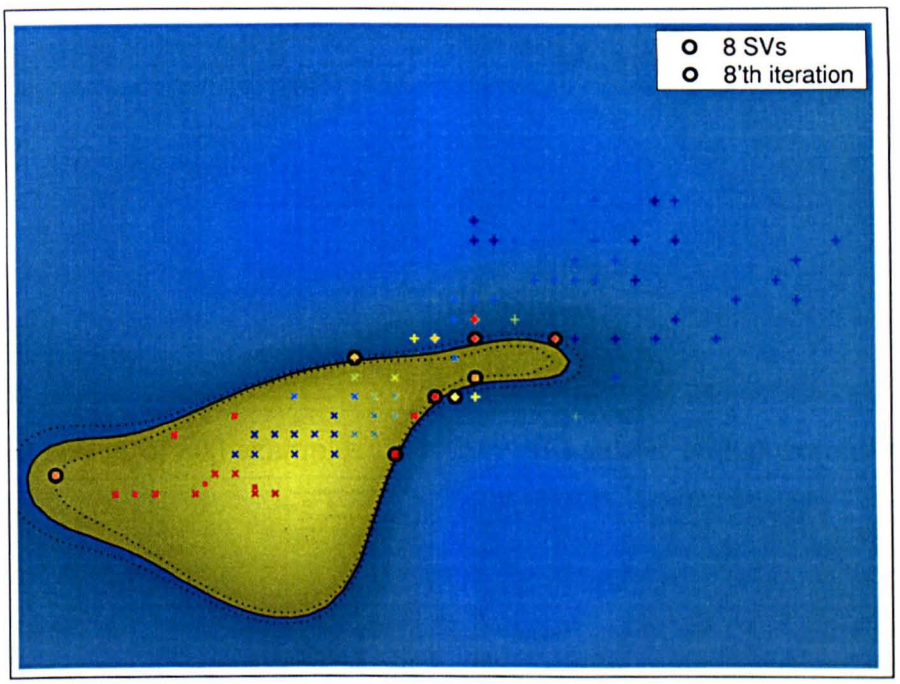


Figure 4.6: The decision boundaries found with a Gaussian SVM using the ECO algorithm for two-feature Iris data set. Same markers as in Figure 4.2(b) are used.

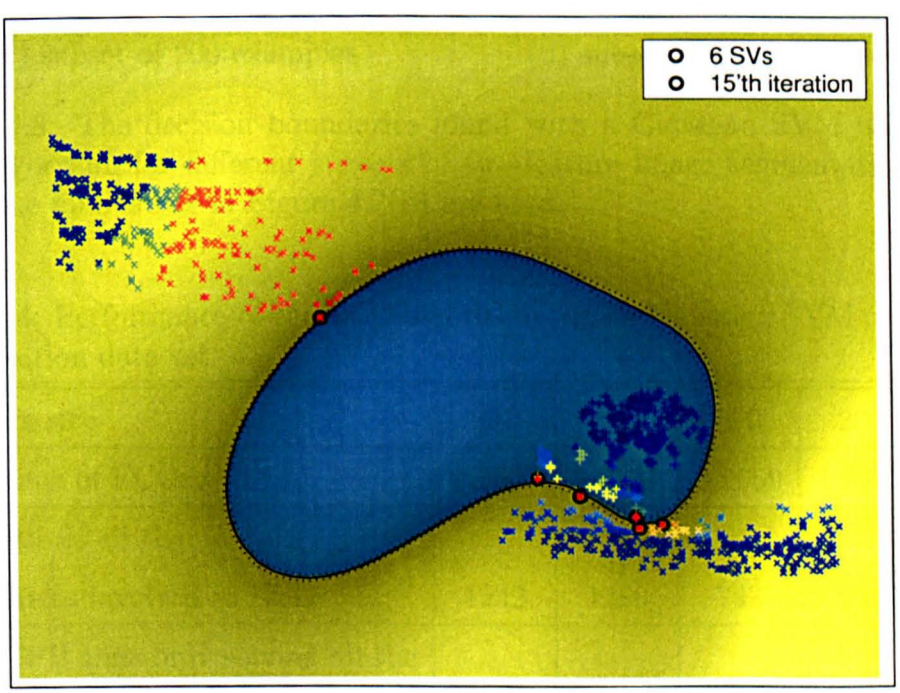
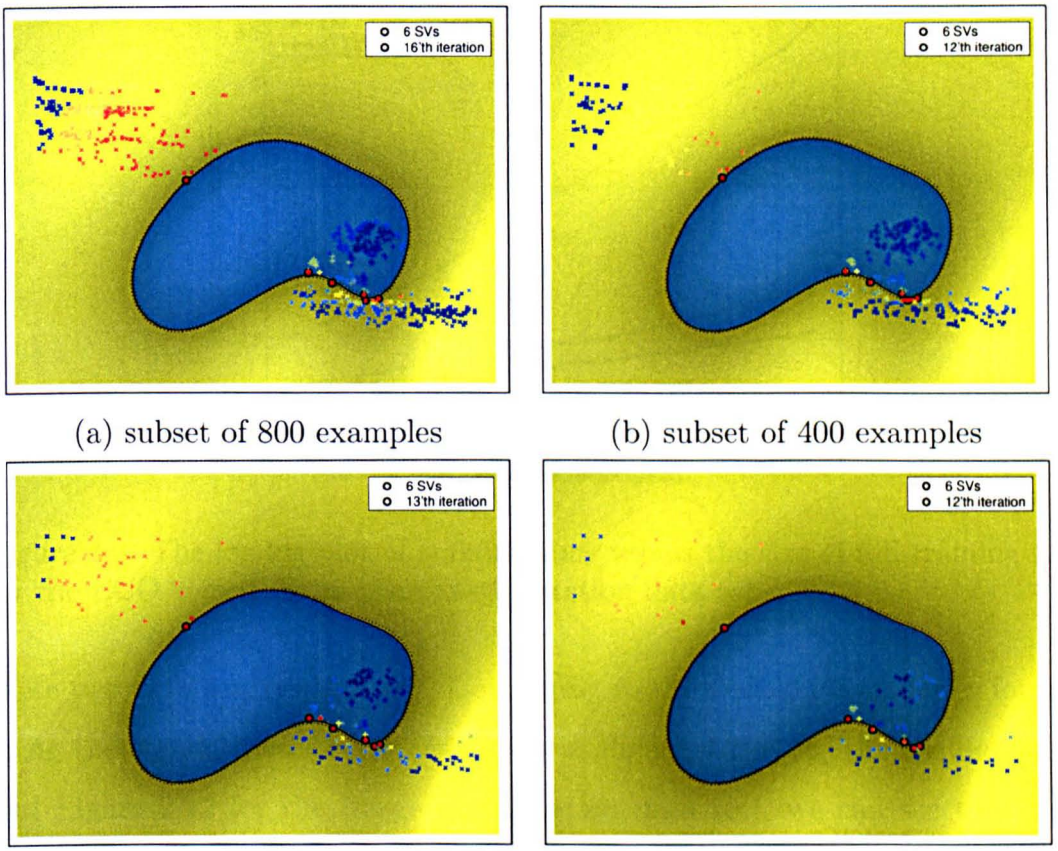


Figure 4.7: The decision boundaries found with a Gaussian SVM using the ECO algorithm for two-feature image segmentation data set. Same markers as in Figure 4.2(b) are used.



(a) subset of 800 examples (b) subset of 400 examples

(c) subset of 200 examples (d) subset of 100 examples

Figure 4.8: The decision boundaries found with a Gaussian SVM using the ECO algorithm for different subsets of two-feature image segmentation data set. Same markers as in Figure 4.2(b) are used.

Table 4.4: Performance of the ECO algorithm with a Gaussian SVM on image segmentation data set.

problem size	891	800	400	200	100
CPU time of ECO	9.1603	8.8201	3.2659	1.9307	0.9404
total size of all the Q matrices involved in ECO	1242	1280	742	934	610
total CPU time only solving all the QP subproblems involved in ECO	0.2137	0.1834	0.1133	0.1274	0.0863
no. of ECO iterations	15	16	12	13	12

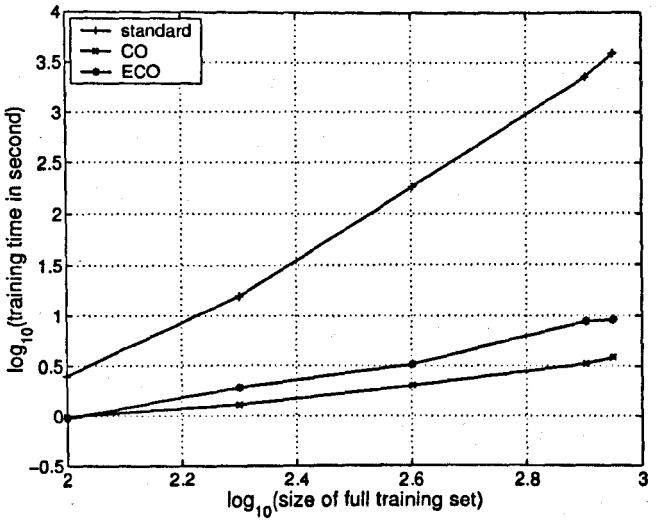


Figure 4.9: The log-log plot of training time versus the size of full training set for the ECO algorithm on image segmentation data set.

these two data set, respectively. Comparing to the decision boundaries found using the standard technique (as shown in Figure 4.2(a) for the Iris data set and Figure 2.12 for the image segmentation data set), we can see that the results are exactly the same. Therefore, the optimal SVM has been found by the ECO. Moreover, since no randomness resides in the ECO procedure the decision boundary generated by the ECO for a particular training set is certain and unique.

The increase of training time with the growth of the size of training set has been investigated for the ECO algorithm. Figures 4.8(a) - (d) show the decision boundaries found with a Gaussian SVM using the ECO for training subsets of 800, 400, 200 and 100 examples, respectively. Since the whole set of support vectors has been included in each of these subsets deliberately, the optimal decision boundary for each of these training subsets is expected to the same as that of the complete training set. As shown in Figures 4.8(a) - (d), the optimal decision boundary has been found by the ECO for each training subset. Table 4.4 summarises the performance of the ECO algorithm for a Gaussian SVM on the image segmentation data set. Considering that the conditions of a

CPU varies, we averaged the CPU times over 100 independent runs. The ECO converges faster than the CO in terms of the number of iterations. The total amount of CPU time spent in solving QP problems is less since the total size of matrix Q's in a ECO run is less than in a CO run. However this is achieved at the cost of error checking. The running time of ECO is dominated by error checking. Figure 4.9 shows the log-log plot of training time in seconds versus the size of the full training set for the ECO, the CO and standard technique on the image segmentation data set, respectively. The ECO is much faster than the standard technique. While due to the extra time spent in error checking, the ECO algorithm requires longer training time than the CO. However, the ECO can guarantee an optimal solution while the CO cannot. The training time of ECO scales $l^{1.04}$, i.e. grows almost linearly with the size of the full training set.

4.4.3 ECO for soft-margin SVMs

The algorithm

Last subsection demonstrated the success of the ECO algorithm for hard-margin SVMs. However, when the ECO is applied to a SVM with soft margin problem arises.

For a SVM with soft margin, noisy examples are allowed to remain inside or even on the wrong side of the optimal margin. On the contrary, by applying the KKT conditions in error checking and then involve error centers into training, the ECO actually tries to push all the training examples outside the final margin. It may happens that even though all the examples lie inside or on the wrong side of the margin are identified by the KKT conditions in the error checking step, the QP solving step will allow their cluster centers to remain inside or on the wrong side of the margin. Consequently, the decision boundary does not move. The same group of error points are detected. Further iterations

will bring no improvement. The problem is that the training procedure of ECO will not stop until all the training examples are outside the margin. To solve this problem, the soft version of the ECO stops when no new error cluster is formed, i.e. when no new error center is added to the working set.

Experiments and results

The ECO has been tested on the image segmentation data set for $C = 1000$, $C = 100$, $C = 10$ and $C = 1$. The resulting decision boundaries are shown in Figures 4.10 - 4.13, respectively. And the numbers of support vectors are given in Table 4.5. These decision boundaries have been tested on the training set. The number of errors on the training set is also given in Table 4.5. Comparing with those obtained using the standard technique for the same values of C (see Table 2.2 and Figures 2.8 - 2.11), the decision boundaries obtained using different algorithms are almost the same for large values of C. However, as the value of C decreases, difference appears between the results obtained by the ECO and the standard QP technique. As shown in Figure 4.12 for $C = 10$, the ECO allows more margin errors than the standard technique. And as C decreases to 1, significant difference are observed. The reason for this is that, in the training procedure of ECO, the SVM is trained on and thus penalises cluster centres rather than individual examples. If the clusters of errors are small and each contains just a few points, then the number of error cluster centres and the number of error examples should be similar. The decision boundary obtained using the ECO should be similar to that obtained using standard technique. This is exactly the case when C is large, such as observed in Figures 4.10 and 4.11 where error points scatter alone the margin boundaries and each error cluster contains very few error points. When C is small, the number of margin errors is expected to be large and error clusters each containing a few of example points may form. As a result, the number of error cluster

centres should be much less than the number of error examples; the second term of objective function (2.4.1) becomes smaller under ECO as if a smaller C had been used, such as the case shown in Figure 4.13. The motivation of introducing C into SVM is to discard noisy examples automatically. If a small C is used, examples containing classification information may be lost and the number of support vectors will be much higher while most of them are at the upper bound. Considering this, the use of a small C is not recommended in practice. While if a large C is used, the result given by the ECO is expected to be the same or at lest very similar to that by the standard technique.

The increase of training time with the growth of the size of the training set has been investigated for soft-margin SVMs using the ECO. Like in experiments with hard-margin SVMs, subsets of image segmentation data set have been used. Figures 4.14(a) - (d) show the corresponding decision boundaries found with a Gaussian SVM for $C = 100$. Comparing with that shown in Figures 4.15(a) - (d) for the standard technique, similar decision boundaries have been found by the different algorithms. Table 4.6 summarises performance of the ECO on these subsets. Comparing with that listed in Table 4.7 for the standard technique, the group of noisy examples identified by the different algorithms are almost the same for each value of C. Figure 4.16 shows the log-log plot of training time in seconds versus the size of the full training set for the ECO and the standard technique on image segmentation data set when $C = 100$. Similar scaling of increment in training time has been observed for the ECO with different values of C. Again, the great potential of ECO for fast training of SVMs with large training set is demonstrated.

4.5 Conclusion

The standard QP technique is not suitable for the case of SVM training. Considering this, new centre-based algorithms, the CO algorithm and the ECO

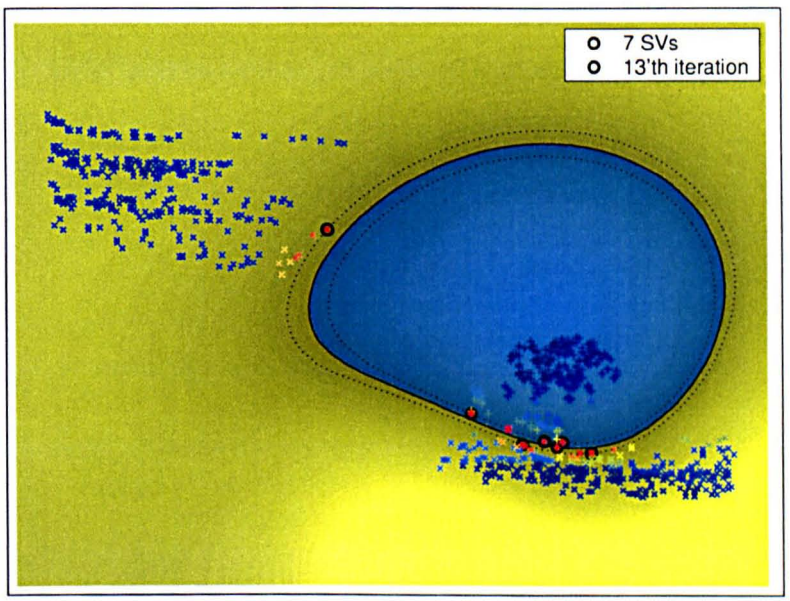


Figure 4.10: The decision boundary found with a Gaussian SVM using the ECO algorithm for $C = 1000$. Same markers as in Figure 4.2(b) are used. And the training error points are marked with an extra magenta square.

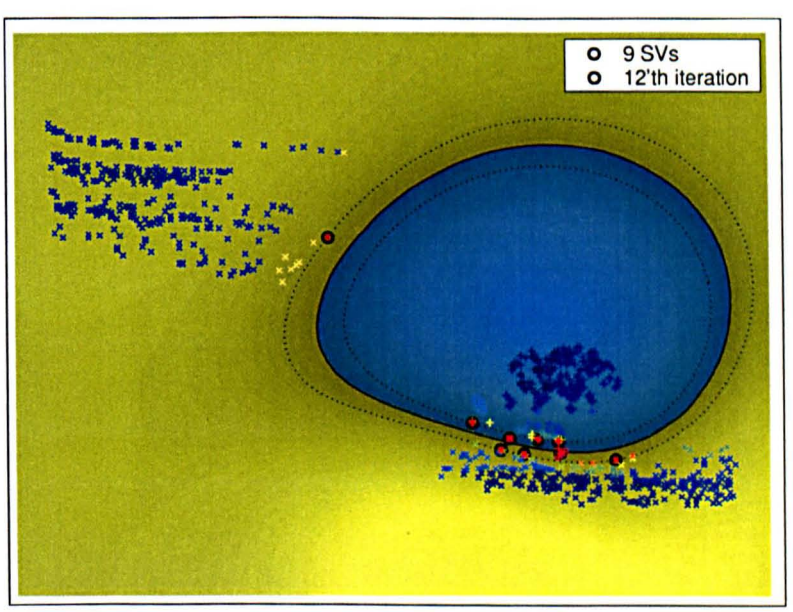


Figure 4.11: The decision boundary found with a Gaussian SVM using the ECO algorithm for $C = 100$. Same markers as in Figure 4.10 are used.

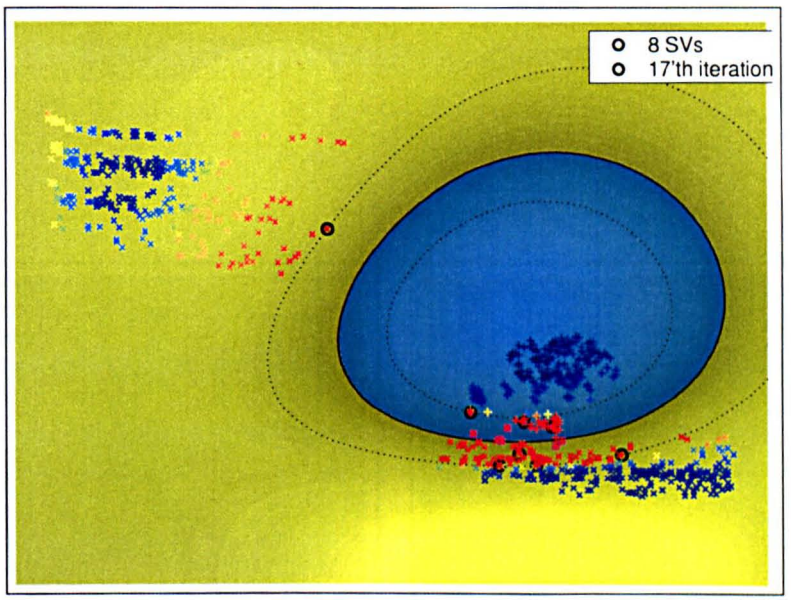


Figure 4.12: The decision boundary found with a Gaussian SVM using the ECO algorithm for $C = 10$. Same markers as in Figure 4.10 are used.

