UNIVERSITY OF SOUTHAMPTON

Class-dependent features and multicategory classification

by

Alex Bailey

A thesis submitted for the degree of
Doctor of Philosophy

in the

Faculty of Engineering and Applied Science
Department of Electronics and Computer Science

7th February 2001

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

Faculty of Engineering and Applied Science

Department of Electronics and Computer Science

Doctor of Philosophy

Class-dependent features and multicategory classification

by Alex Bailey

The problem of pattern classification is considered for the case of multicategory classification where the number of classes, k, is greater than two. Many classification algorithms are in fact 2-class classifiers and are generalised to solve k-class problems. Which classifiers are naturally multicategory and the nature of the generalisation of a 2-class classifier to k classes is not often investigated. A thorough analysis of multicategory classification is given in this thesis which provides a new taxonomy of popular classification algorithms, and goes on to derive these from a probabilistic viewpoint. A clear distinction is made between classifiers that partition the input space and those that partition the set of k classes. Of the classifiers which partition the set of classes, the one-of-n, pairwise, and hierarchical methods of decomposition are shown to be equivalent in the knowledge of the true data distributions. The scaling properties of these algorithms are analysed for increasing k. The effects of learning models on finite data are then investigated to show the practical differences between each decomposition.

In classification problems with many classes it is commonly the case that different classes exhibit wildly different properties. In this case it is unreasonable to expect to be able to summarise these properties by using features designed to represent all the classes. In contrast, features should be designed to represent subsets of classes that exhibit common properties without regard to any class outside the subset. The value for classes outside the subset may be meaningless, or simply undefined. The multicategory classification schemes proposed explicitly deal with such class-dependent features, and attractive properties of these classifiers are demonstrated for a real-world handwritten digit recognition application.

Contents

List of Tables

List of Figures

Nomenclature

BFS	Best-First Search
CGD	Contour-based Gradient Distribution
DDD	Directional Distance Distribution
DFBB	Depth-First Branch-and-Bound
DFS	Depth-First Search
DTC	Decision Tree Classifier
ECOC	Error-Correcting Output Encoding
EM	Expectation Maximization
FLIERS	Fuzzy Land Information in Environmental Remote Sensing
HME	Hierarchical Mixtures of Experts
*k*NN	*k*-Nearest Neighbours
MAP	Maximum A Posteriori
MLP	Multi-Layer Perceptron
NN	Neural Network
PCA	Principle Component Analysis
RBF	Radial Basis Function
SOM	Self-Organising Map
STL	Standard Template Library
SVM	Support Vector Machine

Notation

k	the number of classes
ω_i	the ith class
Ω^{all}	the set of all classes, $\Omega^{all} = \{\omega_i;\ i = 1, \ldots, k\}$
Ω_i^{sub}	a subset of Ω^{all}
Ω^l, Ω^r	left and right class subsets (a dichotomy)
d	the number of inputs or features
	(the number of dimensions of the input space)
$\mathbf{x} \in \mathbb{R}^d$	an input vector (input point)
x_i	an element of $\mathbf{x}$ (a single input or feature)
$R = \mathbb{R}^d$	the input space
r_q	a subregion of R
$\mathbf{x} \in \omega_i$	the input points that belong to the ith class
μ_i	the mean of the set of input points that belong to the ith class
Σ_i	the covariance of the set of input points that belong to the ith class
$P(\omega_i)$	the prior probability of the ith class
$P(\omega_i \vert \mathbf{x})$	the posterior probability of the ith class given an input vector
$P(\Omega_i^{sub})$	the prior probability of a subset of classes
$P(\Omega^{all})$	the prior probability of the set of classes $P(\Omega^{all}) = 1$
$P(\mathbf{x} \vert \omega_i)$	the conditional probability of the input vector given the ith class
$P(\omega_i \vert \mathbf{x}, \Omega_j^{sub})$	the posterior probability of a class given the input and a class subset
$P(\omega_i \vert \mathbf{x}, r_q)$	the posterior probability of a class given the input
	and a subregion of input space
$\mathbf{w}, b$	the weight vector and bias of a linear discriminant
n_t	the number of training points for a class

Acknowledgements

I would like to thank my supervisor, Chris Harris, and the internal and external examiners, Steve Gunn and John Edmunds.

I would also like to thank Steve Gunn and Jaz Kandola for the use of their code, John Manslow, and Adam Prugel-Bennet for their helpful discussion and clarity of thought, Graham Bent, and Martin Brown for their valuable involvement, and Sharon and Tony for their moral support.

I thank also the EPSRC and IBM UK for their support of this work.

Chapter 1

Introduction

1.1 Problem overview

In a recent review paper Jain *et al.* define pattern recognition as the following (Jain, Duin and Mao, 2000):

> Pattern recognition is the study of how machines can observe the environment, learn to distinguish patterns of interest from their background, and make sound and reasonable decisions about the categories of the patterns.

This is a complex task that is an innate ability for humans; the recognition of people by their face, their voice, or their gait, or the recognition of writing on a page, are skills which people often take for granted, but to design a machine to solve such problems poses formidable research challenges. Other pattern recognition applications such as the diagnosis of diseases, the recognition of aircraft, and the categorisation of consumers in market research are examples of more difficult tasks for humans where we may look to machines to aid our decisions. This interesting and challenging topic has received a wealth of research interest since the use of electronic computers in the 1950's and still today remains a very active research area.

There are many aspects of pattern classification that are still open research fields from learning theory and model combination, to feature extraction and feature selection. Interestingly, multicategory classification is often overlooked as a research topic in its own right, shown by its absence in Jain, Duin and Mao (2000). This thesis covers the field of multicategory classification which involves the classification of patterns into a finite set of classes where the number of classes is greater than two, and studies the effects for a subset of multicategory classifiers as the number of classes increases significantly. The problems being addressed are firstly the fact that current methods in multicategory classification are unrelated and seldom compared. This is addressed via a probabilistic formulation by which equivalences between important algorithms can be shown, and algorithms can be clearly placed in a novel taxonomy. This then highlights similarities and differences between algorithms to allow an analytical comparison. Secondly the generalisation of a binary problem to a multicategory problem is often considered to be a simple extension where one class is discriminated from the set of all other classes. This is but one method of generalising a binary problem and alternative methods are brought together and compared in this thesis. Within this framework, the novel concept of class-dependent features is investigated and shown to reduce the number of features used in multicategory classification without a loss in accuracy.

Several existing popular classification algorithms are investigated in terms of their ability to discriminate many classes. The lack of scalability of multi-layer perceptrons for problems of many classes is argued and the lack of cohesion between the different algorithms is highlighted. The heuristic nature of decision tree classifiers and the combination of submodels in pairwise classifiers is also questioned.

In response to these problems a novel formulation of Bayes' rule for statistical pattern classification is given that makes the conditioning on the input space and the set of classes explicit. This then allows the clear distinction to be made between multicategory classifiers that solve the problem of many classes by partitioning the input space, and multicategory classifiers that partition the set of classes. A novel taxonomy of these classification algorithms is given that makes this clear distinction between inherent multicategory classifiers such as the k-nearest neighbour classifier and the decision tree classifier that partition the input space, inherent multicategory classifiers that partition the set of classes, and inherent 2-class classifiers such as linear discriminants and generalised linear discriminants (see Chapters 2 and 3 for further details on these classifiers). Known classification algorithms that partition the set of classes to solve classification problems by a combination of binary classifiers are then clearly placed in this taxonomy. These are known as the one-of-n classifier, the hierarchical classifier, and the pairwise classifier and are grouped under the term *class-decomposition classifiers*. They differ in the number of binary subproblems and the method used to combine the outputs of each binary subproblem to give the final multicategory classification output (details are given in Chapter 4). These are then analysed in their scalability for problems with many classes in terms of the number of submodels to train, the training complexity of each submodel, and the number of models evaluated during classification.

One of the most important results of this new taxonomy is the clear distinction between a decision tree classifier, and a hierarchical classifier that combines 2-class subproblems. These two classification algorithms are very similar in nature; they both rely on a hierarchy of decision problems and take advantage of the fact that a complex problem can be broken down into a set of simpler subproblems. But they are distinct and the fundamental differences are shown in Chapter 4. Much research has concentrated on such tree-based classifiers, but this novel distinction is not addressed.

The probabilistic framework is then used to analyse the class-decomposition classifiers. Statistical pattern classification is a branch of pattern classification that uses the well-founded rules of probability, and in this context a step in deciding the final class output is the estimation of a vector of posterior probabilities. The three most popular class decomposition classifiers, namely the one-of-n, hierarchical, and pairwise classifiers are derived such that they generate the correct posterior probabilities, and are shown to be equivalent when the required probability densities are known perfectly. Such perfect knowledge arises in the hypothetical situation where the data used to estimate the probability densities is infinite.

However, for finite datasets, issues of generalisation and model complexity arise when learning each submodel in a class decomposition. It is shown in Chapter 4 that there are two fundamental components to the complexity of the submodels in class decomposition classifiers. These are:

- the complexity that arises from the boundary that divides any two classes from each other, and
- the complexity that arises from discrimination between many classes.

Although the first component is fixed for a given application dataset, the second is affected by the choice of the class decomposition and can increase with the number of classes. The one-of-n classifier introduces complexity that increases with the number of classes. The pairwise classifier has the mimimum complexity that is constant for an increasing number of classes. The complexity introduced by the hierarchical classifier is controlled by the choice of the class hierarchy and can be minimised by the suitable identification of this class hierarchy.

It is interesting to consider the simple and important linear discriminant, and the effect of using such linear models as the submodels in the class decomposition algorithms. Kressel (1999) concludes that pairwise linear classifiers deserve further attention due to 'good recognition results with extremely low classification requirements'. The pairwise, hierarchical and one-of-n decompositions using linear classifiers are thoroughly investigated in this thesis in Chapters 4 and 7. The pairwise and hierarchical decompositions are shown to work well using linear discriminants for each submodel, while the one-of-n decomposition degrades seriously for problems with more than two classes. This is due to the complexity introduced between many classes by the one-of-n decomposition.

Each decomposition has other advantages and disadvantages, such as the number of submodels to train and then evaluate during classification. The pairwise classifier suffers from having to train a number of models in the order of k^2 where k is the number of classes, whereas the one-of-n and hierarchical need only train k, and $k-1$ models respectively. But the design of the class hierarchy for the hierarchical classifier is not trivial. Techniques that attempt to solve this problem are presented in Chapter 5.

One significant advantage to the hierarchical model is that, of the $k-1$ submodels, in some cases only $\log_2 k$ models need to be evaluated to find the maximum posterior probability. A principled method for selecting the models to evaluate is given in Chapter 4. Also the class hierarchy offers a more transparent model via interpretation of the class subgroupings. An example of this is given in Chapter 7 on remote sensing data.

A direct result of the class decomposition algorithms is that feature selection may be done locally for each submodel. Work in parallel with the work in this thesis is presented in (Oh, Lee and Suen, 1999), where the effect of such feature selection is investigated for the one-of-n decomposition, which was shown to give an improvement in classification accuracy when compared to using the same global set of features for each submodel. The motivation behind such local feature selection is given by investigating the class-dependent nature of features. Class-dependent features are defined and shown to occur naturally in typical classification problems in Chapter 6. In a recent review paper on automatic handwriting recognition (Plamondon and Srihari, 2000), the author suggests that for structural approaches to handwriting recognition to be effective, a classifier will have the property that ‘the number of features used to describe a class of patterns may vary from one class to another’. This is exactly the property that is being investigated in this thesis, although unfortunately it is beyond the scope of this thesis to study structural handwriting recognition. A statistical approach to handwriting recognition is presented in Chapter 7 which shows that the use of class-dependent features is advantageous for all three class decomposition algorithms.