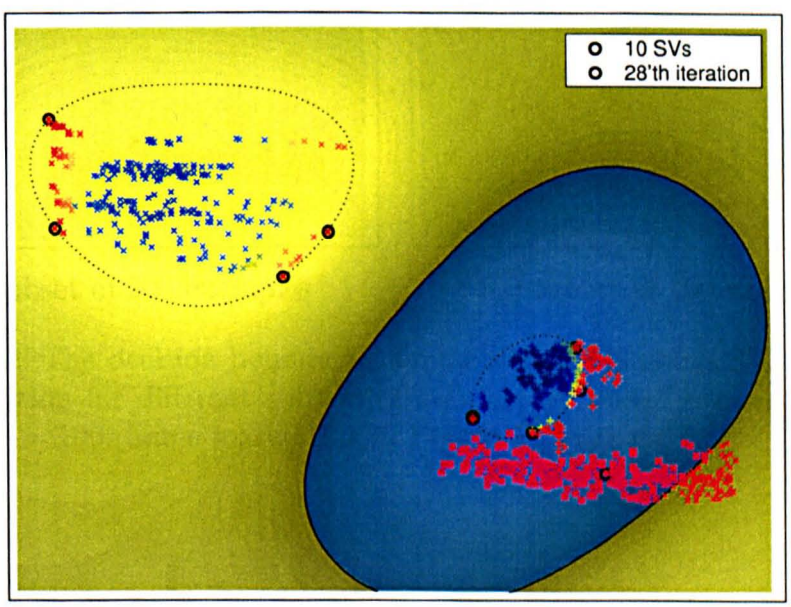
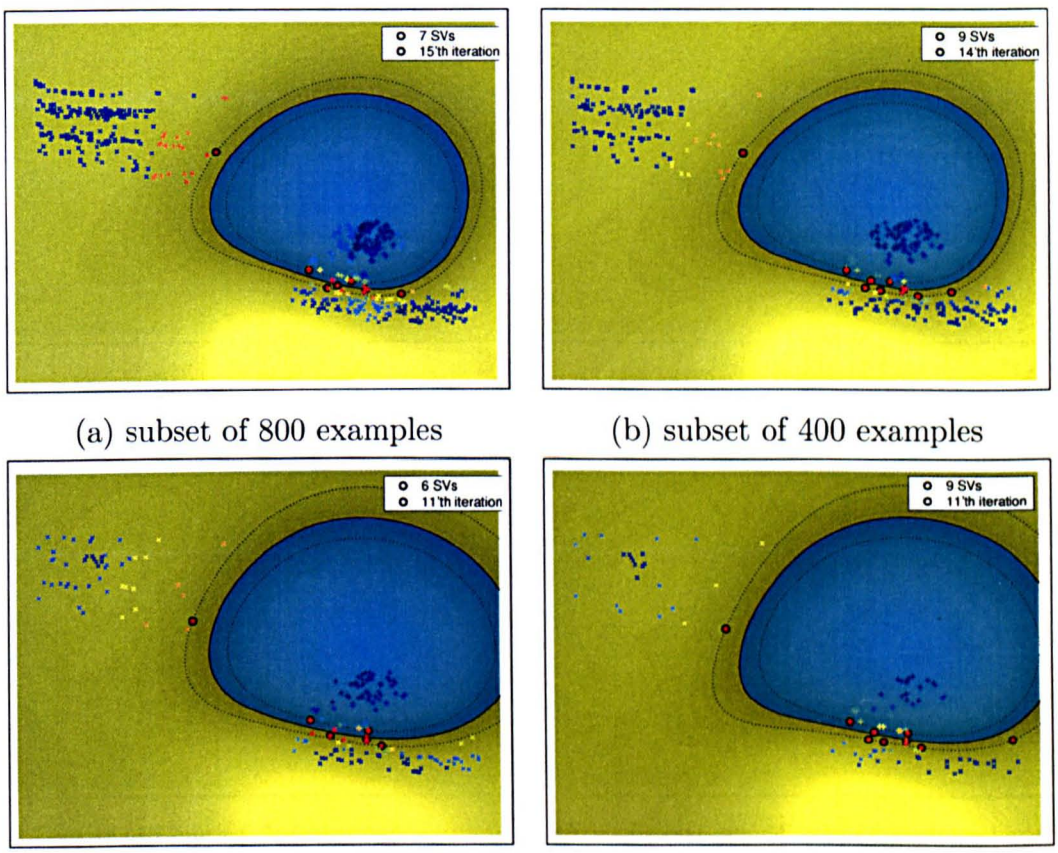


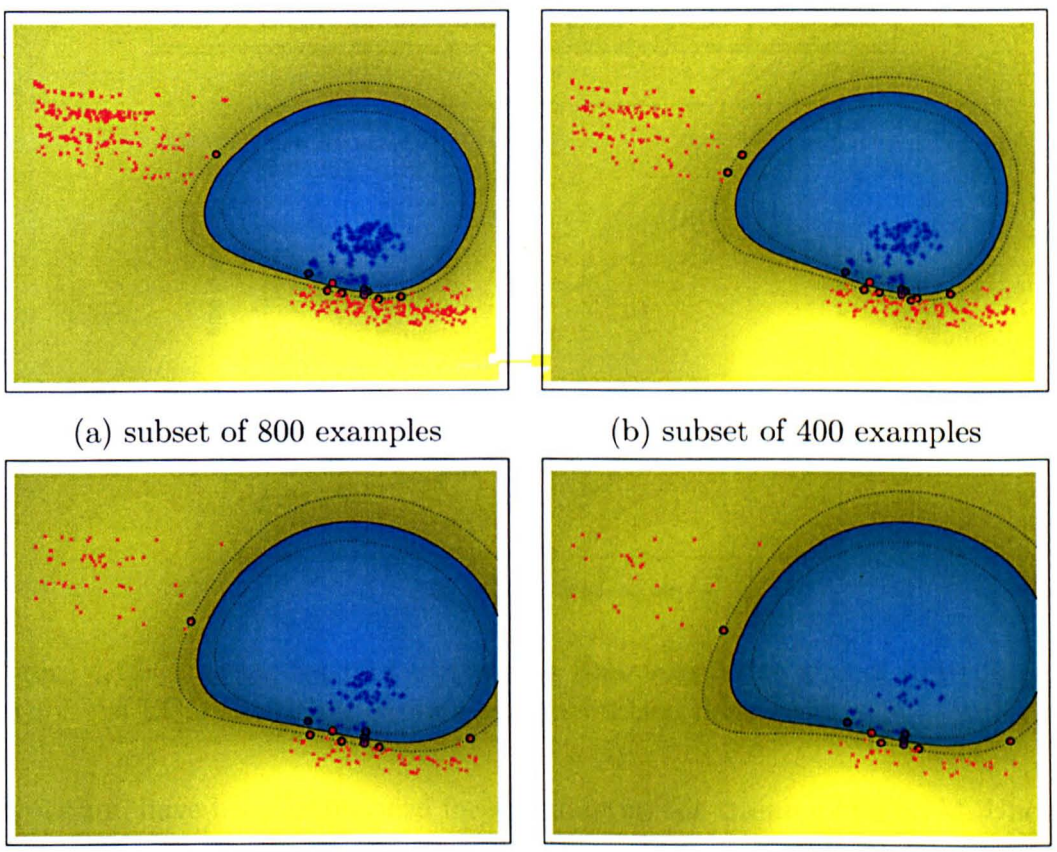
Figure 4.13: The decision boundary found with a Gaussian SVM using the ECO algorithm for $C = 1$. Same markers as in Figure 4.10 are used.



(a) subset of 800 examples (b) subset of 400 examples

(c) subset of 200 examples (d) subset of 100 examples

Figure 4.14: The decision boundaries found with a Gaussian SVM using the ECO algorithm for different subsets of two-feature image segmentation data set when $C = 100$. Same markers as in Figure 4.10 are used.



(a) subset of 800 examples (b) subset of 400 examples

(c) subset of 200 examples (d) subset of 100 examples

Figure 4.15: The decision boundaries found with a Gaussian SVM using standard technique for different subsets of two-feature image segmentation data set when $C = 100$. The positive and negative examples are marked with 'x's and '+'s, respectively. And the training error points are marked with an extra magenta square.

Table 4.5: Performance of the ECO algorithm with a Gaussian SVM on image segmentation data set for different values of C. SVs stands for support vectors.

C	1000	100	10	1
no. of support vectors	7	9	8	10
no. of unbounded SVs	6	5	5	7
no. of errors on training set	1	3	6	224

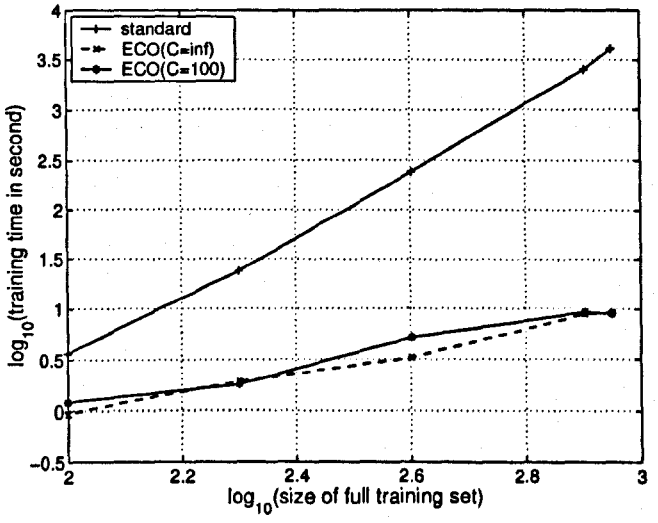


Figure 4.16: The log-log plot of training time versus the size of full training set for the ECO algorithm on image segmentation data set.

algorithm, have been introduced for speeding up the training of a SVM. Under them, the full training set is compressed and represented by the set of cluster centres. In the training process, more and more error centres are added into the current working set until the optimal solution is obtained. The optimality of the solution obtained by the ECO is guaranteed since the KKT conditions have been used as its stop criterion. Moreover, its great potential for large training sets has been demonstrated by the experimental results as the training time scales almost linearly in the training set size.

Table 4.6: Performance of the ECO algorithm with a Gaussian SVM on different image segmentation subsets when $C = 100$. SVs stands for support vectors.

problem size	891	800	400	200	100
CPU time of ECO	8.9181	9.2428	5.1553	1.7952	1.1951
total size of all the Q matrices involved in ECO	1141	1236	1267	688	875
total CPU time only solving all the QP subproblems involved in ECO	0.2907	0.2861	0.3494	0.1521	0.2035
no. of ECO iterations	12	15	14	11	11
no. of SVs	9	7	9	6	9
no. of unbounded SVs	5	5	5	4	7
no. of errors on training set	3	2	3	2	2
index of error points	376 553 589	796 797	13 396 397	196 197	96 97

Table 4.7: Performance of the standard QP technique with a Gaussian SVM on different image segmentation subsets when $C = 100$. SVs stands for support vectors.

problem size	891	800	400	200	100
CPU time of standard algorithm	4138.6	2543.8	241.86	23.752	3.6493
size of Q involved in standard training	793881	640000	160000	40000	10000
no. of SVs	11	10	12	10	9
no. of unbounded SVs	6	5	6	5	5
no. of errors on training set	2	2	2	2	2
index of error points	376 553	796 797	396 397	196 197	96 97

Chapter 5

UNSUPERVISED LEARNING AND CLUSTERING

So far, our interest has focused on supervised learning, whose task is to learn the decision boundary between the classes given by the teacher. From this chapter, we depart from this path and concentrate on unsupervised learning, where the learner seeks to develop a concept description from examples that have not been pre-classified by the teacher.

5.1 Introduction

In unsupervised learning all one has is a collection of unlabelled samples. One might wonder why anyone is interested in such an unpromising problem, and whether or not it is possible even in principle to learn anything of value from unlabelled samples. There are at least five basic reasons for interest in unsupervised procedures.

First, collecting and labelling a large set of sample patterns can be surprisingly costly. For instance, recorded speech is virtually free, but accurately labelling the speech - marking what word or phoneme is being uttered at each

instant - can be very expensive and time consuming. If a classifier can be crudely designed on a small set of labelled samples, and then "tuned up" by allowing it to run without supervision on a large, unlabelled set, much time and trouble can be saved.

Second, one might wish to proceed in the reverse direction: train with large amounts of (less expensive) unlabelled data and only then use supervision to label the groupings found. This may be appropriate for large *data mining* applications, where the contents of a large database are not known beforehand. Actually, this is what scientists (say, biologists) have been doing for centuries, developing such categories as vertebrates, subcategories as mammals or birds, and the like.

Third, in many applications the characteristics of the patterns can change slowly with time. For example, in automated food classification as the seasons change. If these changes can be tracked by a classifier running in an unsupervised mode, improved performance can be achieved.

Fourth, as already mentioned, we can use unsupervised methods to find features that will then be useful for categorisation. There are unsupervised methods that provide a form of data-dependent "smart preprocessing" or "smart feature extraction", e.g. the Principle Component Analysis and Independent Component Analysis methods.

Lastly, in the early stages of an investigation it may be valuable to perform exploratory data analysis and thereby gain some insight into the nature or structure of the data. The discovery of distinct subclasses - clusters or groups of patterns whose members are more similar to each other than they are to other patterns - or of major departures from expected characteristics may suggest that we should significantly alter our approach to designing the classifier.

The answer to the question of whether or not it is possible in principle to learn anything from unlabelled data depends upon the assumptions one is willing to accept. Assuming that we know the complete probability structure

for the problem except the values of some parameters, we can use maximum-likelihood methods to estimate these unknown parameters. Or if we are also provided with a known prior distribution of the unknown parameters, we can take a Bayesian approach to unsupervised learning. Such methodology is classified as parametric pattern classification. If the goal is to find subclasses (as addressed in the last reason above), then a more direct alternative is to use clustering procedures. Pattern classification using clustering procedures is nonparametric where the goal is to optimise an objective function rather than estimate some unknown parameters. The rest of this chapter is devoted a discussion on clustering.

5.2 Hard Clustering

A clustering problem concerns a set of n objects to be clustered into c clusters. If no overlap between clusters is allowed, then the problem becomes a *hard clustering* problem. Typically, the membership of the n objects to each of the c clusters is described by a $c \times n$ matrix, called *partition matrix*. For a degenerate partition, its partition matrix contains one or more empty rows, meaning that fewer than c clusters are obtained by this partition. For an n-object c-cluster clustering problem, the set of all $c \times n$ nondegenerate partition matrices is denoted by $\mathcal{M}$ and defined as follows

$$\begin{aligned} \mathcal{M} \;=\; & \{\mathbf{U} \in \mathcal{R}^{c\times n} \mid \sum_{i=1}^{c} U_{ik} = 1, 0 < \sum_{k=1}^{n} U_{ik} < n, \\ & 0 \le U_{ik} \le 1;\ i = 1,\ldots,c,\ k = 1,\ldots,n\} \end{aligned} \tag{5.2.1}$$

where $\mathbf{U}$ denotes a partition matrix.

For hard partitions, $U_{ik} \in \{0,\ 1\}$. Given an object $\vec{x}_k$, for all $k = 1,\ldots,n$,

each element of the kth column of $\mathbf{U}$ is defined by

$$U_{ik} = \begin{cases} 1, & \text{if } \vec{x}_k \text{ belongs to the } i\text{th cluster;} \\ 0, & \text{otherwise.} \end{cases} \tag{5.2.2}$$

The size of $\mathcal{M}$, i.e. the number of ways to cluster n objects into c nonempty groups, is a Stirling number of the second kind [70], given by

$$(1/c!)\sum_{l=0}^{c}(-1)^{c-l}\binom{c}{l}l^n, \tag{5.2.3}$$

where, as in the formulae above, n is the number of objects and c is the desired number of clusters.

In clustering, there are various objective functions. The following is one of the typical objective functions used in clustering

$$J_1(\mathbf{U},\mathbf{V}) = \sum_{i=1}^{c}\sum_{k=1}^{n} U_{ik}\cdot \mathrm{D}_{ik}^2(\vec{v}_i,\vec{x}_k) \tag{5.2.4}$$

where $\vec{x}_k \in \mathcal{R}^s$ is the feature vector describing the kth object, $\vec{v}_i \in \mathcal{R}^s$ is the feature vector describing the representative of the ith cluster, and $\mathrm{D}_{ik}^2(\vec{v}_i,\vec{x}_k)$ is the squared Euclidean distance between vectors $\vec{v}_i$ and $\vec{x}_k$. The set of c representatives, one for each cluster, composes a matrix $\mathbf{V} = [\vec{v}_1,\vec{v}_2,\ldots,\vec{v}_c]^{\mathrm{T}}$.

The objective function defined by equation (5.2.4) describes the accumulated error when replacing each object by the representative of the cluster it belongs to. Therefore, it is actually a measure of distortion. The objective of the problem is to minimise this distortion and consequently find the optimal partition. This clustering criterion is termed *sum-of-squared-error criterion* [3].

5.3 Hard c-Means Clustering Algorithm (HCMCA)

Of the various techniques that can be used to simplify the computation and accelerate convergence, c-means (also known as k-means) is an elementary but very popular clustering algorithm.

It has been shown that minimising distortion $J_1(\mathbf{U}, \mathbf{V})$ with respect to a hard partition matrix leads to the following two conditions [71]: for all $i = 1, \ldots, c$ and $k = 1, \ldots, n$, firstly

$$\vec{v}_i = \frac{\sum_{k=1}^{n} U_{ik}\vec{x}_k}{\sum_{k=1}^{n} U_{ik}}, \tag{5.3.1}$$

i.e. the representative of a cluster should be the centroid of the objects in that cluster (for this reason, representatives of the clusters are also referred to as cluster centres); and secondly,

$$U_{ik} = \begin{cases} 1, & \text{if } i = \arg\min_j \{\mathrm{D}_{jk}^2(\vec{v}_j, \vec{x}_k)\}; \\ 0, & \text{otherwise,} \end{cases} \tag{5.3.2}$$

i.e. an object should be assigned to the cluster with the closest centre.

Hard c-means clustering algorithm (HCMCA) adopts these two conditions and minimises the objective function $J_1(\mathbf{U}, \mathbf{V})$ by alternatively updating matrices $\mathbf{V}$ and $\mathbf{U}$ using equations (5.3.1) and (5.3.2). In this sense, HCMCA can be generalised as an iterative procedure of *alternate minimisation* which is widely adopted in machine learning. At the first step of each iteration, such a procedure fixes parametric model $M_1 = M_1|_{\text{old}}$ and gets parametric model $M_2|_{\text{new}} = \arg\min_{M_2} F(M_1, M_2)$, where function $F(M_1, M_2)$ is the objective function of both M_1 and M_2 . And then at the second step of each iteration, alternately, the procedure fixes $M_2 = M_2|_{\text{old}}$ and gets $M_1|_{\text{new}} = \arg\min_{M_1} F(M_1, M_2)$. The alternate minimisation procedure guarantees to reduce the objective function $F(M_1, M_2)$ until it converges to a local minimum.

For procedures implementing HCMCA, model $M_1 = \mathbf{V}$, model $M_2 = \mathbf{U}$ and the objective function $F(M_1, M_2) = J_1(\mathbf{U}, \mathbf{V})$. And it is traditional to let c examples randomly chosen from the data set serve as the initial set of cluster centres.

HCMCA was introduced by Lloyd [71], which inspired many variations, such as the use of the Mahalanobis distance [72] instead of the Euclidean distance for equations (5.2.4) and (5.3.2), or the use of fuzzy measures [73, 74] to allow overlapped clusters. HCMCA and its variations are generally called the c-means methods since they all use the centre to represent a cluster. The c-means methods are essentially calculus-based, where cluster centres tend to move in the directions that the gradient of the objective function descends most. As hill-climbing techniques, the c-means methods display high efficiency and have been effectively applied in different areas. They may be used to generate a representative set of prototypes for a data set, often in conjunction with other pattern recognition schemes. For example, c-means may be used to generate a set of vectors to be used as centres in a radial basis function (RBF) classifier [75]. Vector quantisation [76] is an application of the c-means procedures. In a texture classification problem, a c-means method has been used to obtain a reduced set of samples characterising each class of texture [77]. c-means for speech coding has been discussed in [78].

Although the c-means methods are efficient, they have a common inevitable drawback. The next section discusses this drawback in the context of HCMCA.

5.4 Local Search - The Crucial Problem of c-Means Clustering

As already mentioned, HCMCA is a calculus-based method. Therefore, it is efficient and powerful in local optimisation. However, as stated in [71],

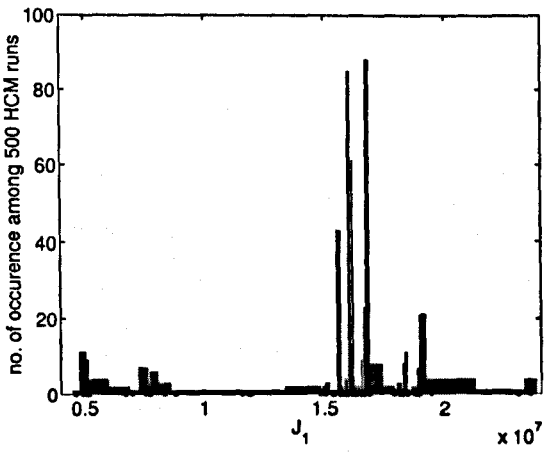


Figure 5.1: Distribution of the J_1 values associated with the partitions found when classifying Chernoff faces using HCMCA (see section 7.3 for a description of the experiment).

the two conditions defined by equations (5.3.1) and (5.3.2) are necessary but not sufficient conditions for the existence of the optimal partition. In other words, more than one partition (local extremum) may satisfy equations (5.3.1) and (5.3.2). Moreover, the objective function HCMCA tries to optimise is typically a highly non-linear function. As often reported in the literature, the hill-climbing HCMCA is fairly sensitive to initial conditions and can be easily trapped in a local optimum that is far away from the global one. Figure 5.1 plots the distribution of the J_1 values associated with the partitions found when applying HCMCA to a real-world clustering problem - the Chernoff faces data set [79]. As we can see, the distribution is broad and thus indicates the local optimality of these solutions and reveals the sensitivity of HCMCA to the initial conditions. Moreover, the majority of the partitions found for this application problem using HCMCA causes a J_1 value (i.e. distortion) much higher than the optimal one. To tackle these problems, a novel algorithm has been proposed, which adopts the stochastic nature of genetic algorithms and thus performs a global optimisation of the clustering objective function $J_1(\mathbf{U}, \mathbf{V})$.

5.5 Conclusions

This chapter has been devoted to an introduction to the problem of hard clustering and the common used clustering algorithm - HCMCA. The formulation of the problem and the method has been presented. The inevitable drawbacks of c-means have been discussed in the context of HCMCA, which urges the employment of a more advanced searching method, such as the genetic algorithms. Before presenting the new genetically guided clustering algorithm, we devote the next chapter to an introduction to the genetic algorithms.

Chapter 6

FUNDAMENTALS OF GENETIC ALGORITHMS

6.1 Introduction to Evolutionary Algorithms

The family of *evolutionary algorithms* (EAs) is a collection of optimisation algorithms that are inspired by natural evolutionary phenomena and derived from simulating Darwinian evolutionary theory.

Darwinian evolution is essentially a robust search and optimisation mechanism. Evolved biota demonstrate optimised complex behaviour at every level: the cell, the organ, the individual and the population. The problems that biological species have solved are typified by chaos, chance, temporality, and nonlinear interactivities. These are also characteristics of problems that have proved to be intractable to classic methods of optimisation. The evolutionary process can be applied to problems where heuristic solutions are not available or generally lead to unsatisfactory results. As a result, EAs have recently received increased interests, particularly with regard to the manner in which they may be applied for practical problem solving. EAs are classified as *stochastic*[1]

[1]The searching procedure by a genetic algorithm can be formulated as a finite-dimension Markov chain.

search algorithms for *global*[2] optimisation problems and have found many engineering and industrial applications (see, for example, [23] - [25], [80] - [84]). The massive scale of research activities concerning EAs undertaken world-wide in recent years symbolises a systematic knowledge transition from life science and human science through mathematics and computer science to engineering technology.

6.2 Types of Evolutionary Algorithms

In the area of evolutionary computation, there are three broadly similar branches: *Genetic Algorithm* (GA), *Evolution Strategy* (ES), and *Evolutionary Programming* (EP). All of these three optimisation techniques maintains a population of trial solutions, imposes random changes to them, and incorporates selection. But they also have important differences. These differences are philosophical and due to their different beliefs in biology.

In the field of biology, there are diametrically opposed opinions as to what exactly is being evolved. Davis [85] (p. 2) suggests that "evolution is a process that operates on chromosomes rather than on the living beings they encode." In sharp contrast, Mayr [86] (p. 162-163) defined that "evolution is change in the adaptation and in the diversity of populations of organisms." In brief, Davis advocates evolution on genotype while Mayr advocates evolution on phenotype.

Following the philosophy of Davis, GAs represent each trial solution as genes along a chromosome and impose genetic operators on these genes. GAs emphasise the reductionist, bottom-up assembly of *building blocks* - gene sequences with high-performance. New generations of gene strings are primarily created via combined process of selection and recombination. Whereas, in ES

[2]Unlike the traditional optimisation methods, evolutionary algorithms are less likely to be trapped on the local optima.

and EP, the components of a trial solution are viewed as behavioural traits of an individual. Each new generation of these phenotypic traits is created by adding a Gaussian random variable. ES and EP emphasise this mutational transformation which maintain behavioural linkage between each parent and its offspring, respectively at the level of the individuals or the species - sub-populations of individuals.

GAs emphasise the overt genetic attributes of gene recombination (more widely known as *crossover*) and have defined it as the most distinguishing feature. The application of recombination operations to purely genetic information in ES is logically sound because this process operates on simulated individuals. Whereas, the application of recombination in EP is not applicable because each solution is typically viewed as the analog of a species, and there is no sexual communication between species.

6.3 Advantages of Evolutionary Computation

Evolutionary computation - the term now used to describe the field of investigation that concerns all evolutionary algorithms - offers practical advantages to the researchers facing difficult optimisation problems. These advantages are multifold, including the simplicity of the approach, its flexibility, its superiority to classic techniques, its parallelism and many other aspects.

Conceptual simplicity

A primary advantage of evolutionary computation is that it is conceptually simple. The main flow chart that describes every EA applied for function optimisation is shown in Figure 6.1. The algorithm consists of initialisation, where a population of candidate solutions to the problem at hand is initialised, followed by iterative fitness-based selection and variation. Competing solutions

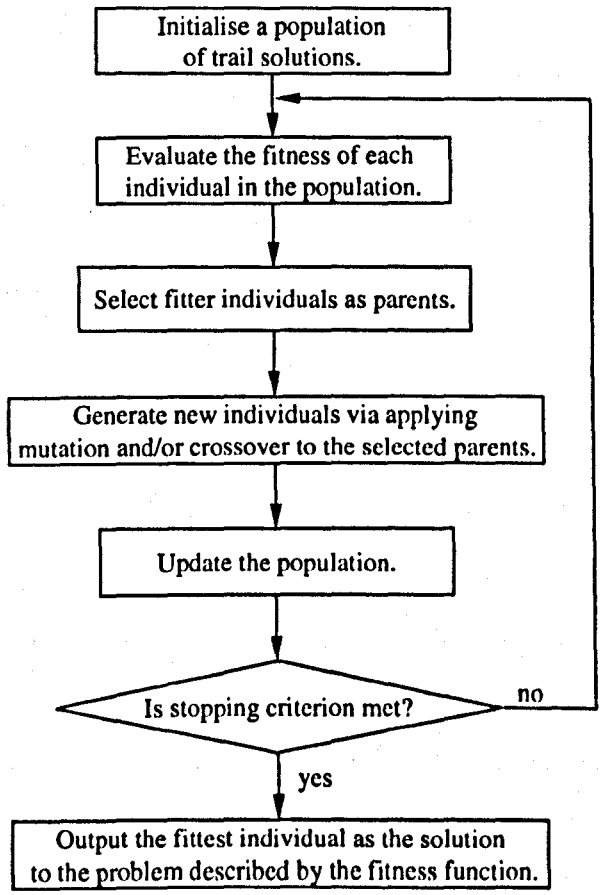


Figure 6.1: The main flow chart of the vast majority of evolutionary algorithms.

are evaluated according to a *fitness* (or performance index) such that two competing solutions can be rank-ordered. Finer granularity is not required. Thus the criterion need not be specified with the precision that is required of some other methods. In particular, no gradient information needs to be presented to the algorithm. Fitness-based selection is applied to determine which solutions will be maintained into the next generation, and with what frequency. These selected parents are then subjected to random variation, including mutation and/or recombination, and the process iterates. Over iterations of selection and random variation, the population is expected to converge to optimal solutions [87, 88].

Broad applicability

EAs can be applied to actually any problem that can be formulated as a function optimisation task. It requires a data structure to represent solutions, a performance index to evaluate solutions, and variation operators to generate new solutions from old solutions. The state space of possible solutions can be disjoint and can encompass infeasible regions, and the performance index can be time varying, or even a function of competing solutions extant in the population. The human designer can choose a representation that follows their intuition. In this sense, the procedure is representation-independent, in contrast with other numerical techniques which might be applicable for only continuous values or other constrained sets. This flexibility allows for applying essentially the same procedure to discrete combinatorial problems, continuous-valued parameter optimisation problems, mixed-integer problems, etc.

Superiority over classical methods on real problems

For simple problems, where the response surface is, say, strongly convex, EAs do not perform as well as traditional optimisation methods [89]. But this is to be expected as these techniques were designed to take advantage of the convex property of such surfaces. However real-world function optimisation problems

often (1) impose nonlinear constraints, (2) require payoff functions that are not concerned with least square error, (3) involve nonstationary conditions, (4) incorporate noisy observations or random processing, or include other vagaries that do not conform well to the prerequisites of classical optimisation techniques. Further, the response surface posed in real-world problems are often multi-modal, and gradient-based methods rapidly converge to local optima (or perhaps saddle points) which may yield insufficient performance. Schwefel [90] has shown in a series of empirical comparisons that in the alternate condition of applying classical methods to multi-modal functions, EAs offer a significant advantage.

Moreover, the problem of defining the payoff function for optimisation lies at the heart of success or failure: inappropriate descriptions of the performance index lead to generating the right answer for the wrong problem. Within classical statistical methods, concern is often devoted to minimising the squared error between forecast and actual data. But in practice, equally correct predictions are not of equal worth, and errors of identical magnitude are not equally costly. Consider the case of correctly predicting that a particular customer will purchase 10 units of a product. This is typically worth less than correctly predicting that the customer will purchase 100 units of that product, yet both predictions cause zero error and weighted equally in classical statistics. Furthermore, the error of predicting the customer will demand 10 units and having him actually demand 100 units is not of equal cost to the manufacture as predicting the customer will demand 100 units and having him demand 10. Yet again, under a squared error criterion, these two situations are treated identically. In contrast, within EAs, any definable payoff function can be used to compare alternative behaviours. There is no restriction that the criteria should be differentiable, smooth, or continuous.

Potential to use knowledge and hybridise with other methods

It is always reasonable to incorporate domain-specific knowledge into an algorithm when addressing particular real-world problems. Specialised algorithms can outperform unspecialised algorithms on a restricted domain of interest [91]. EAs offer a framework where it is comparably easy to incorporate such knowledge. For example, specific variation operators may be known to be useful when applied to particular representations (e.g. partially matched crossover for travelling salesman problem [92]).

EAs can also be combined with more traditional optimisation techniques. This may be as simple as the use of a conjugate-gradient minimisation applied after the primary search with an EA (e.g. [93]), or it may involve simultaneous applications of algorithms (e.g. the use of evolutionary search for the structure of a model coupled with gradient search for parameter values [94]). Further, evolutionary computation can be used to optimise the performance of neural networks [95], fuzzy systems [96], production systems [97], and other program structures [98, 99]. In many cases, the limitations of conventional approaches (e.g. the requirement for differentiable hidden nodes when using back propagation to train a neural network) can be avoided.

Parallelism

Evolution is a highly parallel process. It is often the case that individual solutions can be evaluated independently of the evaluations assigned to competing solutions. The evaluation of each solution can be handled in parallel and only selection (which requires at least pairwise competition) requires some serial processing. As distributed processing computers become more readily available, there will be a corresponding increased potential for applying evolutionary algorithms to more complex problems.

Ability to solve problems that have no known solutions

Perhaps the greatest advantage of EAs cames from the ability to deal with

Table 6.1: Implementation steps of simple genetic algorithms.

Initialise a population of gene strings.
REPEAT
Evaluate each string in the population.
Based on fitness, select pairs of parents from the current population.
Generate offspring of the selected parents via crossover and mutation.
Replace the parents with their offspring.
UNTIL the specified number of generation is reached.

problems for which there are no human experts. Although human expertise should be used when it is available, it often proves less than adequate for automatic problem-solving. Troubles with such expert systems are well known: the experts may not agree, may not be self-consistent, may not be qualified, or may simply be in error. Research in artificial intelligence has broken into a collection of methods and tricks for solving particular problems in restricted domains of interest. Certainly, these methods have been successfully applied to specific problems (e.g. the chess program Deep Blue). But most of these applications require human expertise. They may be impressively applied to difficult problems requiring great computational speed, but they generally do not advance our understanding of intelligence. "They solve problems, but they do not solve the problem of how to solve problems," ([100], p. 259). In contrast, evolution provides a method for solving the problem of how to solve problems. It is a recapitulation of the scientific method [101] that can be used to learn fundamental aspects of any measurable environment.

6.4 Genetic Algorithms (GAs)

Since the work reported in this thesis has related mainly to genetic algorithms (GAs), the rest of this chapter will focuses on a systematic discussion of GAs. Table 6.1 shows the implementation steps of the canonical or simple genetic algorithm (SGA), which illustrates the central ideas in the operation of GAs.

6.4.1 Types of GAs

Besides the SGA described in Table 6.1, there are a number of significant variations. There are the elitist, steady-state, and deme GAs, they all differ from the simple GA in the way that they treat the population of individuals from generation to generation.

Elitism is a simple extension to SGA that ensures that the best individual persists from generation to generation. The steady-state GA also allows individuals to persist from one generation to the next, usually by keeping the best individuals and replacing the worst. It was originally developed by Whitley [102]. The algorithm generates a single individual at each generation, which then replaces a selected member of the population (usually the worst). Syswerda [103] describes a similar algorithm that is the same except that instead of a single individual multiple new genomes (the number of newborns is set via a parameter called generation gap) are created each generation. The advantage of these approaches is that the population always contains of the best solutions discovered by the algorithm so far. Whereas, under SGA it is possible that the algorithm "loses" important genetic material due to a good individual not being selected or because of the subsequent crossover and mutation.

The deme GA is significantly different. Instead of a single population of individuals, a number of sub-populations are evolved independently. Occasionally, the better individuals from a sub-population are selected for migration,

which involves transferring them to another sub-population. The advantage of this technique is that it allows for the easy parallelization of the GA through cluster computing techniques. Sub-populations can be evolved on separate machines and communicate the migrating individuals by a network.

6.4.2 Fitness scaling

Before individuals are selected from the population to generate offspring, the raw score obtained from calling the objective function is converted into a fitness score. There are many approaches to doing this. The simplest is no scaling where the objective score is used directly as the fitness value. This has the disadvantage that a few good individuals may dominate the population within a few generations (i.e. too much selection pressure); or there is too little difference between individuals and so they all have an equal chance of being selected (i.e. too little selection pressure). If the later problem exists then a good alternative is linear scaling where better individuals get a proportionally higher fitness. A good general-purpose scheme is rank-based scaling [102], where fitness is assigned based on the rank of an individual within a population that is ordered in light of objective score.

6.4.3 Selection schemes

After the fitness evaluation, an expected value of offspring is assigned to each individual extant in the population based on its fitness relative to the others. This can be expressed mathematically as

$$ev_x = \frac{\text{fit}(x)}{\sum_{x \in \mathbf{pop}} \text{fit}(x)} \times |\mathbf{pop}| \tag{6.4.1}$$

where ev_x denotes the expected value of individual x, $\text{fit}(x)$ evaluates the fitness value of individual x and **pop** is the set composed of individuals in current population. Then the phase of selection determines the actual number of off-

spring each individual will receive. Baker [104] stated that "the algorithm used to convert the real expected values to integer numbers of offspring is called the sampling algorithm." There have been various sampling algorithms reported. The well-known schemes are stochastic sampling with replacement, stochastic sampling with partial replacement, remainder stochastic sampling with replacement, remainder stochastic sampling without replacement, deterministic sampling, remainder stochastic independent sampling, and stochastic universal sampling.

Stochastic sampling with replacement is actually a fancy name of the first widely-used selection scheme, the *roulette wheel selection.* It is named so because it is analogous to a gambler's roulette wheel with each wheel slice proportional in size to the expected offspring value of an individual.

Stochastic sampling with partial replacement is another name for De Jong's expected-value model [105]. In this algorithm, an individual's expected value is decreased by 1.0 each time it is chosen by the roulette wheel selection.

A *remainder sampling method* involves two distinct phases. In the integral phase, samples are awarded deterministically based on the integer portions of the expected values. The fractional phase then samples according to the expected values' fractional portion. In *remainder stochastic sampling with replacement,* the fractional portions are sampled by the roulette wheel method. The individual's fractions remain unaltered between spins, and hence continue to compete for selection. While in *remainder stochastic sampling without replacement,* after each spin, the selected individual's expected value is set to zero.

A *deterministic sampling* algorithm is suggested and used by Brindle [106]. In this remainder algorithm's fractional phase, the individuals with the largest fractions are selected.

Remainder stochastic independent sampling and stochastic universal sampling were proposed by Baker (1987) [104].

In *remainder stochastic independent sampling*, the fractional portions are treated as the probabilities of successive Bernoulli trials without replacement (since once an individual is selected, its expected value is set to zero). Baker suggested to perform the fractional phase of this algorithm in full parallel, precede the $O(N)$ sequential integral phase.

Baker [104] described *stochastic universal sampling* as being analogous to a roulette wheel with N equally spaced pointers and gave a C code Fragment for it as follows (cited from [104]):

```
ptr = Rand();
for (sum=i=0; i<N; i++)
   for (sum += ExpVal[i]; sum > ptr; ptr++)
      SelectInd(i);
```

Stochastic universal sampling is a simple, single phase, $O(N)$ sampling algorithm. However it is strictly sequential.

An individual's expected offspring value is not always equal to its actual sampling probability. To evaluate various sampling algorithms, Baker [104] introduced three measures: bias, spread and efficiency. The bias indicates the algorithm's accuracy; while the spread indicates the precision. Hence the spread reveals the sampling algorithm's consistency. Table 6.2 summarises the basic characteristics of the various sampling algorithms, where N denotes the population size, ev is the expected value of an given individual, $\lfloor x \rfloor$ and $\lceil x \rceil$ rounds a floating point value x to the nearest integer towards minus and plus infinity respectively, and integer R satisfies $\lfloor ev \rfloor + R \leq N$.

Based on the theoretical analysis summarised in Table 6.2 and the empirical analysis of the conducted bias, Baker [104] concluded that the stochastic universal sampling algorithm is an optimal sequential sampling algorithm, which for the first time, assigns offspring according to the theoretical specifications (equation (6.4.1)) and hence recommended that in sequential environments GAs should employ this algorithm; while if a parallel environment is available, remainder stochastic independent sampling may prove valuable.

Sampling Method	Bias	Spread	Computational Load	Parallelability
Stochastic Sampling with Replacement	zero	unlimited $0 \rightarrow N$	$O(N \log N)$	None
Stochastic Sampling with Partial Replacement	medium	upper bounded $0 \rightarrow \lceil ev \rceil$	$O(N \log N)$	None
Remainder Stochastic Sampling with Replacement	zero	lower bounded $\lfloor ev \rfloor \rightarrow \lfloor ev \rfloor + R$	$O(N \log N)$	None
Remainder Stochastic Sampling without Replacement	medium	minimum $\lfloor ev \rfloor, \lceil ev \rceil$	$O(N \log N)$	None
Deterministic Sampling	high	minimum $\lfloor ev \rfloor, \lceil ev \rceil$	$O(N \log N)$	None
Remainder Stochastic Independent Sampling	low	minimum $\lfloor ev \rfloor, \lceil ev \rceil$	$O(N)$	Fractional Phase
Stochastic Universal Sampling	zero	minimum $\lfloor ev \rfloor, \lceil ev \rceil$	$O(N)$	None

Table 6.2: Comparison of Sampling Methods (cited from [104]).

6.4.4 Crossover operators

For optimisation of a function with real valued parameters, there have been three well-defined crossover operators: one-point crossover, two-point crossover and uniform crossover.