To show evidence for the theory presented in the thesis a number of experiments on simulated and real-world datasets are described in Chapter 7. The real-world data is taken from the field of handprinted digit recognition. These show favourable results, and demonstrate that for linear models the pairwise classifier is by far the most accurate classifier, followed by the hierarchical classifier and then the one-of-n classifier. The accuracy of the class decomposition classifiers is compared using local and global feature selection showing that local feature selection results in better classification performance for fewer features selected. For non-linear models, multi-layer perceptron classifiers are used and in this case the classification accuracy for each class decomposition becomes approximately equivalent, but with the hierarchical classifier performing marginally better. The advantages of local feature selection still remain.

The concluding remarks state that when solving a multicategory classification problem of many classes, although they are asymptotically equivalent, hierarchical and pairwise classifiers have a considerable advantage in terms of classification performance over traditional one-of-n classifiers when using either linear or non-linear submodels. Features should be selected independently for each submodel leading in simpler models without loss of accuracy. The hierarchical model is particularly advantageous since it has the fewest submodels, and the complexity of the submodels can be controlled by identifying a suitable class hierarchy.

1.2 Thesis outline

A chapter-by-chapter summary of the thesis is given in this section. The thesis is divided into eight chapters including this introduction.

Chapter 2 defines concepts central to statistical pattern classification, namely the definition of features and classes, and describes discrimination, generalisation, and regularisation in pattern classification. Standard statistical techniques such as Bayes' rule and linear discriminants are detailed on which later chapters are built.

Chapter 3 goes on to investigate the previous research on multicategory classification including one-of-n, pairwise, the various types of decision tree classifiers and hierarchical classifiers. The lack of a common framework for multicategory classification is highlighted. Although there is little research directly on class-dependent features, related research is described that has similar motivations. Hierarchical and pairwise classifiers are noted for their compatibility with the class-dependent feature paradigm.

Chapter 4 then lays down a probabilisitic framework under which popular multicategory classification algorithms can be derived according to whether they partition the input space or the set of classes. This leads to a novel taxonomy of multicategory classification. In this chapter an equivalence between the one-of-n, pairwise and hierarchical class decomposition classifiers is shown and the difference between decision tree classifiers and hierarchical classifiers is made explicit. The scalability of the three algorithms is shown for increasing problem sizes, and two forms of complexity in a multicateogry problem are distinguished. The pairwise classifier is shown to reduce the complexity due to many classes, the hierarchical classifier can control this complexity through design of the hierarchy, and the one-of-*n* classifier maximises this complexity.

In Chapter 5 the necessary structure identification techniques are described for the hierarchical classifier. Performance metrics such as classification accuracy, and approximations such as the Euclidean distance between class means are shown, and top-down, or bottom-up clustering methods are described. The structure identification is also formulated as a discrete search which is enabled due to the finite number of nodes in the class hierarchy. Efficient operators for combinatorial otimization are described that reduce the cost of retraining when using classification rates as an objective function.

Chapter 6 defines and explores the novel concept of class-dependent features as related to the statistical independence of the feature and the class variable, given a subset of classes, and relates it to existing work. Illustrative examples from real-world applications are shown and the novel distinction between strong and weak class dependence is defined. Local and global feature selection methods are described for the class-decomposition classifiers using motivation for class-dependent features. Classifiers using local feature selection are expected to use fewer features.

Chapter 7 describes experiments on real and simulated data to confirm predictions made in previous chapters, with the experimental results. Firstly the object-oriented design of the software used to carry out the simulations is described. Then the classification accuracy of the class-decomposition models is shown as the number of classes increase using both logistic linear discriminants and multi-layer perceptron classifiers for each submodel. The one-of-*n* classifier is shown to scale badly, while the hierarchical and pairwise models are shown to scale well. Then the classification rate for all three submodels is shown to be consistently better for fewer features selected using local feature selection which makes use of class-dependent feature metrics.

Chapter 8 concludes with a discussion of the work presented and avenues for future research in this area, particularly in the reduction of the number of models in the pairwise classifier and the analysis of strong class-dependent features.

1.3 Contributions

The main contributions of this work are detailed below.

- Novel taxonomy of multicategory classification algorithms (Section 4.2).
- Probabilistic formulation of multicategory classifiers and equivalence analysis between models (Sections 4.1 to 4.5).
- Analysis of scalability of one-of-n, pairwise and hierarchical multicategory classification models (Sections 4.6 to 4.8) with experimental evidence on simulated data (Section 7.3).
- Principled approach to computationally beneficial approximations to hierarchical classifier (Section 4.10.2).
- Definition and analysis of strong and weak class-dependent features (Section 6.1) with experimental evidence on real-world data for weak class-dependent features extending the work of Oh, Lee and Suen (1999) (Section 7.6).

1.4 List of publications

T.J. Dodd, A. Bailey and C.J. Harris (1998). A data driven approach to sensor modelling, estimation, tracking and data fusion. In M. Bedworth and J. O'Brien, editors, *EuroFusion98*, pages 103–111.

C.J. Harris, A. Bailey and T.J. Dodd (1998). Multi-sensor data fusion in defence aerospace. *The Aeronautical Journal*, 102(1015):229–244.

A. Bailey and C.J. Harris (1999). Using a hierarchical classifier to exploit context in pattern classification for information fusion. In *Proceedings of the 2nd Conference on Information Fusion (FUSION99)*, Sunnyvale, CA, pp. 1196-1203. ISIF.

Chapter 2

Features and classes for pattern classification

This thesis is concerned with pattern classification. A pattern is a pair of variables (Schurmann, 1996):

$$\text{Pattern} = [\mathbf{x}, \omega],$$

where $\mathbf{x} \in \mathbb{R}^d$ is a vector of observations, and $\omega \in \Omega$ is a label that represents a meaningful concept in the problem domain. For example $\mathbf{x}$ may be the digital image of a face, or of a handwritten character, or the digital recording of a spoken word, and ω may represent a person, a letter of the alphabet, or a word, respectively.

Pattern classification is a mapping from $\mathbf{x}$ to ω. It is most often assumed that the observation vector is of fixed length, d, and that Ω is a set of class labels ω_i for finite $i = 1, \ldots, k$. This constrains the problem in such a way that allows machine learning algorithms to attempt to solve the problem. Normally a model is specified by using a set of known observation and label pairs to define the model structure and parameters. This set of patterns is known as the training set, or training sample. A similar but different set of patterns is used to evaluate the classification accuracy of the model. This is known as the test set, or sample.

To reliably distinguish between classes there must be a certain amount of discriminatory information present in the observations. A simple example being, if the task was to distinguish between images of the handwritten letters 'a' and 'b', then a possible model would be to simply measure their height and designate anything above a certain height as a 'b'. However, there is much variation in the height of handwritten letters so this system will be rather inaccurate. To aid the decision, a vertical stroke could be detected on the left of the letter and then the two measurements combined before deciding what type of letter it is. The extra discriminatory information in the second measurement will most likely increase the accuracy of the classification system. There will still be some confusion over certain images, so more measurements will have to be taken, and perhaps with a more sophisticated measuring technique. Problems will arise if too many measurements are taken on a small population sample, and this, known as 'the curse of dimensionality', is an extremely important issue which affects the size and complexity of models and consequently their learning ability. This states that with finite training samples, arbitrarily increasing the number of measurements can lead to an eventual decline in the overall classification performance.

The concept of features is also described in this chapter. A feature may be simply a measurement on a object, or a combination of measurements on an object. The processes of feature selection and feature extraction are defined and distinguished. It is the relationship between features and classes that is primarily of interest here, the restriction that a feature should be applicable to all classes is being relaxed to allow features to expressly represent subsets of classes without requiring the value of that feature to be meaningful for classes outside such subsets.

This thesis presents a novel taxonomy of multicategory classification which states that a many-class problem should be approached by either partitioning the input space R, or the set of classes Ω. It is shown that when decomposing Ω the distinction between certain classes is best described by an individual feature set for each individual subset of Ω. Central themes in pattern classification are laid down in this chapter to define the issues that are important to the discussion in later chapters.

2.1 Class concepts and observations

It should be noted here that class labels represent arbitrary concepts and there is no reason beyond common sense that the class labels should represent any meaningful categorization of the problem. It would of course be illogical to try and solve a meaningless problem, and most class labels will in fact represent meaningful and distinct concepts, though class concepts should not be thought of as formally distinguishable.

This thesis considers disjoint class concepts such that any observation can only truly belong to one class. There is a set of valid problems by which an observation can belong to more than one class concept at a time. Fuzzy classification and probabilistic class mixture estimation can address such problems of overlap between class concepts but this is a different problem and is not considered in this thesis. A discussion of

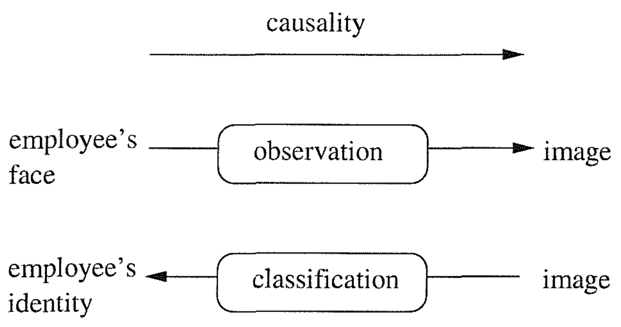


Figure 2.1: Classification is an inverse of the complex process of observation

fuzzy classification and its relationships to statistical pattern classification is given in (Manslow, 2000).

An illustrative example, if each observation **x** represented an image of a face then Ω might be a set of employees, or the set $\{\text{male}, \text{female}\}$, or even the set of the twelve signs of the zodiac. Each of these in theory are valid classification problems except that in practice it would be difficult to tell someone's star sign by looking at their face, due to the fact that there will be little correlation between the class labels and the observations. This is due to the poor design of the classification system through a poor choice of inputs. It may also be the case that two of the employees are identical twins where, although to a lesser extent, there will be ambiguity in the problem definition. This is the result of the more fundamental problem that pattern classification is an inverse process. This is illustrated in Figure 2.1. The process of observation is a many to many mapping and its inverse may be ambiguous. It follows that for some problems the observations will not be sufficient for perfect classification.

Even if the class concepts are defined wisely there may be ambiguity imposed by the observational process. If a system is to classify objects, it must be able to observe them. Observation can be an incredibly complicated and uncertain process which in itself accounts for most of the problems in pattern classification. Observations are a result of a complex non-linear combination of three types of information:

- the observable properties of the object itself,
- the observational conditions, and
- any noise effects resulting from the measurement device.

The observable properties are what one hopes to measure, and this would be a person's face. The observation conditions will affect how the face is observed, it may

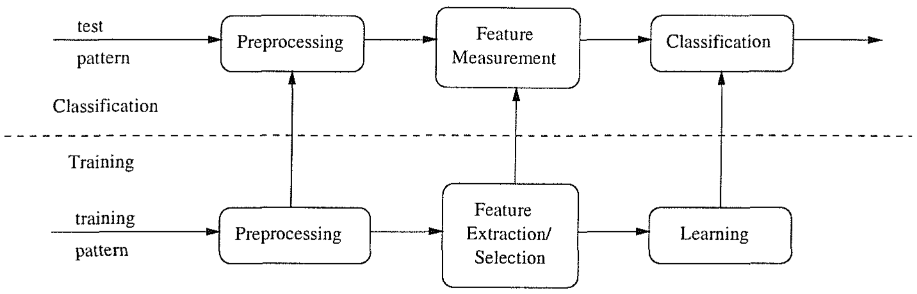


Figure 2.2: Model for statistical pattern recognition, from (Jain, Duin and Mao 2000)

be dark, or the person may be covering his face with a hand, sunglasses, or an item of clothing. The camera may introduce measurement noise by electromagnetic interference, or simply by poor focusing or insufficent resolution. All these will have an effect on the final image supplied to the classifier and it is the task of the classification system to cope with these effects. A broad outline of how a classification system infers the class label from such an observation is given in the next section.