Under *one-point crossover*, a crossing site is selected at random and then the sub-strings that follow the site are exchanged. As indicated by its name, *two-point crossover* selects two crossing sites, again at random. Then the alleles in the segments delimited by the sites are exchanged, considering the chromosomes circular. For example, two possible parents could be

Parent1:	A	B	C	D	E
Parent2:	a	b	c	d	e

One-point crossover could possibly yield the following offspring for the parents above (with the bar indicating the crossing site):

Child1:	A	b	c	d	e
Child2:	a	B	C	D	E

and 2P could yield offspring as follows:

Child1:	A	b	c	D	E
Child2:	a	B	C	d	e

Since one-point crossover introduces an undesirable bias with respect to the extremities of the chromosome, widely used is the next lowest-disruption choice - the two-point crossover. However two-point crossover is a low-disruption crossover. Further, it introduces an extra bias: short schemata are disrupted less often than long ones. A *schema* is a similarity template describing a subset of strings with similarities at certain string positions. The trouble is that we usually do not know a priori which schemata should be the least disrupted [107].

Very different from one-point and two-point crossover operators, *uniform crossover* constructs a new string by randomly copying, for each locus, the

allele randomly from a parent. This is achieved by constructing a binary *mask*, one bit per locus, typically with equal probability for the alternative values, as in the following example:

Parent1:	A	B	C	D	E
Parent2:	a	b	c	d	e
Mask:	0	1	0	0	1
Child1:	a	B	c	d	E
Child2:	A	b	C	D	e

The first child inherit the allele from the first parent where there is a 1 in the mask, and from the second parent where there is a 0. The second child is constructed according to the opposite rule. Since the mask's values at different loca are set independently, uniform crossover disregards any linkage there might be among genes on a string.

Syswerda examined the utility of various crossover operators, concentrating on one-point, two-point, and uniform crossover. Syswerda [103] emphasised that the overt purpose of crossover is to combine good schemata together into one genome and studied several binary function optimisation problems. The empirical results did show that generally uniform crossover yielded better performance than two-point crossover, which in turn yielded better performance than one-point crossover.

Since its popularisation by Syswerda in 1989, uniform crossover has become perhaps the most widely used crossover operator. However, the worth of uniform crossover was questioned by Falkenauer in 1999 [107]. In his paper, Falkenauer discussed the two popular arguments backing uniform crossover, and gave corresponding counterarguments.

The arguments in favour of uniform crossover and the corresponding counterarguments by Falkenauer are summarised as follows:

1. Uniform crossover is capable to reach in a single step any point in the search space, provided that each bit of the corresponding gene string is

present in at least one of the parents.

Falkenauer: Random search is equally capable to reach in a single step any point in the search space (regardless of the parents). However, few researchers would hail random search as an efficient search method.

2. Uniform crossover has a better recombining power than one-point and two-point crossovers.

   Falkenauer: The disregard of gene linkage by uniform crossover and its outperformance than low-disruption operators indicates that genes must be largely independent from each other. However, such a situation also implies that the alleles of those genes can be set separately (i.e. the problem is separable), suggesting a hill-climber as a better search device for the problem. Conversely, when a problem is difficult for a simple hill-climber (i.e. the problem is non-separable), and where GA becomes a device of interest, uniform crossover should be outperformed by a low-disruption crossover.

Based on his arguments, Falkenauer concluded that "uniform crossover's high rate of disruption is harmful" and baked his conclusion with experimental results on a simple function that is "mildly non-separable (there is linkage between genes on the chromosome)". Nevertheless, Spears and De Jong [108] indicated that disruptiveness need not always be viewed negatively. In the situation of premature convergence, it may be the only mechanism for advancing the search. While the experiments of Syswerda [103] and others did indicate the potential for taking a positive view of disrupting schemata when implementing a genetic algorithm although the usefulness of such disruption will be highly problem-dependent.

6.5 Basic Theorems of GAs

To understand how a GA acts, it seems that we need to look at the raw data available for any search procedure. While it turns out that we can search more effectively by exploiting similarities in the encoded strings. This is the concept of *similarity template* or schema, which leads to a keystone in GA theory - the building block hypothesis.

GAs inherently process a large quantity of schemata while processing a relatively small quantity of gene strings. It turns out that the number of schemata processed usefully in each generation is something like N^3, where N is the population size. This important processing leverage is apparently unique to GAs, and is called the *implicit parallelism.*

The effects of the three fundamental GA operators have been analysed separately regarding how schemata are processed.

Comprehensively, after the execution of the three basic GA operators, the number of a particular schema H, $m(H)$, remains in the next generation can be given by the following equation [81]

$$m(H,t+1) \geq m(H,t) \cdot \frac{f(H)}{\bar{f}} \left[1 - p_c \frac{\delta(H)}{l-1} - o(H) p_m \right] \tag{6.5.1}$$

where p_c and p_m are the probabilities of operations crossover and mutation, $o(H)$, $\delta(H)$ and $f(H)$ are respectively the order, the defining length and the (average) fitness of a schema H.

In equation (6.5.1), the former variation factor is due to the effect of fitness-based selection; and the later factor indicates that short, low-order schemata have more chances to survive under crossover and mutation. The introduction of a lower bound is because the theoretical derivation contains no source terms and it has been assumed that we lose the schema whenever a crossover occurs between the schema's outermost defining bits, while in practice sometimes one schema's loss is another's gain.

Assume that the particular schema H remains above average an amount $c\bar{f}$ with a constant c. Under this assumption, rewrite the differential equation (6.5.1) as

$$\begin{aligned} m(H,t+1) &\geq m(H,t)\frac{(\bar{f}+c\bar{f})}{\bar{f}}\left[1-p_c\frac{\delta(H)}{l-1}-o(H)p_m\right] \\ &\geq (1+c)\cdot m(H,t)\cdot\left[1-p_c\frac{\delta(H)}{l-1}-o(H)p_m\right] \end{aligned} \tag{6.5.2}$$

Starting at $t=0$ and assuming a stationary value of c, we have

$$m(H,t)\geq m(H,0)\cdot\left[1-p_c\frac{\delta(H)}{l-1}-o(H)p_m\right]\cdot(1+c)^t \tag{6.5.3}$$

According to this calculated result, an important conclusion can be drawn [109] (pp. 102-103): "Short, low-order, above-average schemata (i.e. *building blocks*) receive *exponentially* increasing trials in subsequent generations." This is the well-known *schema theorem*. This allocation strategy is implemented mainly by the selection operation in GAs and has been justified by a two-armed bandit problem [81].

In the light of the schema theorem, the *building block hypothesis* explains the power of the GA: "Instead of building high-performance strings by trying every conceivable combinations, GAs work robustly by identifying good building blocks and by eventually combining these to get larger building blocks." [81]

6.6 Premature Convergence in the GA Search

Premature convergence is an important concern in GAs. This occurs when the population reaches a configuration such that crossover no longer produces offspring that can outperform their parents, as must be the case in a homogeneous population where population diversity equals zero. Under such circumstance, all standard forms of crossover simply regenerate the current parents.

Any further optimisation relies solely on bit mutation and can be quite slow. Suppose the population prematurely converges at a local extremum b bits away from the global optimum. The probability of flipping these b specific bits in a single binary chromosome and not flipping any others is $p_m^b(1-p_m)^{k-b}$, where p_m is the probability of flipping a single bit and k is the bit length of the genome. Premature convergence is often observed in GA research ([85], [105], and others) because of the exponential reproduction of the best-observed solution coupled with the strong emphasis on crossover.

Therefore, for real-world optimisation problems requiring great precision and a long binary coding, it may be expected (although there will be counterexamples) that a GA that does not employ a heuristic method for preventing or postponing premature convergence will not tend to discover even nearly globally optimal solutions in a reasonable number of generations. Instead, the population will settle at a point (or points) on the response surface, after which further optimisation will result only if there is an unlikely flipping of the correct bits.

Davis [85] recommends that when the population converges on a chromosome that would require the simultaneous mutation of many bits to improve it, the run is practically completed and either it should be restarted using a different random seed (e.g. dynamic parameter encoding [110]) or hill-climbing heuristics should be employed to search for improvements.

6.7 Conclusions

This chapter has been devoted to an introduction to evolutionary algorithms, especially to genetic algorithms. The advantages shared by all the evolutionary algorithms are presented. Types of GAs and GA operators are compared with each other. The theorems revealing the reasons of GA's success and the premature convergence problem of GA are addressed. The robustness

of GAs is demonstrated in the next two chapters through the application study in clustering and job-shop scheduling, respectively.

Chapter 7

A GENETIC HARD C-MEANS CLUSTERING ALGORITHM

7.1 Introduction

As already pointed out, by hill climbing, the traditional HCMCA is found to be easily trapped in local optima and very sensitive to initial conditions. A way to avoid local extrema and also reduce the sensitivity to initialisation is to use a stochastic optimisation approach, such as GAs. The advantages of applying GAs to clustering problems has been investigated [111] - [114]. However, as stated in [112], a GA clustering approach takes up to two orders of magnitude more time than HCMCA. Considering this, a new hybrid algorithm is developed which exploits both the efficiency of HCMCA as a hill-climbing technique and the stochastic nature of a GA. This hybrid algorithm introduces a one-step local search using HCMCA into each GA generation. Thus the new algorithm is called genetic hard c-means clustering algorithm (GHCMCA). Unlike previous genetic clustering algorithms, the population members of GHCMCA represents partition matrices instead of sets of cluster centres. A new genetic crossover has been adopted. It effectively exchanges important partition similarities between

a pair of partitions, creating new solutions. GHCMCA has been evaluated in comparison with the traditional HCMCA as well as a previous genetic clustering approach - *genetically guided clustering algorithm* (GGA - abbreviation used in the original paper [112]). Four data sets were used as a benchmark in the experiments. Results show that the genetic approaches may provide a viable way to avoid the local extrema and reduce the influence of initialisation. Compared to GGA, GHCMCA converges to the global optimum more quickly and with greater probability and thus shows its superiority.

7.2 The Algorithm

To overcome the inevitable drawbacks of a hill-climbing technique such as HCMCA, the genetic algorithm (GA) may be adopted. Starting with an initial condition, a GA evolves a population towards successively better regions in the search space by means of genetic processes of *selection*, *crossover* and *mutation*. The given optimisation problem defines an environment that delivers quality information (*fitness values*) for new search points, and the selection process favours those population members with higher quality to reproduce more often than the worse members. The crossover mechanism allows for the mixing of parental information while passing it to the offspring, and mutation introduces innovation into the population and prevent premature loss of important information.

In order to apply a genetic approach to a given problem, a number of fundamental issues must be addressed in advance. The rest of this section describes, in detail, each of these issues with respect to the new genetic clustering algorithm - GHCMCA.

7.2.1 Solution representation and initialisation

Unlike previous genetic clustering algorithms [111] - [114], the population members under GHCMCA represent partition matrices instead of sets of cluster centres. In hard clustering, any object point belongs to the closest cluster exclusively. There is only a 1 down any column of a hard partition matrix. It is possible to simplify a $c \times n$ hard partition matrix $\mathbf{U}$ into an n-dimensional vector $\vec{u}$ where the ith element describes which row the 1 lies down the ith column of the original $\mathbf{U}$. The possible values of the elements of $\vec{u}$ range from 1 to c. GHCMCA has adopted this simplification.

Consider that good choice for starting configurations should be free of overt biases. For the hard partition vectors in the initial population, each element is set to a randomly generated number in the range of $[1, c]$. By doing so, we actually partition the object points to c initial clusters uniformly at random.

7.2.2 Fitness evaluation

The goal of a clustering approach is to minimise the cost function $J_1(\mathbf{U}, \mathbf{V})$. While a GA is inspired by natural evolution and favours fitter population members. To compromise, we use the inverse of $J_1(\mathbf{U}, \mathbf{V})$ function as the fitness function of GHCMCA. And in order to reduce the chance of GHCMCA becoming stuck at a degenerate partition, we have also taken the number of empty clusters into consideration. The final fitness function is defined as follows

$$\text{fitness}(\mathbf{U}) = \frac{1}{J_1(\mathbf{U}, \mathbf{V}) \times (1 + e/c)} \tag{7.2.1}$$

where c is the total number of clusters and $0 \leq e \leq c$ denotes the number of empty clusters and may be evaluated via counting the all-0 rows in $\mathbf{U}$. Cluster centres $\mathbf{V}$ are located using equation (5.3.1). To prevent premature convergence and maintain reasonable selection pressure, a typical linear fitness scaling mechanism [81] has been adopted such that the fitness and the con-

sequent surviving probability of the best population member is C_{mult} times as much as those of the average population member. As suggested in [81], a $C_{mult} = 1.5$ is used in GHCMCA.

7.2.3 Genetic operators

In every generation, a GA selects parents from the current population to form a mating pool. Theoretically, the probability of a member being selected is proportional to its fitness value relative to the others' fitness values. After being selected, parents are mated at random to give birth to their offspring. For each pair of parents, offspring are generated via the operations of crossover and mutation.

a. Selection. As to the selection operator, the *stochastic universal sampling* (SUS) scheme [104] is adopted. It is a sequential sampling scheme. In [104], based on the theoretical and empirical analysis, Baker concluded that this scheme selects parents at rates with zero bias to the expected values and the minimal spread.

b. Crossover. GAs emphasis the overt genetic attributes of crossover. Following the crossing-over during cell meiosis in the natural system, the crossover operator in a GA is responsible for genetic recombination. Here, a new crossover operator has been devised. It reassigns the worst members of each cluster in a parent's partition according to its mating partner's partition. The worst member of a certain cluster is the object point (among those belonging to the same cluster) that is farthest from to the centre of that cluster.

For the ith cluster in parent$_1$'s partition $\vec{u}_1$, the new crossover is carried out as follows:

1. Identify the worst $\eta\%$ members of the ith cluster of $\vec{u}_1$;

2. FOR each of the worst $\eta\%$ members - $\vec{x}_k$,

(a) according to the mating partner's partition vector $\vec{u}_2$, find out to which cluster $\vec{x}_k$ belongs (suppose it belongs to the jth cluster of $\vec{u}_2$);

(b) still according to $\vec{u}_2$, randomly choose another object point from the jth cluster of $\vec{u}_2$ (suppose it happens to be $\vec{x}_l$);

(c) back to $\vec{u}_1$, check out to which cluster of $\vec{u}_1$ the chosen object point $\vec{x}_l$ belongs and then simply reassign $\vec{x}_k$ to that cluster;

where $\eta\%$ denotes the percentage of object points to be reassigned in each cluster. It controls how significantly the offspring will be different from their parents. In all our experiments with GHCMCA, $\eta\%$ is fixed at 20%. While still holding the randomness property, this crossover operator is well guided. At step 2(a), the mating partner's partition $\vec{u}_2$ is referred to answer the following question: which object points should share the same cluster with the current worst member? Since there is at least one (itself) while may be more than one object points available, step 2(b) randomly choose one of them. Finally, at step 2(c), the current worst member of the ith cluster of partition $\vec{u}_1$ is reassigned to a new cluster such that it belongs to the same cluster as the randomly chosen object point does. For better understanding, implementation of steps 2(a) to 2(c) are described graphically in Figure 7.1, where a certain worst member of cluster i is reassigned to cluster i'. The whole procedure is applied to each parent independently. Each implementation generates an offspring. We have named this new operator *partial crossover* (PX), since the fact that under it important partition information missed in a parent may be regained from its partner and consequently the resulting offspring contains partition information partially inherited from both parents.

To see this, consider the following two partitions

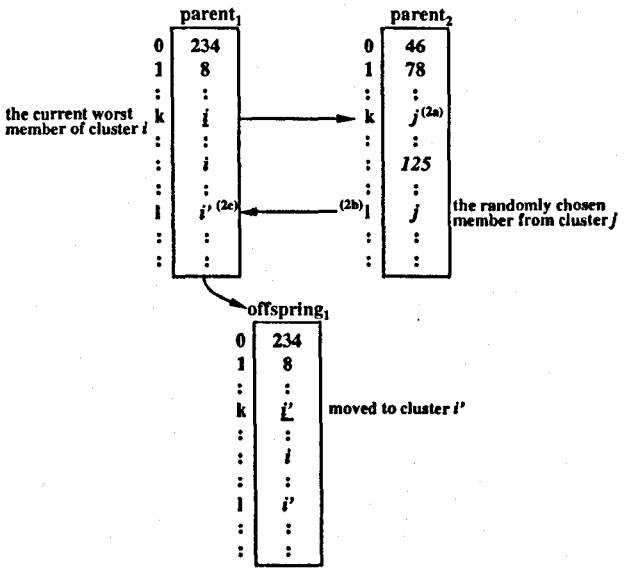


Figure 7.1: Graphical description of the new crossover operator when it is applied to a certain worst member of the ith cluster of parent$_1$.

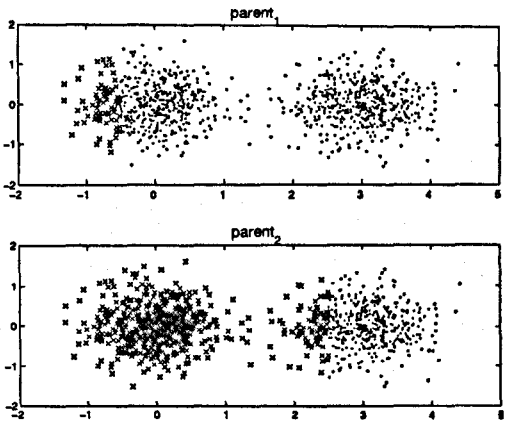
$$
\begin{array}{rccccccccccccccc}
 & 1 & 2 & \ldots & & k & .. & k' & \ldots & j' & .. & j & \ldots & & n \\
\vec{u}_1 = [& 0 & 0 & \ldots & 0 & 1 & .. & 1 & \ldots & 1 & .. & 1 & \ldots & & 1], \\
\vec{u}_2 = [& 0 & 0 & \ldots & & 0 & .. & 0 & \ldots & 0 & .. & 0 & 1 & \ldots & 1],
\end{array}
$$

where integers above the partition vectors index the object points. Both $\vec{u}_1$ and $\vec{u}_2$ define a viable way to partition a set of objects into two clusters, as shown in Figure 7.2(a), respectively. A possible pair of offspring obtained using PX is as follows

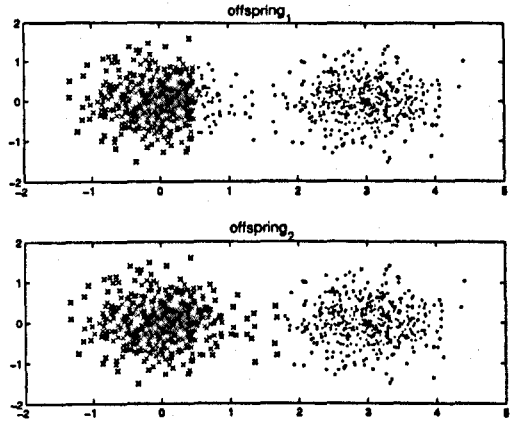
$$
\begin{array}{rccccccccccccccc}
 & 1 & 2 & \ldots & & k & .. & k' & & \ldots & & j' & .. & j & \ldots & & n \\
\vec{u}_1' = [& 0 & 0 & \ldots & 0 & 0 & .. & 0 & 1 & \ldots & 1 & 1 & .. & 1 & \ldots & & 1], \\
\vec{u}_2' = [& 0 & 0 & \ldots & & 0 & .. & 0 & 0 & \ldots & 0 & 1 & .. & 1 & 1 & \ldots & 1].
\end{array}
$$

The corresponding partitions are depicted in Figure 7.2(b). Obviously, by exchanging good partition information both of these two new offspring cause less distortion than their parents.

According to the example above, it seems that when reassigning an object $\vec{x}_k$ in partition $\vec{u}_1$ according to its mating partner $\vec{u}_2$, instead of applying steps



(a) before partial crossover



(b) after partial crossover

Figure 7.2: An example for demonstrating the new partial crossover.

2(a) to 2(c), we can simply set $\vec{u}_1(k)$ to the value of $\vec{u}_2(k)$. However, this is not feasible in general. Consider the situation where $\vec{u}_1$ and $\vec{u}_2$ define the same partitions but with different labelling of clusters. For example, suppose

$$\begin{array}{rccccccccccccc} & 1 & 2 & \ldots & & k & .. & k' & \ldots & j' & .. & j & \ldots & n \\ \vec{u}_1 = [0 & 0 & \ldots & 0 & 1 & .. & 1 & \ldots & 1 & .. & 1 & \ldots & 1], \\ \vec{u}_2 = [1 & 1 & \ldots & 1 & 0 & .. & 0 & \ldots & 0 & .. & 0 & \ldots & 0]. \end{array}$$

For a well-defined crossover operator, the common gene segments of the mating parents (or the identical partition of object points in the case of clustering)

should remain unchanged after the implementation of crossover. However, it is obvious that this will be violated when we simply set $\vec{u}_1(k)$ to the value of $\vec{u}_2(k)$ or vice versa.

Actually, this is the problem that exists in previous genetic clustering algorithms. In most previous genetic clustering algorithms, e.g. [111] - [114], each member in the population represents the set of cluster centres. After selection, the standard one-point crossover (as in [113, 114]) or two-point crossover (as in [112]) is applied. The *building block hypothesis* in the GA theory assumes that through the implementation of crossover, while other genes are recombined randomly gene segments associated with high fitness (building blocks) are passed down in the population such that longer building blocks may be formed and finally the optimal solution may be found. In conventional GA approaches, the standard one-point or two-point crossover is adopted to achieve this, which simulates the natural crossing-over. In a nature system, a gene occupying a given locus on a chromosome represents a particular physical characteristic, so as in most practical problems for GAs to solve. However, as stated above, the labelling of clusters is arbitrary in clustering problems; two different gene strings may represent a same partition. Obviously, the standard one-point or two-point crossover won't be able to discover this implicit similarity between gene strings, neither does it satisfy the building block hypothesis in the case of clustering problems. To solve this problem, steps 2(b) and 2(c) in PX are necessary, through which the index value of a certain cluster used in a parent is converted into that used in another parent.

c. Mutation. After every crossover, mutation is imposed on the newly constructed partition with a mutation probability p_m. Mutation sets a chosen element of the partition vector to a randomly generated integer ranging from 1 to c.

7.2.4 Creation of a new generation

Our genetic clustering algorithm is a steady-state GA, which replaces only a fraction of the population at each generation. The size of the fraction is controlled by a GA parameter - *generation gap*. The motivation of introducing a steady-state GA is to keep a good balance between exploitation of the best regions found so far and continued exploration for potentially better payoff area.

However, as stated in [115], a steady-state GA will increase the variance along the growth curves of the population members. To reduce this variance, a First-In-First-Out (FIFO) deletion has been suggested in [115]. With FIFO deletion, the population is simply a first-in-first-out queue with new members added to one end and deleted members removed from the other end. To permit the use of the steady-state GA with smaller populations, FIFO has been employed in our GHCMCA.

7.2.5 One-step local search with HCMCA

As mentioned above, the new genetic clustering algorithm combines a genetic approach with the traditional HCMCA. At each GA generation, when the fitness evaluation takes place, a single HCMCA updating procedure is applied to each member in the population to complete a one-step local search. Firstly, according to the partition matrix represented by a population member the corresponding cluster centres are evaluated using equation (5.3.1). Then based on equation (5.3.2) each object is reassigned to the closest cluster centre and the partition represented by the given population member is updated. By adopting HCMCA, we have reduced the searching space of a genetic clustering approach significantly.

It is well known that under HCMCA high computational complexity is involved in determining the closest cluster centre for each object. When the

objects to be clustered are defined by a large number of features then this becomes a serious problem. Under GHCMCA, local search by HCMCA is applied to each population member at each generation. This problem becomes more critical. In order to speed up the local search by HCMCA, the method presented in [116] is adopted. Instead of explicitly calculating the distance between an object and each candidate cluster centre and then picking the centre with the largest distance, this method eliminates some candidate centres without calculating the Euclidean distance, based on the fact that

$$\mathrm{D}^2(\vec{x}, \vec{y}) \geq n\mathrm{D}_2{}^2(\vec{x}, \vec{y}) \geq k(m_{\vec{x}} - m_{\vec{y}})^2 \tag{7.2.2}$$

where $\vec{x}$ and $\vec{y}$ are k-dimensional vectors which can be decomposed into n n-dimensional subvectors, $\mathrm{D}^2(\vec{x}, \vec{y})$ is the squared Euclidean distance between vectors $\vec{x}$ and $\vec{y}$, $m_{\vec{x}}$ and $m_{\vec{y}}$ are the mean value of $\vec{x}$ and $\vec{y}$ ($m_{\vec{x}} = \frac{1}{k}\sum_{j=1}^{k} x_j$), and $\mathrm{D}_2{}^2(\vec{x}, \vec{y})$ is the squared Euclidean distance between the vectors consisting of the mean values of the subvectors of $\vec{x}$ and $\vec{y}$.

For a statistical analysis of the amount of mathematical operations that may be saved using this method, see [116].

7.2.6 Other components

The parameter settings of the new algorithm GHCMCA is summarised in Table 7.1. The genetic approach stops after a certain number of generations. Table 7.2 outlines the GHCMCA.

7.3 Experiments

Experiments have been undertaken using four data sets: single feature data, Iris data, Chernoff faces data and Lenna image. The aim is to evaluate the performance of GHCMCA, especially the ability of the new crossover PX to create good solutions.

Table 7.1: The parameter settings of GHCMCA.

parameter	setting
population size	30
crossover rate	0.80
mutation rate	0.001
generation gap	0.6
η% in PX	20%

The single feature data is an artificial data set [117], in which each object is defined by a single feature. The second data set is the Iris data set. It consists of observations of four features for 150 samples from three species of Iris. These two data sets are useful in this study because they were also used in the original work of GGA [112]. Using them allow the comparison of our results with those reported in [112]. Chernoff faces data set [79] is another real-world situation in which clustering techniques have proven valuable. Chernoff faces are two-dimensional faces whose characteristics are geographically determined by eight feature variables. There are seven clusters composed by altogether 22 faces.

The complexity of a clustering problem increases as the number of objects or the number of object features grows, or conversely, as the number of the groups decreases. As clustering is often applied to vector quantisation and images are real-world domains of significant complexity, the problem of image quantisation is considered. In this application case, a 256×256 black-and-white image is firstly divided evenly into small blocks of 4×4 pixels. Then the gray levels of the pixels in each of these blocks compose a vector such that there are 4096 image vectors of 16 features. The goal of this image quantisation problem is to cluster these 4096 image vectors into 256 classes such that the image may be stored using only 1/8 of the original size without much distortion.

Table 7.2: Program outline of GHCMCA.

Symbols:

μ: population size; n: number of objects; c: number of clusters.

Algorithm:

1. Randomly initialise μ n-dimensional partition vectors. Constrain the initial values of elements in these vectors to be within the range of $[1, c]$.
2. Apply local search to each population member using equations (5.3.1) and (5.3.2).
3. For each population member, calculate distortion and fitness by equations (5.2.4) and (7.2.1), respectively.
4. REPEAT
   (a) Select ($\mu \times$ generation gap)/2 pairs of parents using stochastic universal sampling scheme;
   (b) Apply PX and random mutation to each parent to generate offspring.
   (c) Use FIFO deletion to create a new generation, replacing the ($\mu \times$ generation gap) oldest members with the offspring generated in step (b).
   (d) Apply local search to each population member using equations (5.3.1) and (5.3.2).
   (e) For each population member, calculate distortion and fitness by equation (5.2.4) and (7.2.1), respectively.

   UNTIL maximal generation number.

Table 7.3: Brief summary of the experimental data sets.

data set	number of objects	number of clusters	number of object features
single feature	50	6	1
Iris data	150	3	4
Chernoff faces	22	7	8
Baboon image	4096	256	16
Lenna image	4096	256	16

Two standard black and white images -Baboon and Lenna - are used in the experiments.

Table 7.3 gives a brief summary of the five experimental data sets with respect to number of objects, number of clusters and number of object features.

The proposed GHCMCA as well as the traditional HCMCA and GGA have been applied to each of the experimental data sets, respectively. The purpose of comparing with the traditional HCMCA is to show that while HCMCA may be easily trapped into different local optima, a genetic approach is able to avoid these local optima. And the purpose of comparing with GGA is to show that when applied to clustering problems the proposed PX is superior to standard one-point and two-point crossover in the sense that better solutions can be found more efficiently by GHCMCA. To concentrate on comparison of the effectiveness of different crossover operators, the GGA has been recreated with some differences between ours and the original one. These differences are highlighted as follows:

1. While the original implementation used k-fold tournament selection with $k = 2$, this implementation used a stochastic universal sampling scheme, as used in the GHCMCA.

2. While the original implementation used a conventional GA with an elitist

strategy of passing the two fittest population members to the next generation, this implementation used a steady-state GA with generation gap of 0.6 throughout. Elitism is implemented to keep the fittest member.

3. While the original implementation used a binary gray code representation for the population members, this implementation used real value representation.

Like those used throughout the experiments with the GHCMCA, the same parameter settings have been chosen in all the repeated GGA approaches. Except when better solutions can be obtained with a larger population size, these values offer GGA the best performance [112]. And to speed up the convergence, the one-step local search is also introduced into GGA.

As for the speed-up method mentioned in section 7.2.5, it has been applied in the image quantisation experiments. And in the genetic approaches (including both GGA and GHCMCA) for image quantisation, the population size was increased to 50.

7.4 Results

The *mean squared error* (MSE) rather than the raw value from the clustering objective function $J_1(\mathbf{U}, \mathbf{V})$ has been used as the performance measure for the image quantisation problem since it was also used in [112] for image quantisation. MSE is the value of the J_1 averaged by the number of image pixels. For each data set except the images Baboon and Lenna, we report the raw values of J_1 and the results were obtained over 500 independent trials using the traditional HCMCA and 100 independent trials using each genetic approach, respectively. For images Baboon and Lenna, results were obtained over 100 HCMCA trials, 40 GGA trials and 40 GHCMCA trials, respectively. In each independent trial with each clustering algorithm, the partition matrix

with the lowest cost value (J_1 or MSE) was traced and the cost values of these partition matrices were recorded.

The recorded J_1 values for the single feature and Iris data sets are listed in Table 7.4, along with the times they were found by each algorithm. Same J_1 values were reported in [112] and thus confirm the validity of our simulation programs. According to the times that an algorithm finds the optimal solution, we can estimate the probability for that algorithm to find the optimal solution with a certain confidence level. Table 7.5 gives the confidence interval for this probability for each testing algorithm, respectively, with a confidence level of 0.95. As shown in Table 7.4, most trials of HCMCA ended up with a J_1 value higher than the optimal values (0.9348 for single feature data set and 78.941 for Iris data set). And as shown in Table 7.5, the probability for HCMCA to find the optimal solution is slight while GGA and especially GHCMCA can find the optimal solution with a higher probability. These indicate that the traditional HCMCA did stop at different local extrema, while, on the other hand, illustrate the ability of a genetic approach to avoid them.

From the recorded cost values, the average cost, the standard deviation and the lowest cost have been evaluated for each experimental data set and are listed in Table 7.6.

For clustering problems with high-dimensional data space, the distribution of local optima is expected to be broad. Figures 7.3(I), 7.3(II) and 7.3(III) show the distribution of the cost values obtained by each algorithm when applied to Chernoff faces, image Baboon and image Lenna, respectively.

Figures 7.4(I) and 7.4(II) show the best compressed images obtained using each algorithm for the images Baboon and Lenna, respectively.

From Table 7.6 and Figures 7.3 and 7.4, the following observations are drawn:

1. As indicated by the low standard deviation values in Table 7.6 and as

Table 7.4: Frequency of different partitions found by HCMCA, GGA and GHCMCA when applying to single feature and Iris data, respectively.

Single Feature				Iris Data			
J_1	times			J_1	times		
	HCMCA (500 runs)	GGA (100 runs)	GHCMCA (100 runs)		HCMCA (500 runs)	GGA (100 runs)	GHCMCA (100 runs)
0.9348	8	72	90	78.941	18	40	100
0.9469		9	2	78.944		1	
0.9554		4	2	78.945	360	59	
1.1890	238	15	6	142.852	2		
1.3016	72			142.859	8		
1.3297	170			142.879	3		
1.3473	9			143.454	9		
1.6118	3						

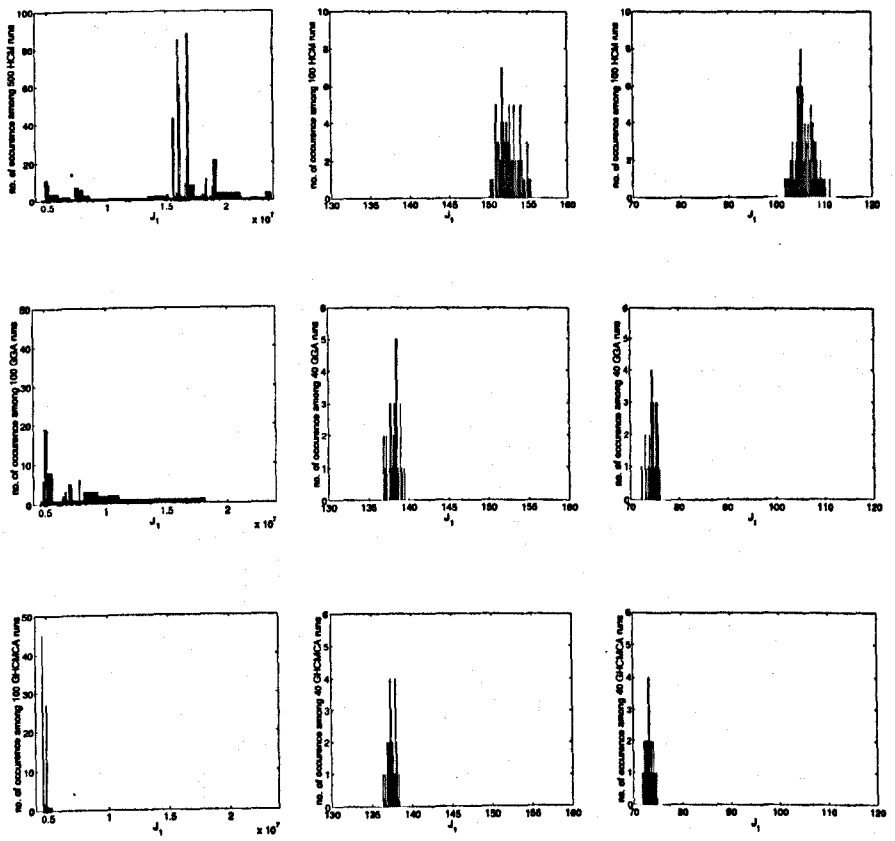
Table 7.5: Probability for finding the optimal partition when applying HCMCA, GGA and GHCMCA to single feature and Iris data, respectively. The confidence intervals (lower and upper bounds) for these probabilities are calculated with a confidence level of 0.95 here.

Single Feature			Iris Data		
	confidence interval			confidence interval	
algorithm	lower bound	upper bound	algorithm	lower bound	upper bound
HCMCA	0.838	0.938	HCMCA	0.971	1.000
GGA	0.640	0.786	GGA	0.324	0.483
GHCMCA	$3.52e^{-3}$	0.046	GHCMCA	0.014	0.075

Table 7.6: Comparison of results obtained by different algorithms.

data set	algorithm	average J_1	standard deviation	lowest J_1 found
single feature	HCMCA	1.2554	0.0828	0.9348
	GGA	0.9748	0.0906	0.9348
	GHCMCA	0.9507	0.0606	0.9348
Iris data	HCMCA	81.7680	13.1721	78.9408
	GGA	78.9434	0.0021	78.9408
	GHCMCA	78.9408	0	78.9408
Chernoff faces ($\times 10^6$)	HCMCA	15.6130	3.3369	4.6331
	GGA	6.2923	1.9313	4.6262
	GHCMCA	4.7290	1.4906	4.6262

data set	algorithm	average MSE	standard deviation	lowest MSE found
Lenna	HCMCA	105.9800	1.9582	101.7270
	GGA	74.6203	0.9591	72.1945
	GHCMCA	73.2502	0.7916	71.7535
Baboon	HCMCA	152.4841	1.1871	150.1970
	GGA	138.1310	0.6502	136.8030
	GHCMCA	137.4604	0.4625	136.3390



(a) HCMCA

(b) GGA

(c) GHCMCA

(I) Chernoff faces (II) Baboon (III) Lenna

Figure 7.3: Distribution of the J_1 values associated with the partitions found when applying HCMCA, GGA and GHCMCA to the Chernoff faces, Lenna image and Baboon image, respectively.

(a) original

(b) HCMCA

(c) GGA

(d) GHCMCA

Figure 7.4: Comparison of the visual quality of Baboon and Lenna images after quantisation: (a) original image, (b) using HCMCA, (c) using GGA, (d) using GHCMCA.

shown in Figures 7.3(b) and 7.3(c), partitions with similar low cost values were repeatedly found by GGA and GHCMCA. This indicates that these results are indeed nearly optimal and not simply locally optimal.

2. In Table 7.6, the standard deviations obtained by HCMCA are always higher than those by the genetic approaches. In Figure 7.3(a), broad distribution of results is observed with the traditional HCMCA. All of these expose the dependence of HCMCA on the initial conditions and meanwhile shows the viability of a genetic approach to alleviate the difficulty of choosing an initialisation for an HCMCA.

3. As shown in Table 7.6, for the two image quantisation problems, the lowest MSE values obtained by HCMCA are much higher than the corresponding value obtained by GGA or GHCMCA. Furthermore, as shown in Figure 7.4, the visual quality of the compressed images obtained by HCMCA is much worse than that by GGA or GHCMCA. This indicates that the results obtained by HCMCA in these two application cases are far from the global optimal point.

4. On average, the final cost values obtained by GHCMCA are slightly lower than that obtained by GGA. In no case did GGA result in a cost value lower than the lowest value found by GHCMCA. Moreover, the smaller standard deviation of GHCMCA indicates that it may find the global optimal or near-optimal solutions with a higher probability.

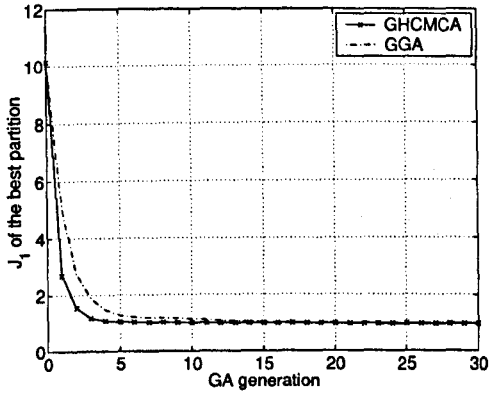
For stochastic approaches like the GA approaches, another important performance measure is the convergence rate. This becomes more important for the new algorithm due to the heavy computation load brought by the clustering problem. From the results obtained with each experimental data set, the convergence properties of GGA and GHCMCA in terms of the generation number are shown in Figures 7.5(a)-(e). For the first three data sets, the J_1

values are averaged over 100 independent trials, respectively. And for the two images, the MSE values are averaged over 40 independent trials, respectively.

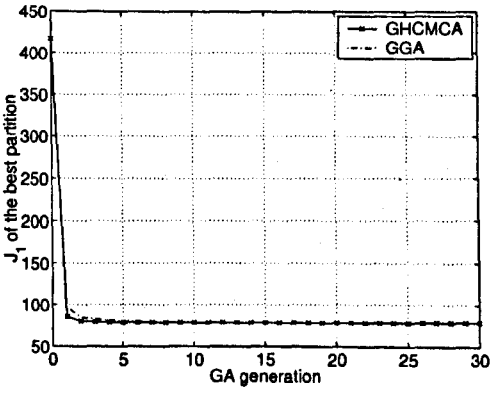
From Figure 7.5(a)-(e), we observe that the initial convergence rate is very high and as the generation progresses the convergence rate decreases rapidly. After a few generations, the genetic clustering approaches can find a cost value lower than the corresponding average value obtained by HCMCA. As the complexity of the clustering problem increases, such as illustrated in Figures 7.5(d) and 7.5(e) for the image quantisation problems, GHCMCA shows its superiority over the GGA in the sense that during the early generations GHCMCA converges much faster than GGA and quickly reaches the desired region where the global optimal partition resides. This suggests to us that in special cases where speed as well as performance is required, GHCMCA may provide a much faster way to find a convincing solution. Furthermore, as HCMCA assures local optimality and, due to its hill-climbing nature, converges much faster than any genetic approach, instead of waiting for the genetic approaches to reach an exact optimal solution, we may stop the genetic search after a necessary number of generations and then use HCMCA to find a local optimum nearby.