2.2 A typical classification system

The classifier is one of several processes in the classification system, albeit the most important process, but there are other processes that are typically undergone before information is fed to the classifier.

To minimise the noise introduced in the observations one might design an appropriate physical system, for example ensure adequate lighting, use a camera with a suitable level of image detail etc. And once the observations have been recorded, then these are presented to the classification system which can take further steps to minimise the effects of the inevitable measurement noise that has been recorded. A typical statistical pattern classification system is shown in Figure 2.2 (Jain, Duin and Mao, 2000). The first step is usually known as pre-processing and involves such processes as filtering to reduce the measurement noise, and normalisation to reduce the variance due to the observational conditions. More sophisticated dimension reduction methods such as principle component analysis (PCA) may be used to reduce the number of dimensions in the input vector via an eigenspace transformation. This will result in a pre-processed input vector.

When training a classifier the features can then be extracted from the preprocessed input vector to give a feature vector. Then the most appropriate features are selected and the reduced feature vector is input to the classification system for learning, which will calculate the output error and adapt the model parameters. When classifying, the set of features are measured, or calculated, according to the optimum set chosen by the feature extraction and selection processes. The classifier can then evaluate its output according to the measured feature set. These and related concepts are described in the following sections although specific details will depend on the application problem.

2.3 Features

In a pattern classification problem, features are generated by functions of the raw or pre-processed measurement data. These functions are called *feature extractors*. The feature extraction process may be complex transformations of the measurement data such as the Fourier transform, or a simple copying of the measurement values.

When designing a machine learning system, there is always a trade-off between the complexity of the feature extractors and the complexity of the classification algorithm. If simple feature extractors are used for a complex problem then the classifier will need enough complexity to learn the class concepts from these simple features. However, if complex feature extraction processes are used that allow for most of the complexity of the problem, then a simpler classification algorithm may be sufficient. Usually prior knowledge would be needed to design complex feature extractors that adequately represent the problem.

An illustrative example which will be returned to thoughout the thesis is the handwritten digit recognition problem. Two pre-processed digits are shown in Figure 2.3. In this case the raw pixel values may be presented to a classifier, but a complex classifier would be needed to be able to accurately classify the digits as no application knowledge has been fed into the problem.

Alternatively, application-specific feature extractors, in this case sophisticated techniques to detect loops and lines, or estimate the slant and curvature of the strokes in the image could be designed. Then the classifier design would be matter of defining a simple rulebase that classifies digits according to these high-level features, which may even be possible to do by hand.

The trade-off is between the human design effort and the amount of learning expected of the machine. In the first case the machine is expected to learn all the invariances of the raw data and completely parameterise the problem in such a way that it can be solved. In this case the machine is effectively learning feature extractors

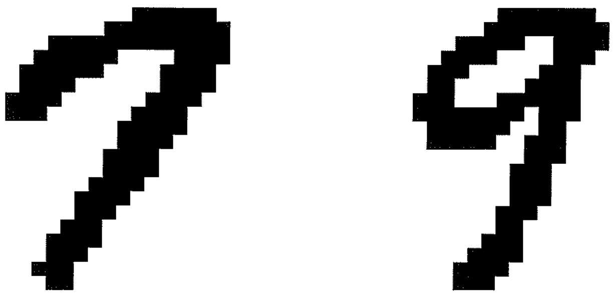


Figure 2.3: An example of a digitised handwritten digit with and without a loop

internally. In the second case, much prior knowledge is being used to design features such that they incorporate known invariances and extract the informational content of the raw data.

This trade off is important in this thesis since transparency and prior knowledge are of interest. If we expect the machine to learn everything about the problem - if such a task is tractable; then it is often the case that the solution of the problem gets lost in the parameters of the system and little insight can be gained as to how the problem is being solved. Also in most real-world problems there will be a base of prior knowledge that can be incorporated into the system, which should be used whenever possible.

Another important property of features is the associated cost of evaluating a feature. The cost may be in the time taken to calculate the numerical value of the feature from raw data, the risk involved in measuring the feature, or even the monetary cost of measuring the feature. This will depend on the application and in some cases it is important to minimise this cost and to evaluate the smallest number of features possible when making a classification decision. It is shown in Chapters 4 and 7 that the hierarchical model is particularly efficient in reducing the number of features to evaluate during classification, which can also result in a more interpretable model.

Some example applications are:

- Time-critical systems where the time taken to evaluate complex features needs to be reduced to a minimum.

- Medical systems where measuring a feature may involve costly tests, or even surgery.
- Pharmaceutical research involving the identification of molecular compounds.
- Strategic defence systems where measuring a feature using an active sensor such as radar involves a risk of being detected.
- Marketing analysis where measuring a feature involves large-scale surveys.

In these situations the need to reduce the number of features is more critical than the possible effects of the curse of dimensionality and feature reduction techniques are important.

2.3.1 Feature extraction and feature selection

Feature extraction and feature selection provide an important role in classification problems in reducing the complexity of a classifier and reducing the overall cost of feature measurements.

Feature extraction is the transformation, or combination, of inputs based directly on the raw data to provide more meaningful, abstract and concise features for use by the pattern classifier. This may involve a simple transformation of the whole data for data compression, such as principle component analysis, or a complex domain specific process, such as locating geometric features such as the eyes in a face recognition task. Several distinct feature extraction processes may be applied to the raw data in parallel to generate a vector of features. The importance of feature extraction is to condense maximally useful information from the raw data available, while retaining the discriminatory features/attributes.

Alternatively, feature selection is the process of choosing the minimum set of the available features that provide the maximum discrimination between classes when presented to the pattern classifier. These features may be those generated from a feature extraction process, the raw data itself, or a mixture of the two. The importance of feature selection is to select only those features that are relevant to the decision being made. For example, in the handwritten digit recognition problem, if the pixel values were to be used as features, then feature selection would concentrate on the pixels in the main body of the image and ignore the uninformative pixels around the extremities of the image.

Pattern classification is known to perform best using the minimal set of features that contain the maximum information relevant to the problem.

2.3.2 Invariance

Invariance is a very important concept regarding feature extraction. When dealing with a set of objects from a single class, it is reasonable to expect some variation between the properties of the individuals, although there will be properties which are invariant for all the individuals in that single class. For example if the classification task was to classify a selection of shapes into the three classes $\{\text{circle}, \text{triangle}, \text{square}\}$, then all the individuals might vary in their size and orientation. However the number of vertices is invariant for all members of the same class. Since the number of vertices varies between members of different classes it would make a good discriminant for the three class problem.

Assuming that we do have meaningful and distinct class concepts, it is the notion of invariance that should be captured in a particular feature extraction technique. A feature should be invariant, that is insensitive, to properties of the measurements that are irrelevant to the problem. For example, in object recognition we are not interested in the size or orientation of the object in the image. The feature extractors should therefore be scale and rotation invariant. Certain image processing techniques such as the Fourier transform are known to be translation invariant, and this would make a good feature extractor if the degree of translation of an image is a property of the measuring device or observational conditions, and not a property of the problem we are trying to solve. An earlier example suggested the height of a handwritten character should be used to distinguish the letters ‘a’ and ‘b’. But this will not be scale invariant. A better feature would be the ratio of width to height, called the aspect ratio, since this is now scale invariant. It is easily seen that the aspect ratio would not be a good feature when discriminating the letters ‘a’ and ‘e’ since they will typically have similar values.

A feature should also have within-class invariance, such that the value of the feature is invariant for all points taken from a single class. If there is a small within-class invariance and a large inter-class variance then that feature is a good discriminant for the problem. For the purposes of this thesis, which deals with class subsets, it is also important to consider properties which are invariant for sets of classes. For example, the set of digits {‘0’, ‘6’, ‘8’, ‘9’} has a small within-set variance for the loop feature since all the digits within it contain a loop. However, the loop feature would be good at discriminating this whole set of classes from another set of classes such as the ‘straight line’ numbers such as {‘1’, ‘7’}.

2.4 Discrimination

The task of a classifier is to discriminate between the classes. Discrimination is ultimately achieved by placing a decision boundary between points in feature space. This decision boundary may be explicitly represented by a parametric form, such as in a linear discriminant, or indirectly through the parameterisation of the class distributions used to estimate posterior probabilities. Other non-parametric classifiers such as k-Nearest Neighbour techniques do not explicitly describe a decision boundary but it can be seen that the effective boundary is a fine-grained tessalation of regions around the set of stored representative points when $k = 1$. See Figures 4.5 and 4.6 for examples of decision boundaries in multicategory classification.

This thesis is concerned primarily with statistical pattern classification and the estimation of class-conditional distributions which lead to parametric decision boundaries, or so called semi-parametric cases such as multi-layer perceptrons. As with all learning systems, classifiers are governed by the laws of learning theory and the most important terms are explained in the section below.

2.4.1 Bias, variance, generalisation and regularisation

Generalisation is the most important concept in machine learning and data modelling (Bishop, 1995). A model must be able to learn relationships between the inputs and outputs in the training data and then be able to make predictions on unseen data. Generalisation is the ability to extract the 'essence' of the problem without learning the specifics of the training sample. Rote learning is an example of a very bad generaliser, since new problems can only be solved if they have been seen before in *exactly* the same manner, that is if they are to be found in the training set. The other extreme is an over-general model that treats all input samples as the same, and outputs a constant value. In the presence of finite data, good generalisation is achieved by a balance between model bias and model variance. Regularisation is a technique to improve the generalisation ability of a model. These terms are explained below.

In data modelling, bias and variance are conflicting properties of a model concerning its flexibility. A model is said to exhibit high model bias if it has a rigid inflexible structure, such as a linear model (see Figure 2.4). A flexible model such as one based on nth-degree polynomials is said to have less bias, as it is flexible enough to fit to many more problems. However, a flexible model will inevitably have high model variance since it can vary its structure to fit itself too closely to the data - resulting in poor generalisation (see Figure 2.5). Also the number of features in the feature vector

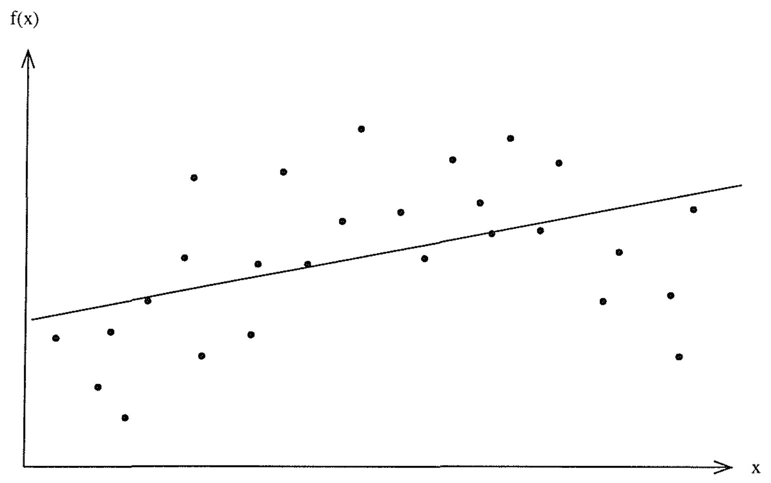


Figure 2.4: A linear model exhibiting high bias and low variance

has an effect on the generalisation performance since for most parametric classifiers the number of parameters increases with the number of features. A large number of parameters means a more flexible model and the chance of overfitting is higher.

Regularisation is a technique to overcome high variance by constraining the model in some way. A simple and elegant solution is to constrain the model to have low curvature (see Figure 2.6). This is in effect stating the requirement that small changes in the model inputs should lead to small changes in the model output. This is a sensible requirement for good generalisation.

Regularisation is also a form of applying prior knowledge. From a Bayesian viewpoint, specifying constraints on the system before exposing it to any data is equivalent to imposing prior knowledge upon the system.

When comparing the performance of different classification algorithms, the classification rate is the most representative measure of accuracy. In other words the percentage of correct classifications on a sample dataset. However this must be a comparison of generalisation ability. This simply requires that the classification rates quoted must be performed on *unseen* data samples that have not been used in the training or design of the classifier. But for some applications it is not always the accuracy which is the dominating factor in choosing a classifier. Issues such as training time, classification time, storage requirements, and interpetability can be important enough to use a classifier with an accuracy less than the state of the art.

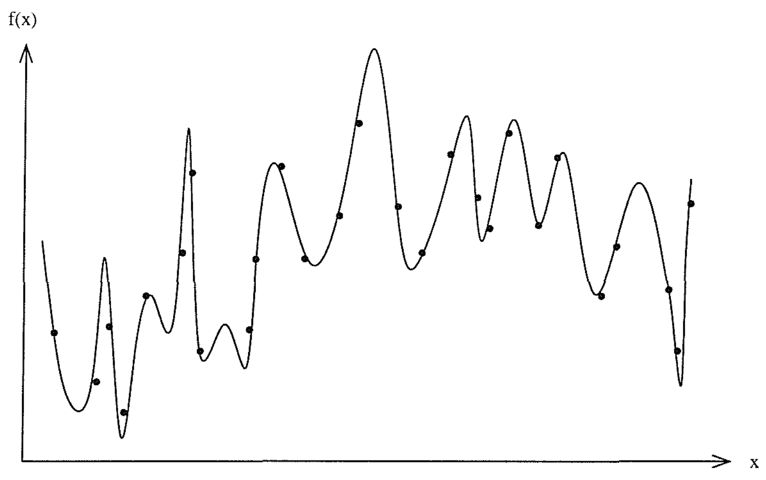


Figure 2.5: A non-linear model with low bias and high variance over-fitting the data

2.4.2 Effects of large sets of classes

There is a significant difference between a 2-class classification problem and a k-class classification problem where $k > 2$. A 2-class or binary problem is special because it can be represented by a single inequality. This can lead to a simpler problem analytically and is the preferred problem for classifier design. Many classification algorithms are described in terms of a binary decision problem, since the general case of m classes is assumed to be a trivial extension of a two class problem. This assumption is challenged in this thesis and the effects of discrimination between large sets of classes are addressed.

There are several ways of converting a k-class problem into a set of 2-class problems. They have their advantages and disadvantages and the most common methods are detailed below:

- One-of-n output encoding: Each class is in turn discriminated against a set of all the other classes. A total of k models are learnt. Model outputs are compared directly and the class for the model with the highest output is chosen.
- Pairwise classification: Each class is discriminated against each other class separately. A total of $k(k-1)/2$ models are learnt. Model outputs are combined additively and the class with the highest output is chosen. (Jia and Richards 1998; Kressel 1999)
- Hierarchical classification: The set of classes is split into left and right subets

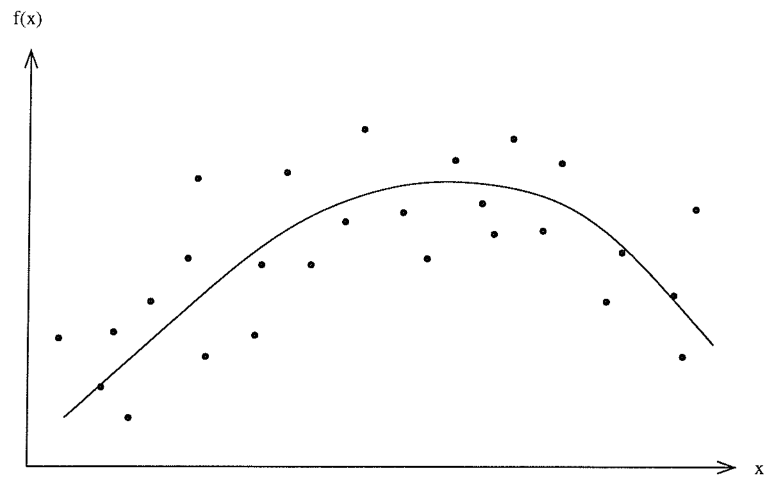


Figure 2.6: A flexible model with constrained curvature reduces the variance

which are discriminated against, the subsets are then split recursively into left and right subsets until all classes have been discriminated. A total of $k-1$ models are learnt. Model outputs are multipled and the class with the highest output is chosen (Schuermann and Doster 1984).

The number of features that can be effectively used at each stage may vary depending on the algorithm. Issues arise when the number of classes increase, such as the number of models to be learnt and the complexity of the tree structure for hierarchical models. These issues are addressed fully in Chapter 4.

2.4.3 The number of features

With a small number of features there will be increased ambiguity at the decision boundaries in feature space. It is expected that the data points for separate classes will overlap severely in feature space unless the few features represent all the discriminatory information for the problem. With many features the dimensionality of the feature-space will increase, leading to a general increase in distances between points (a natural property of high-dimensional spaces), leading to low data density. However problems will arise from learning models in high dimensional spaces if there are insufficient training samples to learn from. This effect is described in the next section.

The curse of dimensionality is known by several names; Hughes' phenomenon, or the peaking phenomenon. A discussion of the peaking phenomenon is given in (Trunk, 1979). Hughes' phenomenon is described here, but the idea is essentially the same.

2.4.4 Hughes' phenomenon

In his 1968 paper, Hughes analysed the mean accuracy of two-class pattern classifiers with discrete-valued feature vectors (Hughes, 1968). The mean accuracy is formulated in terms of the dimensionality of the measurement vector d, the number of data points and the prior probability of one of the classes. The mean accuracy is written as $P_{cr}(d,n,P(\omega_1))$ where n is the number of data points, and $P(\omega_1)$ is the prior probability of class ω_1. Hughes showed that in the case of infinite data, increasing the number of measurements increases the mean accuracy asymtotically to its maximum value, which depends on the prior probabilities.

However, in the usual case of a finite number of data points, it was shown that there is an optimal measurement dimension d after which the mean accuracy begins to drop. The optimum dimension of the measurement vector is a function of the number of data points, increasing with greater volumes of data.

This is now a well known effect in machine learning and is known as either Hughes' phenomenon, or *the curse of dimensionality*. It is the motivation for feature selection described below.

There are several intuitive explanations for Hughes' phenomenon, namely that as the number of dimensions of a measurement space increases, more data points are needed to accurately specify the probability distributions in the high-dimensional space. If the number of data points is fixed (as is often the case when training a pattern classifier) then arbitrarily increasing the number of measurements will eventually degrade performance.

This can be explained in terms of non-parametric probability density estimation, where the measurement space is spanned by a set of bins, and the density estimate is calculated as the number of points falling in the bin, divided by the volume of the bin. If each measurement axis is divided into h bins then as the number of dimensions d increases the number of bins increases as h^d. There will obviously soon come a limit when there are not enough points to distribute amongst all the bins that represent areas of non-zero probability density. In this case there will be bins that register zero probability density erroneously. The number of erroneous bins will increase dramatically with d, rendering the overall probability estimates inaccurate.

2.4.5 Cover's theorem

Cover's theorem (Cover, 1965) simply stated says that a complex pattern-classification problem recast non-linearly in a high-dimensional space is more likely to be linearly separable than in a low-dimensional space. Linear separability is desirable in a pattern classification problem, as there is an easy and analytical solution in the linear separable case.

Cover's theorem might appear counter-intuitive to Hughes' phenomenon since it states that transforming a low-dimensional problem into a higher-dimensional provides an easier problem, and consequently a more accurate classifier. It should be noted that Hughes' phenomenon still applies, however if the separating hyperplane is determined properly in the high demensional space, then this may be overcome to such an extent that Cover's theorem provides a good basis for designing a pattern classifier.

The problem with mapping a problem into a higher dimensional space and then fitting a linear separating hyperplane is that in many dimensions there are many more possible hyperplanes than in the lower space, increasing the likelihood of choosing a non-optimal one. By use of regularisation theory one can constrain the problem to give a unique solution which is optimal given the constraints. This is how kernel-based methods such as the SVM operate (Burges, 1998).

2.5 Classification and interpretability

There now exist two possible avenues for designing a pattern classification system. One is to use a particularly flexible model with good generalisation capabilities such as a Support Vector Machine (SVM) and train it with the raw fixed-length observation vectors. This is a relatively new technique that has been proven to give very good classification performance. Another viewpoint is to select and design good feature extractors that represent the structural invariances in the problem and learn a structured classifier based on the features generated by these feature extraction algorithms.

The former will shed little light on the process of classification, the internal mappings of a SVM will, for a problem of any complexity, be far too abstract to be meaningful. However the latter approach is termed 'interpretable' since the feature extractors can be designed to have meaning in terms of the problem and the final classification process can then be understood in such a way that sheds light on the solution of the problem. (There is current research within the ISIS Research Group on an interpretable SVM algorithm termed 'SUPANOVA' by Steve Gunn (Gunn and Brown, 1999), and a Neurofuzzy SVM (Chan and Harris, 2001)).

However, when designing feature extractors that are meaningful for a classification problem with many classes it soon becomes apparent that not all feature extraction processes are 'meaningful' for all the classes. This can be illustrated via the handwritten digit recognition problem which is addressed throughout this thesis.

A meaningful feature in an image of a handwritten digit is the presence of a loop (see Figure 2.3 for an example of a preprocessed digit with and without a loop). In turn, if a loop is found further analysis may be performed on the detected loop. Further features such as the relative position of the centre of the loop from the centre of the whole digit can provide discriminatory information. Though, naturally, if there was no loop detected in the first place then the 'relative position' of the loop is meaningless. Whether a value may be assigned at all to these features, or if a value can be generated by some algorithms even for images without loops then those values cannot be expected to be at all meaningful.

The treatment of such features, which are termed here as *class-dependent features*, requires much further anyalsis. For a classifier of many classes to be interpretable then these questions need to be addressed. Chapter 6 deals with this issue in greater detail.

For now the concept of conditional independence is introduced which can be used to measure the class-dependent nature of features. Conditional independence (Pearl, 1988) is defined for events, A, B, and C such that:

$$P(A|B,C) = P(A|C),$$

if A and B are *conditionally independent* given C.

This is an important concept regarding class-dependent features since it can be formulated as:

$$P(\omega|x_i,\Omega) = P(\omega|\Omega).$$

This means that the class variable, ω, and a particular feature, x_i can be independent conditioned on the set of classes, Ω. In other words, the dependence between features and classes may vary given a set of classes as a context.