7.5 Conclusions

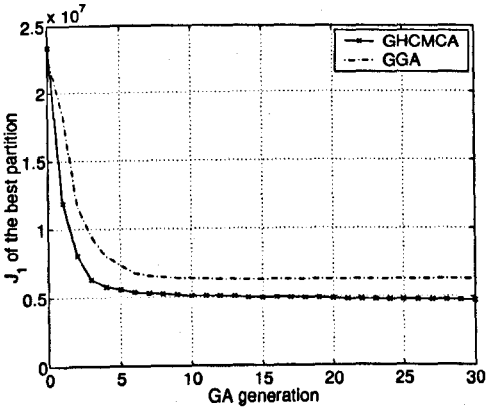
This chapter presents a novel genetic clustering algorithm - GHCMCA. GHCMCA exploits both the efficiency of the calculus-based HCMCA and the stochastic nature of a GA. A new crossover operator has been adopted in GHCMCA, which was specially designed for clustering problems. The ability of a genetic approach to avoid local optima and reduce the sensitivity of initialisation are demonstrated in the comparative experiments. Compared to the previous GGA, GHCMCA may achieve better ultimate convergence with a higher probability. Moreover, for complex clustering problems (such as image quantisation), it converges much more quickly to the global optimum and



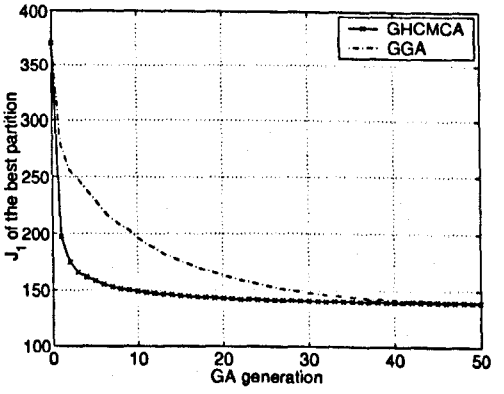
(a) Single feature data set



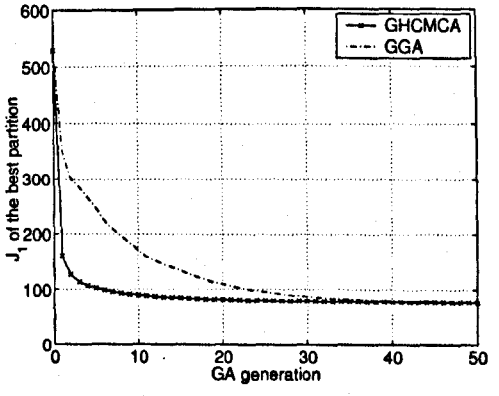
(b) Iris data set



(c) Chernoff faces data set



(d) Baboon image



(e) Lenna image

Figure 7.5: Comparison of convergence properties of genetic approaches - GGA and GHCMCA.

therefore provides a viable way out of the dilemma where the classical HCMCA may be easily trapped by local extrema and a conventional genetic approach is time consuming.

The next chapter presents another successful application of GAs, where job-shop problems is the problems to be solved.

Chapter 8

APPLICATION OF GENETIC ALGORITHMS TO JOB-SHOP PROBLEMS

8.1 Introduction

The class of job-shop problems is the main area of machine scheduling. The objective of a job-shop scheduling problem (JSP) is to find a sequence called *schedule*, in which the jobs pass between the machines such that a given performance measure is optimized. The class of JSPs arises because of its strong basis in reality. A diversity of optimization problems in fields such as production operations in manufacturing industry, parallel and distributed systems, logistics and traffic controls can be summarized within the general class of JSPs. It is well-known that the class of JSPs is among the hardest combinatorial optimization problems. They are NP-complete [119] and each job in a given JSP demands a specific machine order.

Apart from heuristic methods, *branch-and-bound* [120, 121] is probably the solution technique most widely used in job-shop scheduling. It is an implicit

enumeration method, which checks every possible schedule but does not consider every possibility explicitly. It eliminates many possible paths on route. However, the number of operations required, and hence the time required, to solve a JSP by branch-and-bound is unpredictable and may be too great for such an exhaustive search to be humanly possible.

As the searching technique of genetic algorithms (GAs) became popular in the mid 1980s, many researchers started to apply this heuristic searching scheme to machine scheduling problems [122, 123, 124]. In the application of a GA, only is the fitness value of each chromosome/individual in the population required. Therefore, GAs are suitable for such optimization problems as JSP, where there is no derivative information available. However, due to the ineffective schemes used for the solution encoding and GA operations, most of them converge slowly and their implementations are fairly complicated. The best convergence results of the existing genetic scheduling algorithms [122, 123] are far from satisfactory while the algorithm in [124] requires a large number of generations (more than 5000 generations) to reach a high quality solution. The essential difficulty of the application of GA to JSP is due to the fact that the classic implementation schemes of the genetic operators - recombination and mutation - are not suitable for ordering problems like JSP. They may nearly always result in an infeasible schedule with some operations missing while others represented twice, and the specified machine orders on the jobs violated. Intuitively (also as in many present genetic scheduling algorithms), extra adjustments - such as the Giffler and Thompson's method [125] (p. 158) used in [122, 124] - may be applied immediately after each recombination and mutation phases to ensure the feasibility of the schedules in the current GA population. However, the feasibility checking over all schedules at each GA generation will definitely cost a lot of computation time. And it should be expected that some good gene permutations (i.e. the building blocks) will be disrupted by this artificial interference.

To tackle these problems, a new genetic scheduling algorithm (GSA) has been proposed. Our aim is threefold: simplicity of structure, ease of implementation and high quality of ultimate convergence.

8.2 Problem Formulation

An $n \times m$ JSP involves a set of n *jobs* $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ waiting to be processed through a set of m *machines* $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$. Each job must pass through each machine exactly once. The processing of a job on a machine is called an *operation*. The operation of the ith job by the jth machine is denoted by o_{ij}. There are $n \cdot m$ operations need to complete, which compose the set of operations $\mathcal{O}$. Each operation, $o_{ij} \in \mathcal{O}$, takes a certain length of time, the *processing time* t_{ij}, to perform. The added constraint demands that each job must pass through the set of machines in a particular order - the specified *machine order*. For simplicity, in our study of JSP, the demanded machine orders of an $n \times m$ problem are represented by an $n \times m$ matrix $\mathbf{M}$, where the ith row specifies the machine order of job J_i; the processing times of the $n \cdot m$ operations are represented by an $m \times n$ matrix $\mathbf{T}$, where $\mathbf{T}(j, i) = t_{ij}$ for any pair of $j = 1, 2, \ldots, m$ and $i = 1, 2, \ldots, n$.

The general objective of a JSP is to find a sequence called *schedule* that is

(a). compatible with the given machine orders, i.e. a *feasible* schedule, and

(b). optimal with respect to some criterion of scheduling performance.

Makespan is the primary criterion of performance in the literature of JSP. *Makespan*, $C_{\max}$, is the total elapsed time of a schedule. In our work on JSP, different schedules are evaluated by their makespans. More formally, the objective of the problem is to find a start time s_{ij} for each $o_{ij} \in \mathcal{O}$ such that:

$$C_{\max} = \max_{o_{ij} \in \mathcal{O}} (s_{ij} + t_{ij}) \tag{8.2.1}$$

is minimized subject to

1. $t_{ij} \geq 0, \quad \forall o_{ij} \in \mathcal{O}$;

2. $s_{i,m_{j+1}} - s_{i,m_j} \geq t_{i,m_j}$, if $m_j = \mathrm{M}(i, j)$ and $m_{j+1} = \mathrm{M}(i, j+1)$, $i = 1, \ldots, n, \ j = 1, \ldots, m-1$;

3. $(s_{i_2,m_j} - s_{i_1,m_j} \geq t_{i_1,m_j}) \bigvee (s_{i_1,m_j} - s_{i_2,m_j} \geq t_{i_2,m_j})$, if $m_j = \mathrm{M}(i_1, j_1) = \mathrm{M}(i_2, j_2), \ i_1, i_2 = 1, \ldots, n, \ j_1, j_2 = 1, \ldots, m$.

8.3 The New Genetic Scheduling Algorithm

Under GA, each solution to the given problem is encoded and represented by a chromosome. Very often the key of GA success with practical problems lies in the development of a suitable combination of solution encoding and genetic operators. Unfortunately, the classic GA crossover and mutation can easily destroy the feasibility of the solutions of a constrained ordering problem like JSP. There are two ways to solve this problem:

- by modifying the GA operators such that they always produce chromosomes to which feasible solutions correspond; or
- by defining a new encoding/decoding scheme for solution representation such that all possible chromosomes are decoded to feasible solutions.

We have chosen the first alternative and developed a new genetic scheduling algorithm - GSA.

8.3.1 Solution encoding

In machine scheduling, the process of deriving a schedule from an operation sequence is called timetabling. A complete schedule involves not only the sequencing/ordering information but also the timetabling results, i.e. the start

Table 8.1: The numbers of feasible operation sequences for three JSPs

problem size	number of feasible operation sequences
6×6	2.6702×10^{24}
10×10	2.3571×10^{92}
20×5	2.4343×10^{116}

and finish times of each operation. However, considering the complexity of the description of a schedule, we prefer a simpler chromosome form in our GSA and thus encode only the sequencing information into a chromosome. The sequencing information of a schedule is visualized by its operation sequence. Clearly, an operation sequence is simply a permutation of all the operations involved. Thereby it is natural to encode the problem in the permutation form. In JSP, operations are identified by two factors: to which job it belongs and on which machine it should be processed. Considering this, under GSA, each operation o_{ij} $(i = 1, \ldots, n,\ j = 1, \ldots, m)$ of an $n \times m$ JSP is assigned with a distinct index number equal to $(i-1)m + j$. Consequently, a chromosome becomes a permutation of these index numbers, i.e. a permutation of the set of integers ranging from 1 to $m \cdot n$. There are $(m \cdot n)!$ such permutations. Notice that, for an $n \times m$ JSP, there are $(m!)^n$ possible sets of machine orders; while once a specific JSP is defined the problem demands a particular set of machine orders and all feasible solutions to the problem must be compatible with it. Thus the total number of feasible operation sequences for an $n \times m$ JSP is

$$\frac{(m \cdot n)!}{(m!)^n}. \tag{8.3.1}$$

Based on equation (8.3.1), the total numbers of feasible operation sequences of three JSPs are listed in Table 8.1.

On one hand, the astronomically large numbers in Table 8.1 show the difficulty for any algorithm to locate the optimal solution. On the other hand, although the numbers in Table 8.1 are astronomically large, they are finite.

In other words, we are minimizing a function over a finite set and thus the existence of an optimal solution under GSA is ensured.

8.3.2 Initialization

In the construction of operation sequences, two things need to be taken in consideration: the avoidance of deadlock and the compliance with the demanded machine orders. In order to meet them, a procedure, which is a variation of the decoding scheme proposed in [126], has been used to construct chromosomes in the initial population. This procedure is carried out as follows. Firstly, compose a ready list with only those operations that do not have any predecessors. Then, randomly pick up an operation from the ready list and update the ready list. Repeat the picking and updating until no operation remains in the ready list. Notice that only operations having no predecessors would appear in the ready list and random picking is always from the ready list. Consequently, deadlocks are avoided and the machine order for each job is naturally kept. In other words, all operation sequences constructed by this procedure are feasible.

8.3.3 Cost and fitness evaluation

As stated, makespan is the cost measure in our study. We can work out the makespan of a feasible operation sequence by plotting a simple diagram called Gantt chart, such as the one in Figure 8.1. Table 8.2 lists the pseudo code for generating a Gantt chart.

By examining the pseudo code in Table 8.2, it can be seen that the earliest starting time of an operation on a schedule can be delayed only by the operations on the same machine and the preceding operations belonging to the same job. Hence, different operation sequences may deduce a same schedule. To reduce the searching space of GSA, after each makespan evaluation, an

Table 8.2: Algorithm of drawing Gantt chart of a given operation sequence

Symbols:

$S(l)$: the lth operation of the given operation sequence S,

$M_{\text{time}}(j)$: the ready time of the jth machine,

$J_{\text{time}}(i)$: the ready time of the ith job.

Algorithm:

set $M_{\text{time}}(j)$ to 0, $j = 1, \ldots, m$;

set $J_{\text{time}}(i)$ to the release time of job J_i, $i = 1, \ldots, n$;

set l to 1;

REPEAT

get the job index j and the machine index i of operation $S(l)$ using equation (8.3.1);

compute the earliest start time: $start = \max(J_{\text{time}}(i), M_{\text{time}}(j))$ and finish time: $finish = start + t_{i,j}$;

set $J_{\text{time}}(i)$ and $M_{\text{time}}(j)$ to the earliest finish time $finish$;

draw job J_i on the Gantt char following machine M_j with a block from time instant $start$ to $finish$;

increment l;

UNTIL $(l > m \cdot n)$.

operation sequence will be replaced by the one corresponding to the derived schedule, omitting the timetabling information. The operation sequence of a schedule is obtained by sorting the start times of all the operations in ascending order.

It has been proved that an optimal schedule providing minimal makespan must be active and at least semi-active [125]. A semi-active schedule ensures the processing of each operation starts as soon as it can be, while obeying the demanded machine orders. In an active schedule, the operation sequence is such that no operation can be started any earlier without either delaying some other operation or violating the machine orders. Obviously, schedules deduced by the algorithm in Table 8.2 will be semi-active however not necessarily active. The reason is that by that algorithm, all operations are scheduled in the fixed order determined by the gradually extended operation sequence such that blank interval with long length may be left in the schedule. When the length of any blank interval becomes longer than the processing time of an operation processed later on the same machine, the schedule becomes only semi-active and therefore may have a shorter makespan. Considering the optimal schedule is in the set of active schedules, we would like the operation sequence represented by each chromosome to be active. This can be achieved if we check the blank intervals long enough to complete the next operation within it before appending that operation at the end of the schedule and simply fill the first possible blank interval, if there is any.

Since the objective of JSP is to minimize the makespan while the GA favors fitter individuals, the fitness of the ith individual in the population is defined as

$$fitness(S_i) = \frac{\max_{S_j} C_{\max}(S_j)}{C_{\max}(S_i)}, \tag{8.3.2}$$

where $C_{\max}(S_i)$ is the makespan of the operation sequence represented by the ith chromosome in the GA population.

To prevent premature convergence and maintain reasonable selection pressure, a typical linear fitness scaling mechanism [81] has been adopted.

8.3.4 Genetic operations

Selection

In GSA, the *stochastic universal sampling* scheme [104] has been used for proportional selection.

Crossover

As mentioned, for an ordering problem like JSP, crossover is the most difficult GA operator to design. Recall that, with the encoding scheme of GSA, each operation has a unique index number. For a complete operation sequence, each of these index numbers should present once only. This may lead to the problem that any segment of a chromosome cannot be directly replaced by any segment from another chromosome. Otherwise, according to the resulting sequence, some operations may be processed more than once while some others may be missing, let alone the compliance of the demanded machine orders. In general, this problem is the primary argument against permitting simple crosses of traditional GA between ordered strings. To solve this problem, three similar crossover operators - partially matched crossover (PMX), order crossover (OX), and cycle crossover (CX) - have been developed independently (see [81] for a brief introduction of these crossover schemes). All arose in considering ways to tackle a blind travelling salesman problem (TSP) and succeeded in maintaining the completeness of solutions in a GA search for TSPs. However, in a TSP, only one machine (the salesman) is concerned and thereby, unlike in JSP, no so-called machine order is imposed. This suggests that we must modify these crossover operators before being able to adopt them in a GA for JSP. In a JSP, the order of the operations of a same job is constrained while

the operations of different jobs can be arranged freely. In this light, a new crossover operator has been devised in our previous work [127], which adopts the basic mechanism behind OX but works on the job basis rather than the gene (operation) basis as the past crossover operators do. We named the new crossover operator job-based order crossover (JOX).

Instead of choosing splitting site(s) along a chromosome, JOX divides the n job index numbers, $\{1, 2, \ldots, n\}$, into two exclusive sets, the desired set $\mathbf{S}$ and the undesired set $\bar{\mathbf{S}}$, with each set containing at least two elements. Then in a manner similar to OX, JOX removes, from a parent's operation sequence, the operations belonging to the jobs in $\bar{\mathbf{S}}$ and refill these vacant position with the same operations in the exact order they appear along the other parent's operation sequence. For example, we have the following machine order matrix:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 4 & 2 \\ 1 & 3 & 2 & 4 \\ 3 & 2 & 4 & 1 \end{bmatrix}$$

and the following pair of chromosomes have been chosen to crossover (for convenience, we represent an operation in the form of $(\text{job inedx})_{(\text{machine index})}$):

$$parent_1 : 2_3\, 2_1\, 3_1\, 4_3\, 3_3\, 1_1\, 4_2\, 3_2\, 1_2\, 1_3\, 2_4\, 2_2\, 1_4\, 4_4\, 3_4\, 4_1$$

$$parent_2 : 1_1\, 1_2\, 4_3\, 3_1\, 1_3\, 3_3\, 2_3\, 4_2\, 4_4\, 2_1\, 1_4\, 2_4\, 2_2\, 3_2\, 4_1\, 3_4$$

Suppose $\mathbf{S} = \{J_1,\ J_3\}$ and $\bar{\mathbf{S}} = \{J_2,\ J_4\}$. When $parent_1$ maps to $parent_2$, the operations belonging to J_2 and J_4 will be removed and leave holes (marked by an H) in the sequence:

$$parent_1' : H\, H\, 3_1\, H\, 3_3\, 1_1\, H\, 3_2\, 1_2\, 1_3\, H\, H\, 1_4\, H\, 3_4\, H$$

These holes are then filled with the same operations taken from the $parent_2$, keeping the appearing order unchanged. Performing this step and completing the complementary cross we obtain the offspring as follows:

$$offspr_1 : 4_3\,2_3\,3_1\,4_2\,3_3\,1_1\,4_4\,3_2\,1_2\,1_3\,2_1\,2_4\,1_4\,2_2\,3_4\,4_1$$

$$offspr_2 : 1_1\,1_2\,2_3\,3_1\,1_3\,3_3\,2_1\,4_3\,4_2\,2_4\,1_4\,2_2\,4_4\,3_2\,4_1\,3_4$$

Syswerda [128] conjectures that the order as well as the position of genes in the permutation of a JSP are meaningful. To be more precisely, we expect the absolute position to be of particular interest because it directly express precedence relations among operations in a schedule. Under JOX, the absolute positions of the operations of the desired jobs are inherited invariantly from a parent while the relative positions of the operations of the undesired jobs are inherited invariantly from the other. Moreover, as a consequence, no machine order will be violated by JOX and thus there is no need for time consuming feasibility checking and correction afterwards.

Mutation

In GSA, a new mutation operator has been devised specially for JSP. The new mutation operator randomly extracts an operation from the given operation sequence and then re-insert it to a randomly selected position between its preceding and following operations of the same job.

8.3.5 Creation of a new generation

Our genetic clustering algorithm is a steady-state GA. Like in GIICMCA, the First-In-First-Out (FIFO) deletion has been adopted here.

Moreover, a newly generated sequence is inserted into the population only if its makespan is better than that of the one it is generated from.

8.3.6 Hybrid with a local search

Local search has been known as a useful tool for solving combinatorial problems. The basic strategy in a local search is simple and straightforward:

Table 8.3: The Parameter Settings of GSA

parameter	settings
population size	200
crossover rate	0.80
mutation rate	0.001
generation gap	0.5

replace a solution with the best solution in its neighborhood. It has been observed that local optima of JSP tend to be relatively close to each other and the known global optimum. This fact suggests that local search should be effective in finding near-optimal solutions for JSP. In this light, local search has been inserted in GSA. To perform a local search, the neighborhood of a solution needs to be defined beforehand. We inherit the definition used in the simulated annealing algorithm [129] and tabu search [130] for JSP and define the neighborhood of a solution to JSP (a schedule) as the set of solutions which can be obtained by reversing a pair of adjacent operations on its critical path. In JSP, the *critical path* of a schedule is composed of the operations processed on the same machine as the last finished operation on that schedule. For an $n \times m$ JSP, there are $n - 1$ pairs of adjacent operations on the critical path of a schedule. Due to the imposition of the required machine orders, dead lock and thus an infeasible schedule may be caused by simply reversing a pair. Therefore, the size of the neighborhood of a schedule may be less than $n - 1$.