To illustrate this, if the feature x was the number of legs, and the class variable was to be one of the following $\Omega = \{\text{ants}, \text{spiders}, \text{flies}\}$; then given $\Omega = \{\text{ants}, \text{flies}\}$ then x and ω are independent. That is, knowing x tells you no information about ω, since $x = 6$ for either class. Whereas conditioned on $\Omega = \{\text{spiders}, \text{flies}\}$, x and ω are dependent as x will now be either 6 or 8, depending on the value of ω. It is this property which will be shown to be exploited in the analysis of class-dependent features.

2.6 Statistical pattern classification

The following section follows standard techniques used in statistical pattern recognition, and forms the basis for the remainder of this thesis. For a more lengthy discourse the reader is recommended to read the excellent texts by Bishop (1995), Schurmann (1996) and Webb (1999).

The goal of statistical pattern recognition is to estimate the posterior probabilities for each class in a set of classes, $\Omega^{\text{all}} = \{\omega_i; i = 1, \ldots, k\}$. This is usually written for each ω_i as $P(\omega_i|\mathbf{x})$. In this thesis the condition on the set of classes is made explicit, since techniques are considered where this conditioning is important. So the final goal for each classification algorithm in this thesis is to estimate $P(\omega_i|\mathbf{x}, \Omega^{all})$.

To decide the class label using the information given by the posterior probabilities, a decision rule is applied. The Bayes decision rule for minimising the risk is stated as follows (Jain, Duin and Mao, 2000): Assign input pattern $\mathbf{x}$ to class ω_i for which the conditional risk

$$R(\omega_i|\mathrm{x}) = \sum_{j=1}^{k} L(\omega_i, \omega_j) P(\omega_j|\mathbf{x}) \tag{2.1}$$

is mimimum, where $L(\omega_i, \omega_j)$ is the loss incurred in deciding ω_i when the true class is ω_j and $P(\omega_j|\mathbf{x})$ is the posterior probability.

The maximum a posteriori (MAP) decision rule is then derived by using the straightforward 0/1 loss function which is described by:

$$L(\omega_i, \omega_j) = \begin{cases} 0 & \text{if } i = j \\ 1 & \text{if } i \neq j. \end{cases} \tag{2.2}$$

In this case the Bayes decision rule simplifies to the following: Assign input pattern $\mathbf{x}$ to class ω_i if

$$P(\omega_i|\mathbf{x}) > P(\omega_j|\mathbf{x}) \; \forall j \neq i. \tag{2.3}$$

It is useful at this point to define precisely the probabilities involved in statistical pattern recognition and declare their meaning. This will help to clarify the analysis below.

Classification can be thought of as the observation of an object and the subsequent labelling of that object according to the observations made. The observation and the class labels can be thought of as random variables to be modelled via probability density functions. The observation is a vector, $\mathbf{x}$, of measurements on an object and the class label, C_i, is a category to which that object can be meaningfully assigned.

It is important to stress that the class labels form a finite set, Ω^{all}, and the object is assumed to be represented by one of the class labels.

The following probabilities have the following meanings:

- $p(\mathbf{x})$ - the probability that a certain observation is made (regardless of the type of object).
- $p(\mathbf{x}|\omega_i)$ - the probability that a certain observation is made given that the object being observed belongs to class C_i. This is called the *class-conditional probability* for class ω_i.
- $P(\omega_i)$ - the probability that the object being observed belongs to the class ω_i before any observation is made.
- $P(\Omega_i^{\text{sub}})$ - the probability that the object being observed belongs to the set of classes Ω_i^{sub} before any observation is made.
- $P(\Omega^{\text{all}})$ - the probability that the object being observed belongs to the global set of classes, Ω^{all} (this is always 1 due to the closed world assumption).
- $P(\omega_i|\mathbf{x})$ - the probability that the object being observed belongs to the class ω_i given the observations $\mathbf{x}$. This is called the *posterior probability* for class ω_i.
- $P(\Omega_i^{\text{sub}}|\mathbf{x})$ - the probability that the object being observed belongs to the set of classes Ω_i^{sub} given the observations $\mathbf{x}$.

Traditionally it is the case that Ω^{all} is fixed and a classifier is used to discriminate all the classes in the global set Ω^{all} at once, resulting in a vector of posterior probabilities $[P(\omega_1|\mathbf{x}), P(\omega_2|\mathbf{x}), \ldots, P(\omega_k|\mathbf{x})]$. The elements of this vector are computed using Bayes' theorem, for $i = 1, \ldots, k$:

$$P(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_i)P(\omega_i)}{p(\mathbf{x})}. \tag{2.4}$$

The above equation is in its most common form, where the closed world assumption is implicit. For the purposes of this analysis it is necessary to make this assumption explicit and restate Bayes' theorem as:

$$P(\omega_i|\mathbf{x},\Omega^{\text{all}}) = \frac{p(\mathbf{x}|\omega_i,\Omega^{\text{all}})P(\omega_i|\Omega^{\text{all}})}{p(\mathbf{x}|\Omega^{\text{all}})} \quad \forall\omega_i \in \Omega^{\text{all}}. \tag{2.5}$$

Using Bayes' theorem the classifier designer now has the choice to estimate the prior distributions $P(\omega_i|\Omega^{\text{all}})$ and class-conditional densities $p(\mathbf{x}|\omega_i,\Omega^{\text{all}})$ from the data. The normalisation factor can be calculated as

$$p(\mathbf{x}|\Omega^{\text{all}}) = \sum_{\omega_i \in \Omega^{all}} p(\mathbf{x}|\omega_i,\Omega^{\text{all}})P(\omega_i|\Omega^{\text{all}}). \tag{2.6}$$

One advantage of Bayes' rule is that the prior probability can be defined to represent any *a priori* knowledge about the classification problem if the proportions of the classes in the training set is not thought to be representative of the true prior distribution.

Two principle approaches can be followed, that of estimating the class-conditional densities and priors, or the direct estimation of the posterior probability distribution. Note that Vapnik (1998) promotes bypassing the estimation of any such densities or distributions and concentrating all the learning on the parameterisation of the decision boundary between classes in two-class problems. This has been termed 'transduction' and leads to the increasingly effective and popular Support Vector Machine (SVM) classifier which is derived from the principles of learning theory. While this technique has parallels and influences on the work in this thesis, these issues are dealt with specifically and a thorough treatment is not made nor is necessary here of SVMs.

2.6.1 Estimating class-conditional densities

If we are taking advantage of the decomposition offered by Bayes' theorem, then we require techniques for estimating the class-conditional densities $p(\mathbf{x}|\omega_i,\Omega^{all})$ (again the conditioning on the complete set of classes is made explicit). Several standard techniques are available, and these may be categorised into parametric and non-parametric density estimation methods.

By far the most popular parametric form is the Gaussian normal form which is defined as, for a multivariate observation vector (DeGroot, 1989):

$$p(\mathbf{x}|\omega_k) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mu_\mathbf{k})^T\Sigma^{-1}(\mathbf{x}-\mu_\mathbf{k}).\right\}$$

The maximum likelihood estimate of this form is usually prefered due to its closed form and the statistics, $\hat{\mu}$ and $\hat{\Sigma}$ are estimated by the following:

$$\hat{\mu} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}^n, \text{and} \tag{2.7}$$

$$\hat{\Sigma} = \frac{1}{N-1}\sum_{n=1}^{N} (\mathbf{x}^n - \hat{\mu})(\mathbf{x}^n - \hat{\mu})^T. \tag{2.8}$$

Another important distribution is the Bernoulli distribution which describes the probability of a vector of binary events. This has direct application to image classification. The input vector $\mathbf{x}$ consists of d elements $x_i; i = 1, \ldots, d$. Each $\mathbf{x}$ represents an image. Each x_i can take the values 0 or 1. The probability of a specific input vector element, or pixel, for a given class can be written using the Bernoulli distribution as:

$$p(x_i|\omega_j) = P_{ji}^{x_i}(1 - P_{ji})^{(1-x_i)}, \tag{2.9}$$

where P_{ji} is the probability that the ith pixel from an image of the jth class is a foreground pixel.

Under certain assumptions the posterior probabilities of a Bayes' classifier using either a Gaussian or Bernoulli distribution, depending on the nature of the vector $\mathbf{x}$, can be shown to be represented by a logistic linear discriminant. This is shown in the next section.

Alternative methods for density estimation do not require a strict parameterisation, but instead use the sample of points in the dataset to define the density at specific points in the input space. In *kernel-based* density estimation methods a region is defined in the input space. The most simple being the *Parzen window* which is defined as:

$$H(\mathbf{u}) = \begin{cases} 1 & \text{if} |u_j| < 1/2, \ \forall j = 1, \ldots, d \\ 0 & \text{otherwise.} \end{cases}$$

where $\mathbf{u} \in \mathbb{R}^d$ is a point in d-dimensional space. This is an indicator function that is one if all elements of $\mathbf{u}$ are between $-1/2$ and $1/2$. This describes a hypercube of size 1 centered on the origin. To define a quantity that indicates that a point $\mathbf{x}_i$ lands within a hypercube of side h centered on the point $\mathbf{x}$ we can write $H((\mathbf{x} - \mathbf{x}_i)/h)$. This has a volume of h^d where d is the dimensionality of the input space.

By considering the proportion of data points landing in this volume an estimate of the density at point $\mathbf{x}$ can then be formulated as:

$$p(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{h^d}H\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right).$$

The field of probability density estimation is still an open research field, and in practice the choice of which density model to use is influenced more by the constraints on the design of the system and the properties of the application data. In this thesis Gaussian distributions tend to be favoured for class-conditional densities due to their simplicity, and Parzen window density estimation is used for the class-conditional overlap metric in Chapter 6 as a numerical technique to avoid Gaussian integration.

However the use of Bayes' theorem is not required if one formulates the posterior probabilities directly. The advantages of formulating a posterior probability directly is that the complexity of the model is concentrated on the boundary between the classes and the detail of the class-conditional densities far from the decision boundary is ignored, since it has little effect on the final classification rate. The estimation of posterior probabilities is covered in the next section.

2.6.2 Estimating posterior probabilities

If one prefers to avoid the estimation of class-conditional densities then it is possible to directly estimate the posterior probabilities. This again can be done either parametrically or non-parametrically. Popular parametric forms can be derived from assumptions made on the class-conditionals via Bayes' theorem.

If a linear parametric form is used to estimate the posterior probabilities then this can be shown to be equivalent to assuming Gaussian class conditionals where the priors are equal for all classes and the Gaussian covariances are restricted to strictly diagonal non-zero values (Bishop, 1995).

It is interesting to note that popular classifiers such as decision tree classifiers and k-Nearest Neighbours are effectively estimating a non-parametric posterior density through local decomposition of the input space. Decision tree classifiers split the input space into a (usually) axis-orthogonal hyper-rectangle and then return the class label with the greatest proportion with the specified region (see Figure 3.8 for an example). The class proportions given the specific region are a local non-parametric estimate of the posterior distribution. This is discussed further in Section 4.1.

The k-NN algorithm does much the same, except the region of input space is determined by a distance metric (usually Euclidean) that encloses a specific number of points and then the class proportions are totalled given this set of points and the class with the greatest proportion is chosen.

2.7 Linear discriminants

The use of linear discriminants is of importance later in the thesis and this section follows that of Bishop (1995) to show that the outputs of a linear discriminant using a sigmoid activation function can be interpreted as posterior probabilites under the assumptions of normally distributed classes of equal covariance.

Consider a two-class problem in which the class-conditional densities are given by Gaussian distributions with equal covariance matrices $\Sigma_1 = \Sigma_2 = \Sigma$, so that

$$p(\mathbf{x}|\omega_k) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mu_\mathbf{k})^T\Sigma^{-1}(\mathbf{x}-\mu_\mathbf{k})\right\}. \tag{2.10}$$

Using Bayes' theorem, the posterior probability of membership of class ω_1 is given by

$$\begin{aligned} p(\mathbf{x}|\omega_k) &= \frac{p(\mathbf{x}|\omega_1)P(\omega_1)}{p(\mathbf{x}|\omega_1)P(\omega_1)+p(\mathbf{x}|\omega_2)P(\omega_2)} && (2.11)\\ &= \frac{1}{1+\exp(-a)} && (2.12)\\ &= g(a) && (2.13)\end{aligned}$$

where

$$a = \ln\frac{p(\mathbf{x}|\omega_1)P(\omega_1)}{p(\mathbf{x}|\omega_2)P(\omega_2)} \tag{2.14}$$

and the function $g(a)$ is the logistic sigmoid function given by

$$g(a) = \frac{1}{1+\exp(-a)}. \tag{2.15}$$

If we now substitute expression for the class-conditional densities from Equation 2.10 into Equation 2.14 we obtain

$$a = \mathbf{w}^T\mathbf{x}+w_0 \tag{2.16}$$

where

$$\mathbf{w} = \Sigma^{-1}(\mu_1 - \mu_2) \tag{2.17}$$

$$w_0 = -\frac{1}{2}\mu_1{}^T\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_2{}^T\Sigma^{-1}\mu_2 + \ln\frac{P(\omega_1)}{P(\omega_2)}. \tag{2.18}$$

This shows that using a logistic linear discriminant (represented by $y = g(a)$ using Equations 2.16 and 2.15) is directly equivalent to estimating Gaussian class-conditionals of equal covariance and then calculating the posteriors via Bayes' rule. Thus the outputs of the logistic linear discriminant can be interpreted as posterior probabilites.

Unfortunately the assumption of Gaussian distributed classes with equal covariance is somewhat restrictive. However, in practice a linear discriminant can produce an accurate classifier in many cases where this assumption does not hold. In fact if the two classes are linearly separable (defined in Section 4.6) then the linear discriminant can achieve perfect classification for that particular dataset. The real issue though is whether the linear model generalises appropriately and this can be assessed by a measure such as the cross-validation classification rate to determine the suitability of the linear discriminant to a particular problem. If the linear model is found to be inadequate then techniques suggested in Section 4.11 may be used to initialise a more flexible model such as a MLP that can handle more general distributions of the data.

Also by a similar argument the logistic linear discriminant arises if the observations are vectors of binary values using a Bernoulli distribution under the assumption that the binary vector elements are statistically independent (see Bishop (1995)).

The input vector $\mathbf{x}$ consists of d elements $x_i; i = 1, \ldots, d$. Each x_i can take the values 0 or 1. The probability of a specific input vector for a given class can be written using the Bernoulli distribution as:

$$p(x_i|\omega_k) = P_{ki}^{x_i}(1 - P_{ki})^{(1-x_i)}, \tag{2.19}$$

where P_{ki} is the probability that the vector element x_i for class k has the value 1. If we now assume that the vector elements are statistically independent the probability for the complete input vector is given by the product of the probabilities of the individual vector elements:

$$p(\mathbf{x}|\omega_k) = \prod_{i=1}^{d} P_{ki}^{x_i}(1 - P_{ki})^{(1-x_i)}. \tag{2.20}$$

Again using this formulation for the class-conditional density in Bayes' rule, it can be shown that a logistic linear discriminant can be formulated such that:

$$P(\omega_1|\mathbf{x}) = g(\mathbf{w}^T\mathbf{x} + w_0), \tag{2.21}$$

where $g(a)$ is given by Equation 2.15, and

$$\begin{aligned} w_0 &= \sum_i \ln\frac{1-P_{1i}}{1-P_{2i}} + \ln\frac{P(\omega_1)}{P(\omega_2)} && (2.22)\\ w_i &= \ln\frac{P_{1i}}{P_{2i}} + \ln\frac{1-P_{1i}}{1-P_{2i}}. && (2.23)\end{aligned}$$

For a more detailed derivation please refer to Bishop (1995). This has significance when classifying binary images, although the assumption of statistically independent vector elements is broken.

2.8 Summary

The core concepts that underly statistical pattern recognition have been introduced in this chapter. A classification problem can be broken down in to four core concepts and these are summarised as follows:

- **The complexity of the problem** - this is governed by the number of classes and the number of types of observations (features). The amount of information present in the observations determines the fundamental uncertainty in the problem.

- **The complexity of the model** - the form of the model must be chosen. It may be have many or few parameters, and there may be a choice in the number of features used by the model. This will determine how well the model can at best represent the problem.

- **Training the model to a suitable accuracy** - the parameters of the model must be estimated from finite data. The model must be trained to provide good generalisation performance.

- **The interpretability of the model** - the model may be required to provide an added understanding of the problem. In this case the model structure must be easily understood and represent the underlying problem well.

Usually the problem is predefined and its complexity in the number of classes and number of available features is fixed. The tasks of choosing the complexity of the model and training the model parameters are inter-related and form the bias/variance dilemma. The overall aim is to provide the model with good generalisation ability. Over-simple models suffer due to an inability to represent the problem (strong model-bias), but over-flexible models need to have their complexity (high model variance) controlled in a sensible manner through regularisation. The most accurate classifiers tend to have many parameters and a complex structure that is controlled in this way, however they tend to be hard to interpret. If it is required that the model structure and parameters aid the understanding of a problem, then this usually requires simple models and few parameters, which affects how a designer might approach the bias/variance dilemma. In this thesis interpretability is an issue, and this leads to the requirement that models should have the fewest number of parameters and an easily understood structure.

The discussion here has concentrated on standard techniques, and in the next chapter the paradigms involved specifically with the main themes of this thesis, namely multicategory classification and class-dependent features are explored. The relationship between features and classes has been emphasised. The amount of discriminatory information provided by a single feature given a subset of the possible classes has been defined by way of conditional independence. This is expanded upon in Chapter 6 to define the term class-dependent features. An analysis in terms of an increasing number of classes follows in Chapter 3 and Chapter 4. This leads to a good understanding of the complexity of the algorithms presented, and sheds light on the real problem of choosing the right model complexity for a fixed-size problem.

Chapter 3

Review of existing research

In Chapter 2, pattern classification was defined and the important issues that underlie the field in general were highlighted. In this chapter, the issues behind the central themes in this thesis are considered, namely multicategory classification and class-dependent features. This is presented in the context of existing research in pattern classification and although there is little other research specifically on class-dependent features, it is possible to find other research approaches that consider the problem of class-dependent features indirectly.

The following concepts are of relevance:

- Multicategory classification
- Hierarchical and pairwise classification models
- Class-dependent features

These concepts and related work is discussed in the following sections.

3.1 Classification with many classes

Much of modelling and learning theory has concentrated on the 2-class problem (Friedman, 1997), this often allows rigorous analysis without concern for practical issues that arise in many real multiclass problems. The statement that a 2-class problem generalises trivially to a k-class problem often accompanies such analysis. While this is a sensible statement for small k, there are many applications that require efficient solutions to problems of many classes such as handwriting recognition (especially Chinese), face recognition, speaker recognition etc. In these cases the effects of many classes will have some bearing on the accuracy and efficiency of the different classifiers used. The important issue is the scalability of the multiclass techniques for increasing number of classes k. Approaches to multiclass problems with many classes are reviewed in this section with a qualitative analysis of the scalability problem. A more rigorous discussion is given in the next chapter with quantitative analysis.

An excellent survey of classical statistical pattern recognition is given in Jain, Duin and Mao (2000), although a discussion of the important subclass of multicategory classification is conspicuous by its absence in this otherwise comprehensive review paper. Of the classification algorithms considered in this survey it is important to consider their potential applicability to multicategory classification.

2-class Algorithms	Multiclass Algorithms
Logistic Classifier	k-Nearest Neighbour Classifier
Fisher's Linear Discriminant	Template Matching
Perceptron	Nearest Mean Classifier
Multi-Layer Perceptron	Subspace Classifier
Radial Basis Function Classifier	Bayes' Plug-in Classifier
Support Vector Machine	Parzen Classifier
	Decision-Tree Classifier

Table 3.1: Suitability of established classification algorithms to 2-class and multiclass problems

In Table 3.1 the classification algorithms reviewed in Jain et al. (2000) are divided into two categories: firstly those which are naturally 2-class classifiers, and secondly those which are inherently multiclass classifiers. This distinction needs some clarification as all these algorithms can be used for multicategory classification.

A classifier is assumed to be a natural 2-class classifier if its formulation is derived from a 2-class problem. This is most obvious for the set of linear discriminant classifiers since they are formulated in the most simple case as

$$y = \mathbf{w}^T\mathbf{x} + b, \tag{3.1}$$

where the observation $\mathbf{x}$ is assigned to class ω_1 if $y < 0$ and class ω_2 if $y > 0$. A class label is chosen from $\{\omega_1, \omega_2\}$ arbitrarily if $y = 0$. This is obviously a purely binary classifier, and a combination of such classifiers is needed to solve a multiclass problem. The above formulation describes a perceptron, and is generalised to a logistic classifier by applying the logistic function (Equation 2.15) to the output. Fisher's linear discriminant is also of this form, but provides a closed-form solution to finding $\mathbf{w}$ and b.

Multi-layer perceptrons, radial basis-function classifiers, (Bishop, 1995) and support vector machines (Burges, 1998) are types of generalised linear discriminants and map the inputs into an alternative 'hidden' space which is then fed to a linear discriminant. In this sense they are considered binary classifiers, since in each case a binary classifier is used for the output. The next section covers the methods used to generalise these binary classifiers to many class problems.

Alternatively, the classifiers in Table 3.1 that are considered inherent multiclass classifiers are derived from a multiclass problem, and need no extra formulation. The general multiclass classifiers can be grouped into three categories:

- Bayes' plug-in classifiers (including nearest mean and Parzen classifier),
- subspace classifiers (including kNN and template matching), and
- decision tree classifiers.

The first category transforms the classification task into a density estimation problem, since class conditional densities are modelled and then 'plugged into' Bayes' rule, which results in the required posterior probabilities. The nearest mean can be derived from a Bayes' classifier under assumptions of equal priors and covariances. The Parzen classifier uses Parzen density estimation (Bishop, 1995) to model the class-conditional densities in Bayes' rule. Subspace classifiers and decision tree classifiers can be shown to estimate local posterior probabilities conditioned on where the observation vector lies in the input space; this is shown in Chapter 4.

3.2 Generalising binary classifiers to many classes

If a set of 2-class linear classifiers were to be used to distinguish each class in a k-class problem, then the most common choice would be to train a set of k classifiers such that each one discriminates one class from the remaining set of classes.

In this case each classifier is given the task of distinguishing between two class subsets Ω_i^l and Ω_i^r, $i = 1, \ldots, k$, for the k models, which are defined as follows:

$$\Omega_i^l = \{\omega_i\}, \tag{3.2}$$

$$\Omega_i^r = \{\omega_j; \forall j \neq i\}. \tag{3.3}$$

When using a perceptron or linear discriminant it is often the case that the desired output for each model given a specific input, which is commonly known as the training target, is represented as a vector for a point assigned to class j (Nilsson, 1965),

$$\mathbf{t} = (t_1, \ldots, t_k)^T,\ t_i = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

In this case there are k separate problems being learnt, in each one a single class is being discriminated from all the others. This is illustrated for linear discriminants in Figure 3.1.