8.3.7 Other components

The parameter settings of our new algorithm GSA is summarized in Table 8.3. The genetic approach stops after a certain number of generations.

8.4 Experiments and Results

The proposed genetic schedule algorithm (GSA) has been implemented in C^{++}. Experiments have been conducted on three well-known job-shop benchmark problems [131]:

- MT6×6 (6 jobs, 6 machines)
- MT10×10 (10 jobs, 10 machines)
- MT20×5 (20 jobs, 5 machines)

These three problems have been used as benchmarks to test many other existing algorithms for JSP. The last two problems have been found difficult. By GSA, the optimal schedules for all these three problems have been obtained. The minimal makespan values are the 55, 930 and 1165, respectively. The optimal schedules are shown in Figures 8.1, 8.2 and 8.3, respectively. For each problem, more than one optimal schedule with the same minimum makespan have been found. Table 8.4 compares the minimum makespan of these three benchmark problems found by the GSA with that by some other existing algorithms for JSP. The results of GSA are the same as Carlier's branch-and-bound and better than other algorithms. But it is well known that the large amount of computation required by branch-and-bound for JSP is a notorious problem.

Table 8.4: Comparison of the minimum makespans of the three benchmark problems found by different algorithms

Algorithm	MT6×6	MT10×10	MT20×5
Barker 1985 [120] (branch-and-bound)	55	960	1303
Carlier 1989 [121] (branch-and-bound)	55	930	1165
Nakano 1991 [122] (GA)	55	965	1215
Croce 1995 [123] (GA)	55	946	1178
This paper (GSA)	55	930	1165

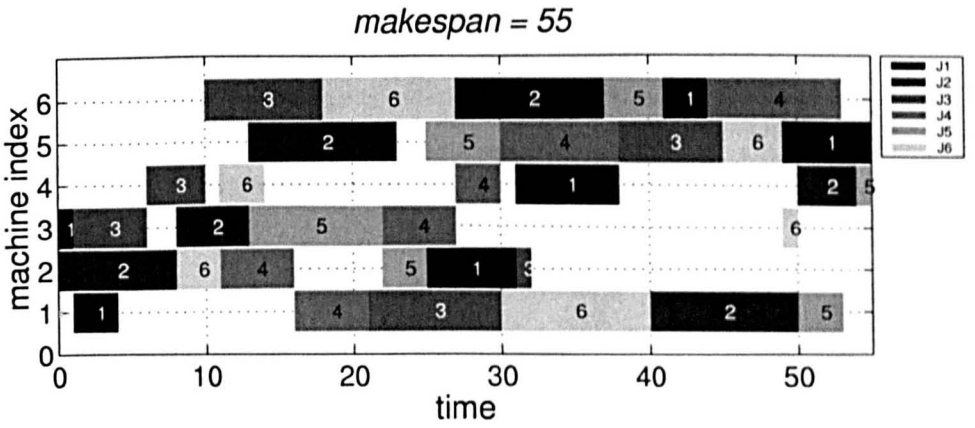


Figure 8.1: Gantt diagram for an optimal schedule of the benchmark problem MT6×6

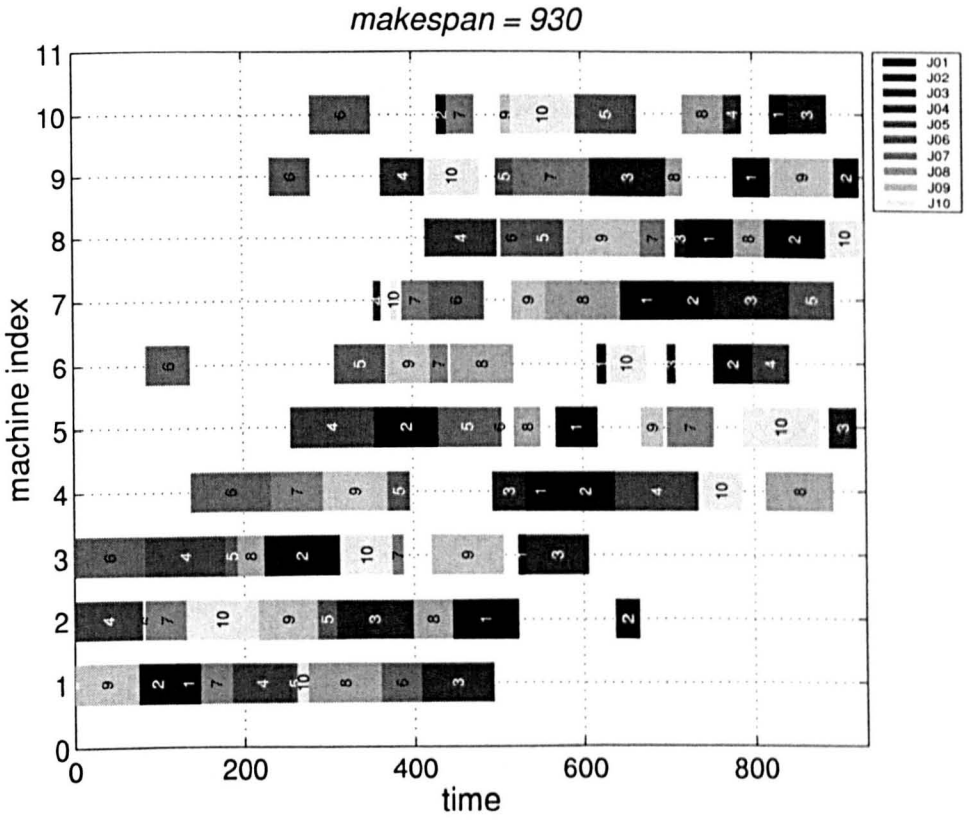


Figure 8.2: Gantt diagram for an optimal schedule of the benchmark problem MT10×10

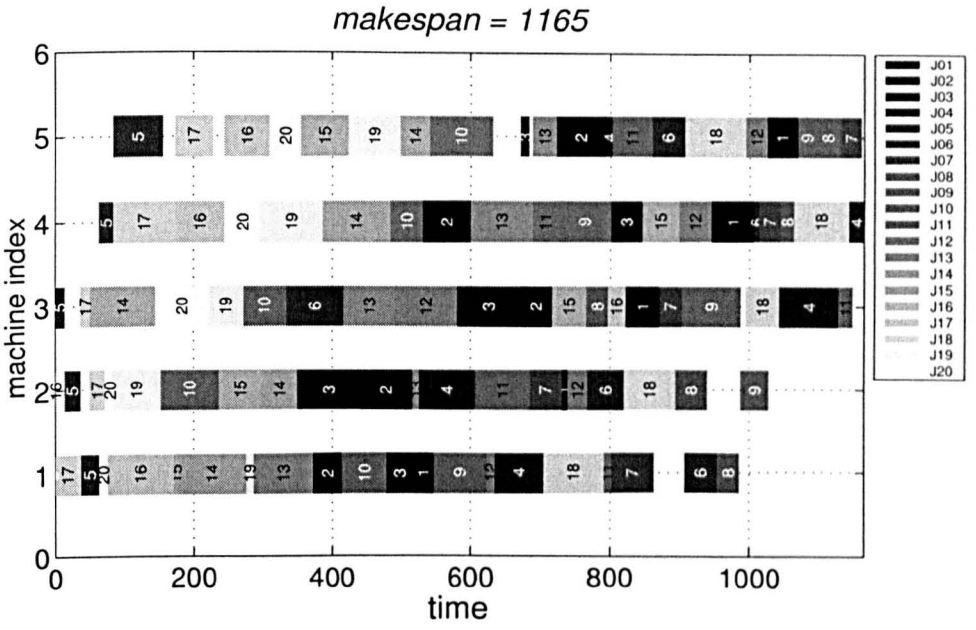


Figure 8.3: Gantt diagram for an optimal schedule of the benchmark problem MT20×5

8.5 Conclusion

In this paper, a new algorithm - GSA has been presented, which is developed specially for job-shop scheduling problems. A new coding scheme is adopted in the GSA, by which the solution space of a general job-shop problem is transformed into a domain suitable for GA search. A simple yet highly effective GA crossover operator has been devised. It has successfully accomplished the role of a GA crossover of combining good gene segments. However, the most significant achievement of the new crossover operator is that no infeasible schedules will be generated by it. Therefore, no complicated and computationally expensive treatment is required. The activeness of the schedules represented by the chromosomes in the population is ensured by the coding scheme and the replacement facility introduced in fitness evaluation. The optimal schedules for all the test problems have been found by the proposed GSA.

Moreover, the techniques developed in GSA for JSP can be adapted to

other closely related problems, such as bin packing, TSP, the scheduling of communication networks or project planning, in a straightforward way.

Chapter 9

CONCLUSION

The summary of the results obtained in this thesis is given below, and by this means its contributions are highlighted. Suggestions for future research are listed at the end.

9.1 Summary of Results

This thesis has been devoted to a discussion on supervised pattern classification using SVMs and unsupervised pattern classification using a GA guided approach, respectively.

In the first part of the thesis, the effect of parameter C on a soft-margin SVM has been investigated and the following conclusions have been drawn:

- For nonseparable data sets, an infinite C would lead to an infinite solution where the margin is infinite small (i.e. margin $\rightarrow$ 0). Otherwise, as the value of C increases, the width of the margin decrease.
- As the value of C increase, more complex decision boundary is required for separating the patterns.
- Small and large values of C may both lead to bad generalisation performance.

For fast training of SVMs, two centre-based algorithms - CO and ECO - have been proposed and the following results have been obtained from the experiments:

- A centre-based algorithm may largely reduce the training time of a SVM.
- Besides the optimality guaranteed by the ECO, its training time scales almost linearly in the training set size. Therefore, ECO may be applied to much larger training sets, in comparison with the standard QP techniques.

In the second part of the thesis, a hybrid algorithm - GHCMCA- has been proposed for clustering, which combines the efficiency of the hill-climbing c-means and the global search of genetic algorithms. Experimental results have shown that:

- A genetic approach is able to overcome the inevitable drawbacks of a hill-climbing technique.
- For complex clustering problems such as image quantisation problems, GHCMCA is superior to the previous genetic clustering algorithm GGA in the sense that it converges much more quickly to the desired region in which the global optimum resides.

Therefore, in the cases where speed as well as performance is required, GHCMCA may provide a solution to the dilemma where the classical clustering algorithm can be easily trapped in different local extrema and the conventional genetic approach is time consuming.

Finally, the application of GAs to NP-complete problems has been extended to JSPs, where a simple yet effective GA crossover operator has been devised.

9.2 Suggestions for Future Work

Here, we address several related points that deserve further investigation.

- Best choice of the kernel function and optimisation of trade-off parameter C

  A significant challenge in SVM application lies in the best choice of the kernel function. If an inappropriate kernel has been used, the generalisation performance will suffer from overfitting (e.g. Figures 3.3 and 3.9). Although some work has been done on limiting kernels using prior knowledge [132, 133], the best choice of the kernel function for a given problem still remains as a research issue. Once the kernel is fixed, SVM classifiers have only one user-chosen parameter - the trade-off parameter C. The effect of C on a soft-margin SVM has been investigated in this thesis, but a general way for determining the optimal value of C for a given problem is still an unsolved problem.

- Multi-class SVM classification

  SVMs were originally studied for two-class classification. However, the real-world problems usually involve more than two classes. Currently, multi-class classification is typically solved by combining several two-class SVM classifiers. Some work has also been done on training a multi-class SVM in one step [134, 135]. As it is computationally more expensive to solve multi-class problems, comparisons of these methods using large-scale problems have not been conducted, especially for methods training a multi-class SVM in one step where a much larger QP problem is involved. Although fast training algorithms can be used to solve this computational problem, the optimal design for multi-class SVM classifiers is still a further area for research, where the competition and interaction between classes should be considered.

- Fuzzy classification using SVMs

  SVMs were originally studied for binary classification. Each example in the training set is associated with a binary value indicating which one of the two classes that example belongs to. Also, the trained SVM will provide a binary value for each testing example. In other words, it is assumed that there is no overlap between classes. However, this is not true for some real-world situations. When overlap exits between classes, the fuzzy relationship of an example to a class is typically represented by a floating point value ranging from 0 to 1. The use of SVMs for fuzzy classification will be of great meaning.

- A genetic approach for fuzzy clustering

  Like in hard clustering, the objective function for fuzzy clustering problems is highly nonlinear and the solution space to be searched is vast. Moreover, there is no deterministic algorithm which can find the optimal partition for all instances of the problem. Under such circumstance, a genetic guided approach should be helpful.

- Automatic detection of cluster number

  A well-known problem in the application of clustering algorithms is the estimation of the number of clusters in the input data set, i.e. the so-called *cluster validity* problem. Most clustering algorithms require the number of clusters to be preselected. And the objective function optimised by the classic clustering algorithms do not involve the number of clusters in their formulas. Whereas the search by a classic clustering algorithm is typically directed according to the gradient of the objective function to be optimised. Shortly, the classical clustering algorithms do not consider the problem of cluster validity. A genetic algorithm require no gradient information about the objective function but only a way to evaluate the fitness of all valid solutions. Therefore, it is possible

to include the number of clusters into the fitness function and thus allow the number of clusters to be determined automatically during a GA search. Hereby, the problem becomes the defining of an appropriate fitness function. It is difficult since the amount of distortion (i.e. the value of the general clustering objective function) decreases monotonically as the cluster number increases. Moreover, an inappropriate cluster number can heavily deteriorate the performance of a clustering approach and lead to serious clustering errors (e.g. when a cluster number larger than the desired one is used, a false cluster may appear in between two clusters causing a mixture of patterns from those two clusters and consequently cause false classification decisions).

As a final conclusion, this thesis has undertaken the investigation of supervised pattern classification using SVMs and unsupervised pattern classification using GA guided clustering. It would be helpful for the further investigation on pattern classification and the future development of methodologies for it.

Appendix A

Notation

Symbols and Operations

$\Leftrightarrow$	is equivalent to
$\rightarrow$	approaches to
$\forall$	for all
$\in$	belongs to
$\mathcal{R}^d$	the d-dimensional real space
$\lim_{x \rightarrow a} f(x)$	the value of $f(x)$ in the limit as x approaches a
$\arg\max_x \{f(x)\}$	the value of x that leads to the maximum value of $f(x)$
$\arg\min_x \{f(x)\}$	the value of x that leads to the minimum value of $f(x)$
$\|\mathbf{S}\|$	number of elements in set $\mathbf{S}$

Vectors and Matrices

$(\vec{x} \cdot \vec{y})$	inner (scalar) product of $\vec{x}$ and $\vec{y}$, i.e. $(\vec{x} \cdot \vec{y}) = \sum_i x_i y_i$
$\|a\|$	absolute value of scalar a
$\|\vec{x}\|_p$	induced p-norm of vector $\vec{x}$, i.e. $\|\vec{x}\|_p = (\|x_1\|^p + \cdots + \|x_n\|^p)^{1/p}$, $1 \leq p < \infty$; $\|\vec{x}\|_\infty = \max_i \|x_i\|$.
$\|\vec{x}\|$	Euclidean norm of vector $\vec{x}$, i.e. $\|\vec{x}\| = (\vec{x} \cdot \vec{x})^{1/2}$.
$\|\mathbf{A}\|_p$	induced p-norm of matrix $\mathbf{A}$, $\|\mathbf{A}\|_p = \sup_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p}$, $1 \leq p < \infty$
$\mathrm{D}^2(\vec{x}, \vec{y})$	Euclidean distance between vectors $\vec{x}$ and $\vec{y}$
$k(\vec{x}, \vec{y})$	a kernel function defining the inner product of vectors $\vec{x}$ and $\vec{y}$

Bibliography

[1] Chi-hau Chen, Louis Francois Pau, and Patrick S. P. Wang (ed.s), *Handbook of Pattern Recognition & Computer Vision.* World Scientific, Singapore, 2nd edition, 1993.

[2] Luc Devroye, László Györfi, and Gábor Lugosi, *A Probabilistic Theory of Pattern Recognition.* New York: Springer, 1996.

[3] Richard O. Duda, Peter E. Hart and David G. Stork, *Pattern classification*, New York: Wiley, 2001.

[4] Ian T. Jolliffe, *Principal Component Analysis.* New York: Springer-Verlag, 1986.

[5] Konstantinos I. Diamantaras and Sun-Yuang Kung, *Principal Component Neural Networks: Theory and Applications.* Wiley-Interscience, New York, 1996.

[6] Te Won Lee, *Independent Component Analysis: Theory and Applications.* Kluwer Academic Publishers, Dordrecht, 1998.

[7] V. Vapnik, *The Nature of Statistical Learning Theory.* New York: Springer Verlag, 1995.

[8] C. Cortes and V. Vapnik, "Support vector networks," *Machine Learning*, vol.20, pp.273-297, 1995.

[9] B. Schölkopf, C. Burges and V. Vapnik, "Extracting support data for a given task,' in U. M. Fayyad and R. Uthurusamy, editors, *Proc. 1st Int. Conf. Knowledge Discovery and Data Mining.* Menlo Park, Canada, AAAI Press, 1995.

[10] B. Schölkopf, C. Burges and V. Vapnik, "Incorporating invariances in support vector learning machines," in C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff, editors, *Artificial Neural Networks – ICANN'96*, pp. 47-52, Berlin, 1996. Springer Lecture Notes in Computer Science, Vol. 1112.

[11] C. Burges and B. Schölkopf, "Improving the accuracy and speed of support vector learning machines," in M. Mozer, M. Jordan and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pp. 375-381, Cambridge, MIT Press, 1997.

[12] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *Proc. Eroup. Conf. Machine Learning*, pp.137-142. Berlin, Germany, Springer-Verlag Press, 1998.

[13] S. Dumais, J. Platt, D. Heckerman, and M. Sahami, "Inductive learning algorithms and representations for text categorization," in *Proc. 7th Int. Conf. Inform. Knowledge Management 1998*, 1998.

[14] H. Durcker, D. Wu and V. Vapnik, "Support vector machines for span categorization," *IEEE Trans. Neural Networks*, vol. 10, pp. 1048-1054, 1999.

[15] S. Mukherjee, E. Osuna and F. Girosi, "Nonlinear prediction of chaotic time series using a support vector machine," in J. Principe, L. Gile, N. Morgan, and E. Wilson (ed.s), *Proc. of 1997 IEEE Workshop on Neural Networks for Signal Processing VII*, pp. 511-520. New York: IEEE, 1997.

[16] K. R. Müller, A. J. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen and V. N. Vapnik, "Predicting time series with support vector machines," in W. Gerstner, A. Germond, M. Hasler and J. D. Nicoud (ed.s), *Lecture Notes in Computer Science 1327: Artificial Neural Networks*, pp. 999-1004. Germany: Springer-Verlag, 1997.

[17] D. Mattera and S. Haykin, "Support vecotr machines for dynamic reconstruction of a chaotic system," in B. Schölkopf, C. J. C. Burges and A. J. Smola (ed.s), *Advances in Kernel Methods - Support Vector Learning*, pp. 211-242. Cambridge, MA: MIT Press, 1999.

[18] L. Xu, A. Krzyzak and E.Oja, "Rival Penalized Competitive Learning for Clustering Analysis, RBF net and Curve Detection," *IEEE Trans. on Neural Networks*, vol.4, No.4, pp. 636-649, 1993.

[19] L. Meng, K. W. Lau and Q. H. Wu, "Pattern Classification using a support vector machine based on subclass centers," in *Proc. of the 3rd Int. Conf. on Control Theory and Applications*, pp. 355-359. Pretoria, South Africa, 2001.

[20] L. Meng and Q. H. Wu, "Error-center-based optimization C a new algorithm for support vector machine training," in D. Li (ed.), *Proc. of the 5th Int. Conf. on Optimisation: Techniques and Applications*, vol. 1, pp. 462-477. Hong Kong, 2001.

[21] L. Meng and Q. H. Wu, "An error-centre-based algorithm for support vector machine training," submitted to *Electronics Letters.*

[22] L. Meng and Q. H. Wu, "Fast Training of Support Vector Machines Using Error-Center-Based Optimization," submitted to *Journal of Global Optimization.*

[23] L. Meng, Q. H. Wu and Z. Z. Yong, "A faster genetic clustering algorithm," in S. Cagnoni et al. (ed.s), *Lecture Notes in Computer Science 1803: Real World Applications of Evolutionary Computation*, pp. 22-33. Springer-Verlag, 2000.