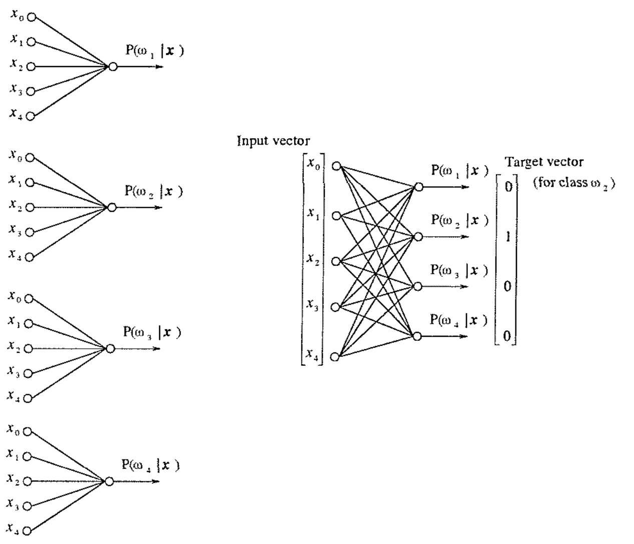


Figure 3.1: Using logistic linear discriminants to discriminate k classes Each x_i is an element of the input vector. These are weighted and summed at the output node. The output represents a posterior probability when passed through the sigmoid function.

It should be noted that only $k-1$ models are needed to discriminate k classes. The posterior probabilities must add up to 1 due to the closed-world assumption, and the value of the last model may be calculated from the first $k-1$ models. However, in practise the outputs are not normalised, and no longer sum to unity. It is therefore usually the case that k models are evaluated.

The diagram on the right of Figure 3.1 is merely a more compact way of representing the same set of models. There is no added interaction between the models. This is due to the single-layer nature of a perceptron or linear discriminant. The case for

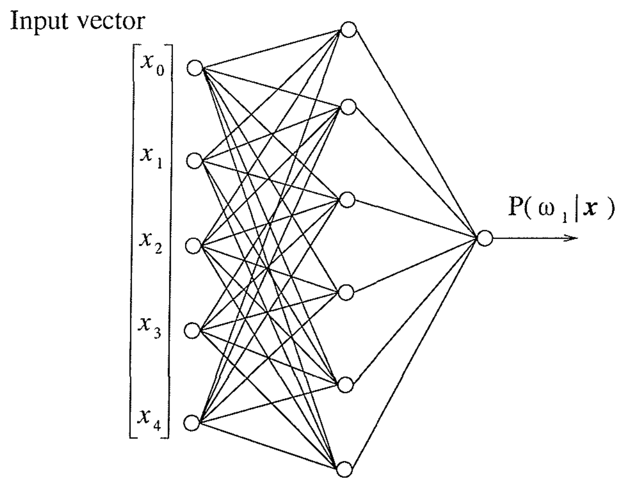


Figure 3.2: Using a MLP discriminate 2 classes, $\{\omega_1, \omega_2\}$

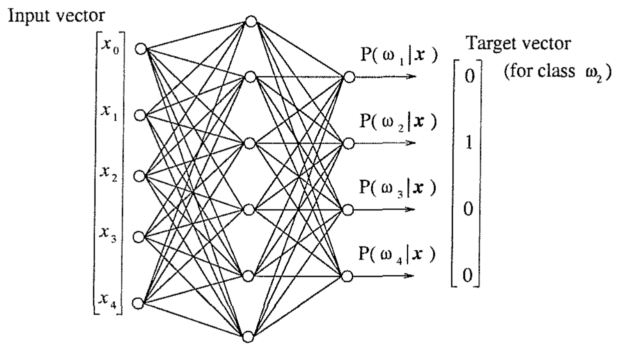


Figure 3.3: Using a MLP discriminate 4 classes, $\{\omega_1, \omega_2, \omega_3, \omega_4\}$

multi-layer perceptrons is different since there is some interaction between the layers. This is illustrated in Figures 3.2 and 3.3.

Figure 3.2 shows a multi-layer perceptron of five input nodes, six hidden nodes and one output node. This model outputs the probability of class ω_1 in a binary decision between classes $\{\omega_1, \omega_2\}$. The logistic linear hidden nodes partition the original feature space by way of linear ridge functions in the input space. This is written

$$\phi_j = g(\mathbf{w}_j^T \mathbf{x} + b_j), \tag{3.4}$$

where $g(.)$ is the sigmoid function (Equation 2.15), $\mathbf{w}_j$ is the weight vector for the jth hidden node, and b_j the bias for the same hidden node. The logistic linear output node then combines these partitioned regions to form a decision boundary between the two classes. This is written

$$y_i(\mathbf{x}) = \sum_{j=1}^{h} w_{ij}\phi_j(\mathbf{x}) + c_i, \tag{3.5}$$

where h is the number of hidden nodes and each $w_{ij}, j = 1,\ldots,h$ form the weight vector, and c_i the bias, for the ith output (in this case there is only one output). The number of hidden nodes is a result of the complexity of the decision boundary between the two classes, and in practice is often found empirically. This is generalised to a multiclass problem in much the same way as the perceptron, by adding extra linear output layers to predict the posterior probabilities for each class from the hidden node feature space as in Figure 3.3.

A foreseeable problem with this technique as a scalable multicategory classifier is that more hidden nodes will have to be added to cope with the complexity of the decision boundaries between several classes. The hidden to output weights will then select which hidden nodes will be used to construct the decision boundaries between neighbouring classes. With many classes, only a few of the total number of hidden nodes will relate to the boundary about any one particular class, and many of the hidden to output layer weights will be zero. Much research has gone into training MLPs and a good training algorithm will no doubt find a sufficient parameterisation for such a problem, but the process will often involve a particularly complex and time consuming non-linear optimisation for which there is no guaranteed solution. When the problem is well understood, it is better to use a dedicated technique to solve the multiclass problem and not rely on an uneconomical optimisation technique to do the work. This is discussed further in Sections 4.7.1 and 4.8.

Radial basis function classifiers work on much the same principle except that each hidden node represents a kernel function, usually of the form

$$\psi_j = \exp\left(-\frac{||\mathbf{x}-\mu_j||^2}{2\sigma_j^2}\right), \tag{3.6}$$

where $\mathbf{x}$ is the input vector; μ_j is the centre, and σ_j controls the radius, of the jth basis function. A linear output layer is defined on these basis functions to give the output for class ω_i as

$$y_i(\mathbf{x}) = \sum_{j=1}^{m} w_{ij}\psi_j(\mathbf{x}) + b_i, \tag{3.7}$$

where m is the number of basis functions and each $w_{ij}, j = 1, \ldots, m$ form the weight vector, and b_i the bias, for the ith output. Again, a certain number of basis functions will give rise to the decision boundaries between neighbouring classes and many of the weights w_{ij} will be zero for a problem with many classes.

Support vector machines have been developed predominantly as binary classifiers, although recently formulations of multicategory classifiers using SVMs have been developed. A suitable approach is given by Kressel (1999) where the k-class problem is decomposed into a set of 2-class problems. This leads nicely into the next section in which alternative approaches to generalising binary classifiers are reviewed including such pairwise discriminants.

3.3 Alternative approaches to generalising a 2-class problem to k classes

Although by far the most common approach, the methods used to generalise a set of 2-class classifiers to a k-class problem described above are not the only options. Alternative research approaches are reviewed in the following.

In a recent tech report (Friedman, 1996), Friedman presented a formulation of a pairwise classifier where each class is compared against every other class by a set of pairwise classification models. The class with the most positive classifications over all models is assigned as the output label. A similar approach is presented as a method of generalising a 2-class support vector machine to a k-class problem, again by combining a set of pairwise models (Kressel, 1999). Good performance is shown for the pairwise model on a handwritten digit recognition problem, even when using simple linear discriminants. In (Jia and Richards, 1998) multicategeory classification is interpreted as a cascade of binary classifiers where pairs of classes are compared. Figure 3.4 illustrates the structure of this classifier, although the pairwise nature of the binary decisions are the same as Friedman (1996) and Kressel (1999).

The pairwise techniques described above are similar in motivation, but differ in the methods used to combine the outputs of each pairwise model to reach a final classification. Jia and Richards (1998) use a method that rejects outright a class that fails any one of the pairwise tests. Friedman (1996) examines a winner-takes-all algorithm that chooses the class label with the most favourable comparisons, however this needs a heuristic for evaluating tie-break situations. Kressel (1999) supplies a heuristic for evaluting such tie-break situations, or suggests that tie-breaks should be rejected. A principled approach to the combination rule for the classifiers is lacking, and it is

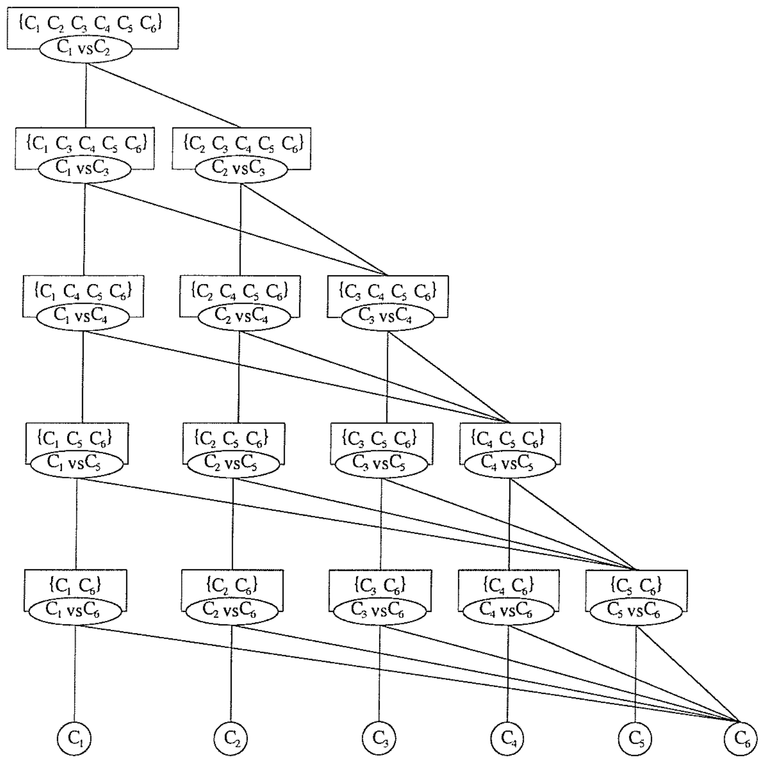


Figure 3.4: Progressive two class discriminant for six classes

shown in Chapter 4 that by using a probabilistic formulation, a principled combination scheme can be derived.

Another technique for multiclass classification is presented by Dietterich and Bakiri (1995) which investigates the use of an output encoding motivated from error-correcting codes often used in data communication. A codebook of binary vectors is generated such that the Hamming distance between all vectors is maximised. These vectors are then arbitrarily assigned to classes as output vectors where a one-of-n output encoding would normally be used. These error-correcting output codes (ECOCs) are longer than their counterpart one-of-n output vectors for many-class problems, but the paper shows a significant improvement in performance when using ECOCs with multi-layer perceptrons and decision trees. This suffers from the need to find a suitable codebook (an example is given in Figure 3.5). The size of the codebook is in the order of $2^{k-1} - 1$ columns for k classes, and a 2-class classifier needs to be learnt for each column in the codebook. Finding a suitable codebook is an open research area.