[24] L. Meng, Q. H. Wu and Z. Z. Yong, "A comparison of genetic clustering algorithms," in *Proc. of the 2nd Int. ICSC Symposium on Engineering of Intelligent Systems*, UK, 2000.

[25] L. Meng, Q. H. Wu and Z. Z. Yong, "A genetic hard c-means clustering algorithm," to appear in *Dynamics of Continuous, Discrete and Impulsive Systems, Series B: Applications and Algorithms.*

[26] V. Vapnik, *Statistical Learning Theory.* Wiley Interscience, 1998.

[27] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in D. Haussler, editor, *5th Annual ACM Workshop on COLT*, pp. 144-152, Pittsburgh, ACM Press, 1992.

[28] V. Vapnik, S. Golowich, and A. Smola, "Support vector method for function approximation, regression estimation, and signal processing," in M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pp. 281-287, Cambridge, MIT Press, 1997.

[29] M. Schmidt, "Identifying speaker with support vector networks," in *Interface '96 Proceedings*, Sydney, 1996.

[30] M. P. S. Brown, W. N. Grundy, D. Lin, N. Christianini, C. Sugnet, T. S. Furey, M. Ares and D. Haussler, "Knowledge-based analysis of microarray gene expression data using support vector machines," in *Proc. Nat. Academy Sci.*, vol. 97, no. 1, pp. 262-267, 2000.

[31] T. Furey, N. Christianini, N. Duffy, D. Bednarski, M. Schummer and D. Haussler, "Support vector machine classification and validation o fcancer tissue samples using microarray expression data," *Bioinformatics*, vol. 16, pp. 906-914, 2000.

[32] A. Zien, G. Rätsch, S. Mika, B. Schölkopf, T. Lengauer and K. R. Müller, "Engineering support vector machine kernels that recognize translation initiation sites in DNA," *Bioinformatics*, vol. 16, pp. 799-807, 2000.

[33] D. Haussler, "Convolution kernels on discrete structures," Univ. of California Santa Cruz, Tech. Rep. UCSC-CRL-99-10, July 1999.

[34] A. J. Smola, P. Bartlett, B. Schölkopf and C. Schuurmans, *Advances in Large Margin Classifiers*. MIT Press, 1999.

[35] Guyon's web page on applications of support vector machines. http://www.clopinet.com/isabelle/Projects/SVM/applist.html.

[36] V. Vapnik, *Estimation of Dependences Based on Empirical Data* (in Russian). Nauka, Moscow, 1979. (English translation: Springer Verlag Press, 1982.)

[37] S. Geman, E. Bienenstock and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural Computation*, vol. 4, no. 1, pp. 1-58, 1992.

[38] T. M. Cover, "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *IEEE Trans. Electronic Computers*, vol. EC-14, pp.326-334, 1965.

[39] W. Karush, *Minima of Funcitons of Several Variables with Inequalities as Side Constraints.* Department of Mathematics, University of Chicago, 1939. MSc Thesis.

[40] H. Kuhn and A. Tucker, "Nonlinear programming," in *Proc. 2nd Berkeley Symposium on Mathematical Statistics and Probabilistics*, pp. 481-492, University of California Press, 1951.

[41] R. Fletcher, *Practical Methods of Optimization.* John Wiley and Sons, Inc., 2nd edition, 1987.

[42] Information systems laboratory at the University of Massachusetts, USA. http://www.ecs.umass.edu/ece/labs/isl/

[43] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Knowledge Discovery and Data Mining*, 2(2), 1998.

[44] R. Courant and D. Hilbert, *Methods of Mathematical Physics*, Interscience, 1953.

[45] N. Aronszajn, "Theory of reproducing Kernels," *Trans. Amer. Math. Soc.*, vol. 68, pp. 337-404, 1950.

[46] F. Girosi, "Equivalence between sparse approximation and support vector machines," Tech. Rep. AIM-1606, Artificial Intelligence Laboratory, Massachusetts Institute of Technology (MIT), Cabridge, Massachusetts, 1997.

[47] N. E. Heckman, "The theory and application of penalized least squares methods or reproducing Kernel Hilbert spaces made easy," 1997. ftp://newton.stat.ubc.ca/pub/mamcy/PLS.ps.

[48] B. Schölkopf, K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik, "Comparing support vector machines with gaussian kernels to radial basis function classifiers," *IEEE Trans. Signal Processing*, vol. 45, pp. 2758-2765, 1997.

[49] J. J. More and S. J. Wright, *Optimization Software Guide.* Frontiers in Applied Mathematics, vol. 14. Society for Industrical and Applied Mathematics (SIAM), 1993.

[50] B. A. Murtagh amd M. A. Saunders, "MINOS 5.4 user's guide," Technical Report SOL 83.20, Stanford University, 1993.

[51] R. J. Vanderbei, "LOQO user's manual -version 3.10," Technical Report SOR-97-08, Princeton University, Statistics and Operations Reserach, 1997. Code available at http://www.princeton.edu/rvdb/.

[52] MATLAB. *User's Guide.* The MathWorks, Inc., 1992.

[53] C. Saunders, M. O. Stitson, J. Weston, L. Bottou, B. Schölkopf, and A. J. Smola, "Support vector machines - reference manual," Technical Report CSC-TR-98-03, Royal Holloway, University of London, 1998.

[54] E. Osuna, R. Freund and F. Girosi, "Support vector machines: Training and applications," MIT A. I. Lab., A. I. Memo AIM-1602, 1996.

[55] E. Osuna, R. Freund and F. Girosi, "An improved training algorithm for support vector machines," in J. Principe, L. Gile, N. Morgan, and E. Wilson (ed.s), *Proc. of 1997 IEEE Workshop on Neural Networks for Signal Processing VII*, pp. 276-285. IEEE, 1997.

[56] T. Joachims, "Making large-scale SVM learning practical," in B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods-Support Vector Learning*, pp. 169-184. Cambridge, MIT Press, 1999.

[57] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," Technical Report MSR-TR-98-14, Microsoft Research, 1998.

[58] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods-Support Vector Learning*, pp. 185-208. Cambridge, MIT Press, 1999.

[59] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya and K. R. K. Murthy, "Improvements to Platt's SMO algorithm for SVM classifier design," Technical Report CD-99-14, National University of Singapore, http://guppy.mpe.nus.edu.sg/ mpessk.

[60] E. Osuna, R. Freund and F. Girosi, "Training support vetor machines: An application to face detection," in *Proc. Computer Vision and Pattern Recognition*, pp. 130-136, 1997.

[61] E. Osuna and F. Girosi, "Reducing run-time complexity in SVMs," in *Proc. 14th Int. Conf. Pattern Recognition*, Brisbane, Australia, 1998.

[62] L. Kaufmann, "Solving hte quadratic programming problem arising i support vector classification," in B. Schölkopf, C. J. C. Burges and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pp. 147-168. MIT Press, 1999.

[63] S. S. Keerthi, S. K. Sheade, C. Bhattacharyya and K. R. K. Murthy, "Improvements to Platt's SMO algorithm for SVM classifier design," Technical Report CD-99-14, Control Division, Dept. of Mechanical and Production Engineering, National University of Singapore, 1999.

[64] A. J. Smola, *Learning with Kernels*, PhD thesis, Technische Universität Berlin, 1998.

[65] GMD-FIRST. GMD-FIRST web site on Support Vector Machines. http://svm.first.gmd.de.

[66] T. T. Frieb, N. Cristianini and C. Campbell, "The kernel Adatrom algorithm: A fast and simple learning procedure for support vector machines," in J. Shavlik, editor, *Proc. Int. Conf. Machine Learning ICML'98*, pp. 188-196. San Mateo, Canada, 1998.

[67] J. K. Anlauf and M. Biehl, "The Adatron: An adaptive perceptron algorithm," *Europhys. Letters*, vol. 10, pp. 687-692, 1989.

[68] S. S. Keerthi, S. K. Sheade, C. Bhattacharyya and K. R. K. Murthy, "A fast iterative nearest point algorithm for support vector machine classifier design," Technical Report TR-ISL-99-03, Dept. of CSA, IISc, Bangalore, India, 1999.

[69] A. Kowalczyk, "Maximal margin perceptron," in A. J. Smola, P. Bartlett, B. Schölkopf and C. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, 1999.

[70] M. Abramowitz and I. A. Stegun, eds., *Handbook of Mathemeitcal Functions*, U. S. Department of Commerce, National Bureau of Standards Applied Mathematical Series. 55, 1964.

[71] S. P. Lloyd, "Least square quantization in PCM", *IEEE Trans. Inform. Theory*, vol. 28, pp. 127-135, 1982.

[72] Jianchang Mao and Anil K. Jain, "A self-organizing network for hyperellipsoidal clustering (HEC)," *IEEE Trans. on Neural Networks*, TNN-7(l), pp. 16-29, 1996.

[73] James C. Bezdek, *Fuzzy Mathematics in Pattern Classification.* Ph.D. thesis, Cornell University, Applied Mathematics Center, Ithaca, NY, 1973.

[74] James C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms.* Plenum Press, New York, 1981.

[75] S. Chen, "Nonlinear time series modelling and prediction using Gaussian RBF networks with enhanced clustering and RLS learning," *Electronics Letters*, vol. 31, no. 2, pp. 117-118, 1995.

[76] Allen Gersho and Robert M. Gray. *Vector Quantization and Signal Processing.* Kluwer Academic Publishers, Boston, MA, 1992.

[77] G. F. Mclean, "Vector quantization for texture classification," *IEEE Trans. System, Man and Cybernetics*, vol. 23, no. 3, pp. 637-649, 1993.

[78] J. Makhoul, S. Roucos and H. Gish, "Vector quantization in speech coding," *Porc. of the IEEE*, vol. 73, no. 11, pp. 1511-1588, 1985.

[79] R. A. Johnson and D. W. Wichern, *Applied Multivariate Statistical Analysis.* Englewood Cliffs, NJ:Prentice-Hall, 1982.

[80] L. Meng, Q. H. Wu and Z. Z. Yong, "A new genetic scheduling algorithm," in *Proc. 5th Int. Conf. Optimisation: Techniques and Applications*, vol. 1, pp. 128-144. Hong Kong, 2001.

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

[82] D. Powell and M. M. Skolnick, "Using genetic algorithms in engineering design optimization with non-linear constraints," in S. Forrest (ed.), *Proc. 5th Int. Conf. Genetic Algorithms*, pp. 424-430. Morgam Kaufmann, 1993.

[83] P. Surry, N. Radcliffe and I. Boyd, "A multi-objective aproach to constrained optimization of gas supply networks," in T. Fogarty (ed.), *Proc. AISB-95 Workshop on Evolutionary Computing*, vol. 993, pp. 166-180. Springer Verlag, 1995.

[84] D. Dasgupta and Z. Michalewicz, (ed.s), *Evolutionary Algorithms in Engineering Applications*, Springer Verlag, New York, 1997.

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

[86] E. Mayr, *Toward a new philosophy of biology: observations of an evolutionist*. Cambridge, MA: Belknap Press, 1988.

[87] D. B. Fogel, "Asymptotic convergence properties of genetic algorithms and evolutionary programming: analysis and experiments," *Cybern. & Syst.*, vol. 25, pp. 389-407, 1994.

[88] G. Rudolph, "Convergence analysis of canonical genetic algorithms," *IEEE Trans. Neural Networks*, vol. 5, pp. 96-101, 1994.

[89] T. Bäck, *Evolutionary Algorithms in Theory and Practice*, Oxford, NY, 1996.

[90] H. P. Schwefel, *Evolution and Optimum Seeking*, John Wiley, NY, 1995.

[91] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," in *IEEE Trans. Evolutionary Computation*, vol. 1, pp. 67-82, 1997.

[92] D. E. Goldberg and R. Lingle, "Alleles, loci, and the travelling salesman problem," in *Porc. an Int. Conf. Genetic Algorithms and Their Applications*, pp. 154-159, 1985.

[93] D. K. Gehlhaar, G. M. Verkhivker, P. A. Rejto, C. J. Sherman, D. B. Fogel, L. J. Fogel, S. T. Freer, "Molecular recognition of the inhibitor AG-1343 by HIV-1 protease: conformationally flexible docking by evolutionary programming," in *Chem. & Biol.*, vol. 2, pp. 317-324, 1995.

[94] S. A. Harp, T. Samad and A. Guha, "Towards the genetic synthesis of neural networks," in J. D. Schaffer (ed.), *Proc. 3rd. Int. Conf. Genetic Algorithms*, pp. 360-369. Morgan Kaufmann, San Mateo, CA, 1989.

[95] P. J. Angeline, G. M. Saunders and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," in *IEEE Trans. Neural Networks*, vol. 5, pp: 54-65, 1994.

[96] S. B. Haffner and A. V. Sebald, "Computer-aided design of fuzzy HVAV controllers using evolutionary programming," in D. B. Fogel and W. Atmar (ed.s), *Proc. 2nd Ann. Conf. Evolutionary Programming*, pp. 98-107, Evolutionary Programming Society, La Jolla, CA, 1993.

[97] S. W. Wilson, "Classifier fitness based on accuracy," in *Evol. Comp.*, vol. 1, pp. 67-82, 1997.

[98] J. R. Koze, *Genetic Programming*, MIT Press, Cambridge, MA, 1992.

[99] P. J. Angeline, D. B. Fogel, "An evolutionary program for the identification of dynamical systems," in S. K. Roger and D. Ruck (ed.s), *SPIE Aerosence 97, Symp. on Neural Networks*, vol. 3077, pp. 409-417, 1997.

[100] D. B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Order No. PC3871, IEEE Press, 1995.

[101] L. J. Fogel, A. J. Owens and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*, John Wiley, NY, 1996.

[102] D. Whitley, "The GENITOR algorithm and selection pressure," in J. D. Schaffer (ed.), *Proc. 3rd Int. Conf. Genetic Algorithms*, pp. 116-121. Morgan Kaufmann, 1989.

[103] G. Syswerda, "Uniform crossover in genetic algorithms," in D. J. Schaffer (ed.), Proc. 3rd Int. Conf. Genetic Algorithms, pp. 2-9. Morgan Kaufmann, 1989.

[104] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," in *Proc. Int. Conf. Genetic Algorithms*, pp. 14-21, 1987.

[105] K. A. De Jong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems* (Doctoral dissertation, University of Michigan), 1975. Dissertation Abstracts International, 36(10), 5140B (University Microfilms No. 76-9381).

[106] A. Brindle, *Genetic Algorithms for Function Optimization*, Ph.D. dissertation, University of Alberta, Alberta, 1981.

[107] Emanuel Falkenauer, "The worth of the uniform," in *Proc. of the 1999 Congress on Evolutionary Computation (CEC99)*, vol. 1, pp. 776-782, Washington, D.C., USA, July 1999. IEEE Press.

[108] W. M. Spear and K. A. De Jong, "An analysis of multi-point crossover," in G. J. E. Rawlins (ed.), *Foundations of Genetic Algorithms*, pp. 301-315. San Mateo, CA: Morgan Kaufmann.

[109] J. H. Holland, *Adaptation in Natural and Artificial Systems.* Ann Arbor, University of Michigan Press, 1975.

[110] N. N. Schraudolph and R. K. Belew, "Dynamic parameter encoding for genetic algorithms," *Machine Learning*, vol. 9(1), pp. 9-21, 1992.

[111] G. P. Babu and M. N. Murty, "Clustering with evolutionary strategies," *Pattern Recognit.*, vol. 27, no. 2, pp. 321-329, 1994.

[112] L. O. Hall, I. B. Özyurt and J. C. Bezdek, "Clustering with a genetically guided optimized approach," *IEEE Trans. Evolutionary Computation*, vol. 3, no. 2, pp. 103-112, 1999.

[113] F. Klawonn, "Fuzzy clustering with evolutionary algorithms," in *Proc. Seventh IFSA World Congress*, vol. 2, pp. 312-323, 1997.

[114] P. Scheunders, "A genetic c-means clustering algorithm applied to color image quantization," *Pattern Recognit.*, vol. 30, no. 6, pp. 859-866, 1997.

[115] K. A. De Jong and J. Sarma, "Generation gaps revisited," *Foundations of Genetic Algorithms 2*, D. Whitley (ed.), pp. 19-28. Vail, CO:Morgan Kaufmann, 1993.

[116] C. H. Lee and L. H. Chen, "Hign-speed closest codeword search algorithms for vector quantization". *Signal Processing*, vol. 43, pp. 323-331, 1995.

[117] J. C. Bezdek, "Cluster validity with fuzzy sets," *J. Cybernetics*, vol. 3, no. 3, pp. 58-73, 1974.

[118] T. Jones and S. Forrest, "Fitness distance correlation as a measure of problem difficulty for genetic algorithms," in L. Eshelman, editor, *Proc. Sixth Int. Conf. Genetic Algorithms*, pp. 184-192, 1995.

[119] M. R. Garey, D. S. johnson and Ravi Sethi, "The complexity of flowshop and jobshop scheduling", *Mathematics of Operation Research*, vol. 1, no. 2, pp. 117-129, May 1976.

[120] J. R. Barker and G. B. McMahon, "Scheduling the general job shop", *Manaagement Science*, vol. 31, no. 5, pp. 594-598, 1985.

[121] J. Carlier and E. Pinson, "An algorithm for solving the job-shop problem", *Management Science*, vol. 35, no. 2, pp. 164-176, 1989.

[122] R. Nakano and T. Yamada, "Conventional genetic algorithm for job shop problems", in *Proceedings of the 4th International Conference on Genetic Algorithms*, pp. 474-479, 1991.

[123] F. D. Croce, R. Tadei and G. Volta, "A genetic algorithm for the job shop problem", *Computers and Operations Research*, vol. 22, no. 1, pp. 15-24, 1995.

[124] S. Kobayashi, I. Ono. M. Yamamura, "An efficient genetic algorithm for job shop scheduling problems", in *Proceedings of the 6th International Conference on Genetic Algorithms*, pp. 506-511, 1995.

[125] S. French, *Sequencing and Scheduling*, Ellis Horwood, Chichester, 1982.

[126] E. Falkenauer and S. Bouffoix, "A genetic algorithm for job shop", *Proceedings of the 1991 IEEE international Conference on robotics and Automation*, 1991.

[127] L. W. Cai, Q. H. Wu and Z. Z. Yong, "A genetic algorithm with local search for solving job shop problmes", *Lecture Notes in Computer Science 1803: Real World Applications of Evolutionary Computation*, S. Cagnoni et al. (ed.s), pp. 107-116. Springer, 2000.

[128] G. Syswerda, "Schdule Optimization Using Genetic Algorithms", *Handbook of Genetic Algorithms*, L. Davis (ed.), pp. 332-349, Van Nostrand Reinhold, NY, 1991.

[129] P. J. M. Van Laarhoven, E. H. L. Aarts and J. K. Lenstra, "Job shop scheduling by simulated annealing", *Operations Research*, vol. 40, pp. 113-125, 1992.

[130] M. Dell'Amico and M. Trubian, "Applying tabu search to the job shop scheduling problem", *Operations Research*, vol. 41, pp. 231-252, 1993.

[131] J. F. Muth and G. L. Thomoposon, *Industrial Scheduling*, Prentice Hall, Englewood Cliffs, New Jersey, 1963.

[132] B. Schölkopf, P. Y. Simard, A. J. Smola, and V. N. Vapnik, "Prior knowledge in support vector kernels," *Advances in Neural information processings systems*, M. I. Jordan, M. J. Kearns, and S. A. Solla (ed.s), vol. 10, pp. 640-646, Cambridge, MA, 1998. MIT Press.

[133] C. J. C. Burges, "Geometry and invariance in kernel based methods," *Advances in Kernel Methods - Support Vector Learning*, B. Schölkopf, C. J. C. Burges, and A. J. Smola (ed.s), pp. 89-116, Cambridge, MA, 1999. MIT Press.

[134] J. Weston and C. Watkins, "Multi-class support vector machines," in M. Verleysen (ed.) *Proceedings of ESANN99*, Brussels, 1999. D. Facto Press.

[135] K. Crammer and Y. Singer, "On the learnability and design of output codes for multiclass problems," in *Computational Learning Theory*, pp. 35-46, 2000.