Yet another approach is to treat the multicategory problem as a set of subproblems which can be represented as a tree, where the terminal nodes of the tree define which

$$\begin{array}{c|ccccccccccccccc|}
 & f_0 & f_1 & f_2 & f_3 & f_4 & f_5 & f_6 & f_7 & f_8 & f_9 & f_{10} & f_{11} & f_{12} & f_{13} & f_{14} \\
C_1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\
C_2 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
C_3 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\
C_4 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
C_5 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
C_6 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
C_7 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
C_8 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\
C_9 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
C_{10} & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1
\end{array}$$

Figure 3.5: An example of a codebook of error-correcting output codes for 10 classes

class label to assign to the output. Since there is a great volume of literature on this type of classifier the next section is devoted to tree-based, or hierarchical classifiers.

3.4 Hierarchical classification

Before embarking on a discussion of tree-based algorithms it is sensible to define the terminology used to describe trees and associated concepts. A tree is a specialisation of a connected graph described by a set of *nodes*, N, and *arcs*, V. An arc is a node pair. A node $n_1 \in N$ is connected to node $n_2 \in N$ if there exists a pair $\{n_1, n_2\} \in V$. A tree differs from a graph in that there is only one path from any one node to any other node (no cycles). Usually a particular node $r \in N$ is called the *root* node and all nodes connected to the root node are called *children*, the root node being their *parent*. These child nodes may have children of their own. Any node except the root node has exactly one parent, although any parent may have many children. The root node has no parents. Nodes that have no children are known as *terminal nodes* or *leaves*. Nodes that have children are called *non-terminal nodes*. A specialisation of a tree is a *binary tree* where each node may have no more than two children. A *balanced* binary tree is a binary tree where each non-terminal node has exactly two children.

An example of a balanced binary tree is given in Figure 3.6. Node A is the root, nodes A, B, C, D, E, F, and G, are non-terminal nodes, and nodes H, I, J, K, L, M,

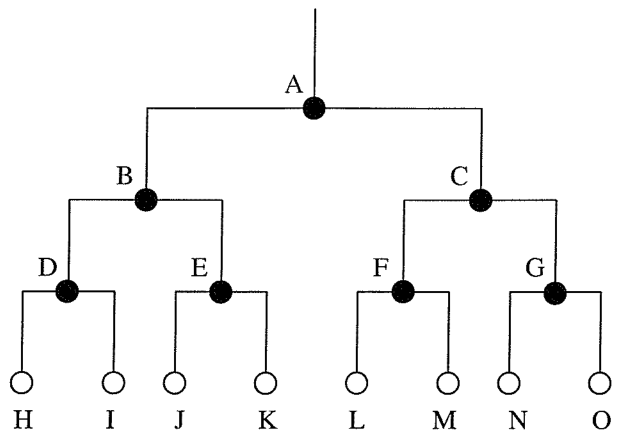


Figure 3.6: An example of a balanced binary tree.

N, and O are leaf nodes. Node A is node B's parent and nodes D and E are node B's children.

Hierarchical classification is a well established field, with early publications from the late nineteen sixties (Fu, 1968; Henrichon and Fu, 1969). It is important to review the historical development of hierarchical classification, and pattern classification in the broader sense, to be able to place the significance of this work. In fact the impetus for this work results from a reinvestigation of the hierarchical classifier with the hindsight of what is now known in machine learning.

It has been long since known by decomposing a classification task of many classes into a hierarchy of subtasks that certain advantages can be gained (Payne and Meisel, 1977; Swain and Hauska, 1977; Mui and Fu, 1980). These are:

- a reduced number of features can be used at each node,
- features are only calculated when required for classification,
- specialised features may be used at specific nodes without impacting the accuracy or efficiency of other nodes, and
- a reduced number of comparisons will be made at each node during classification.

There are also well known disadvantages (Mui and Fu, 1980; Kim and Landgrebe, 1991):

- the number of possible tree structures for a k-class problem is astronomical even for a moderately small k, (see Section 4.4) and
- the optimisation of the tree structure and the feature subsets at each node cannot feasibly be done simultaneously.

There are many ways to design a hierarchical classifier, a good review paper being (Safavian and Landgrebe, 1991). The dominating motivation behind all hierarchical classifiers is that a complex decision-making process can be broken down into many smaller decisions allowing a solution that is easier to interpret, and allowing fewer features to be used at each decision node. However it is shown in Chapter 4 that there is a conceptual difference between hierarchically partitioning the input space and hierarchically partitioning the set of classes, and although this can lead to similar looking hierarchical classifiers the distinction is important. This is a novel distinction that is not acknowledged in the literature on tree-based classifiers.

Nevertheless important design choices to be made when using a hierarchical classifier have been investigated throughout the literature which has shaped the research in this area. The most important design decisions are as follows:

- Decision rule - choice of classifier, hard or soft decisions, number of parameters
- Tree structure - binary tree, n-tree, class overlap
- Tree design - top-down, bottom-up, combinatorial search

These decisions have been explored in the literature on hierarchical classification over the last three decades and this previous work is reviewed in the following sections.

3.5 Decision rule

In any hierarchical classifier, processing is performed at each node, starting at the root and descending down the tree in such a way that a final class label can be chosen at the leaves of the tree. There are many ways that the nodes may behave, resulting in different types of hierarchical classifier with a range of advantages and disadvantages. Usually the decision made at a node dictates which of the branches from that node are followed, resulting in the most common family of hierarchical classifiers, the decision tree classifiers (DTCs). These are usually described as making hard splits, or hard decisions. However there are other ways of propagating the information from the root of the tree to the leaves, most significantly by calculating probability values for each branch in the tree and propagating the necessary values down to the leaves. This conversely is decribed as soft decision making. This is the technique adopted in this thesis, but it is important to survey the complete literature on hierarchical classifiers to highlight the distinctions and relative merits of the alternative approaches.

This section is divided into subsections that describe the prior research firstly on hard decision-making classifiers, including those that restrict each decision to axis-orthogonal splits, which leads to a popular rule-based DTC. Then soft-decision making tree classifiers are reviewed.

3.5.1 Hard decision nodes

A discussion of hard and soft decision making in tree-based classifiers is given in (Sethi, 1995). Decision tree classifiers that have hard decision nodes tend to be more interpretable in terms of the resulting rulebase, but decision trees with soft decision nodes tend to be more robust to error and exhibit better classification performance.

This section is concerned only with the effects of making a hard decision using any appropriate family of functions used to represent that decision boundary. By far the most common decision strategy is to use a simple linear decision boundary in the input space. This is often restricted to a function of only one input variable, which leads to axis-orthogonal splits, which are described in Section 3.5.2. The most common decision boundaries are described using a linear combination of the inputs, (Gama, 1997) or a quadratic decision boundary that results from a maximum likelihood classifier under the assumptions of normally distributed data (Swain and Hauska, 1977; Mui and Fu, 1980).

Early papers have investigated the performance of classifiers using such hard-decision boundaries. Swain and Hauska (1977) describe a decision tree classifier in which decisions are made between distinct subsets of classes at each node in the tree. Although the issue is not addressed explicitly, they restrict the class hierarchy such that each class may appear on only one branch of a node. Each node may also have more than two branches. The tree structure is designed either by manual inspection of the class distributions for each input feature, or by a discrete search algorithm described as 'a guided forward search with pruning'. The objective function used is a combination of classifier accuracy and computation time for that classifier. They report favourable results, with the overall accuracy reproducing the accuracy of a single stage classifier but with a significant decrease in computation time when classifying.

Similarly, in an application that involves the classification of many classes in a high-dimensional input space, Kim and Landgrebe use a hierarchical classifier to improve classification accuracy in remote sensing (Kim and Landgrebe, 1991). They show that by using a decision tree classifier and a suitable feature selection criterion that the hierarchical classifier produces higher accuracy with fewer features than a conventional single-layered classifier.

3.5.2 Axis-orthogonal splits

A well known and used algorithm for classifying patterns using a hierarchical rule-base is described by Quinlan in his book *C4.5 : Programs for Machine Learning* (Quinlan, 1993) developed fom research first introduced by Friedman (1977). A decision tree can be generated from a training set of labelled feature vectors. This can then classify unseen feature vectors as belonging to one of the possible class labels. Each node in the tree represents a rule on a single element of the feature vector. If a rule is satisfied, then a particular branch of the tree is chosen that leads either to a leaf node with a class label, or a subtree with further rules. A path from the root to a leaf is found on the basis of a given feature vector and the label attached to the leaf is assigned as the classification. This is known as hard decision making and is illustrated in Figure 3.7.

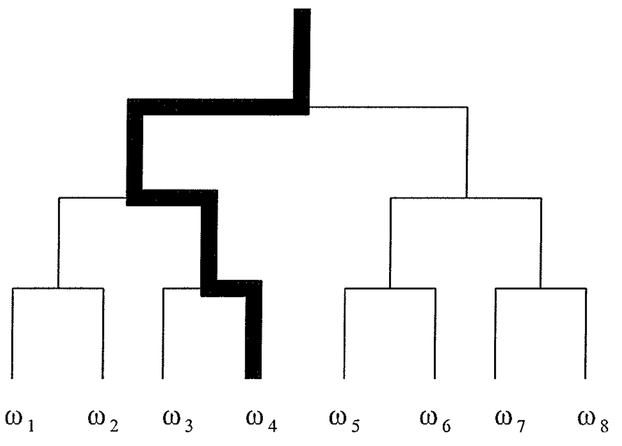


Figure 3.7: A tree classifier using hard splits to discriminate 8 classes. A single branch is chosen at each node and there is only one leaf node chosen.

Since the decisions are made on a single input variable at a time, the decision boundaries are orthogonal to the input axis corresponding to this input and parallel to all others. This is illustrated in Figure 3.8, which shows that many splits are needed to separate classes whose boundaries are oblique to the input axes.

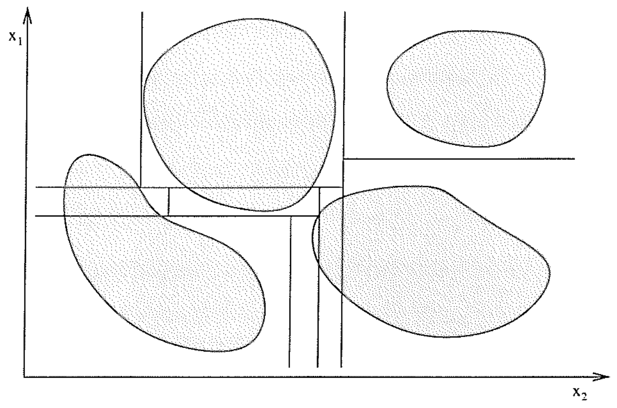


Figure 3.8: Using axis-orthogonal splits to discriminate 4 classes. Each shaded region represents the area of input space covered by a single class.

These trees are trained in a top-down fashion, whereby a rule is found that splits the input space into the most homogeneous set of subspaces using entropy-based measures, which then may be split further until a satisfactory model is achieved. Regularisation is often applied by pruning lower branches of the tree that over-fit the training data, this improves the generalization properties of the model. A human-readable rule-base can also be generated from such a model, but the model can be said to exhibit a high model bias due to the univariate nature of the rules. Another significant disadvantage with this type of classifier is that the resulting rulebase for a complex multicategory problem often has many rules which makes the rulebase hard to interpret.

There are many variations to these models that use, for example, multivariate splits at each node (Gama, 1997) or different splitting criteria. An extensive review of decision tree classifiers in general is given by Safavian and Landgrebe (1991). An example is given in Figure 3.9 of a decision tree that uses bivariate, or oblique splits. This allows the same classes to be separated using fewer decisions.

3.5.3 Soft decision nodes

As an alternative to making hard decisions at each node in the tree, where only one branch from each node is traversed, each node may output a value for each of its branches. Then these values can be combined from parent to child node as the tree is traversed, to produce a vector of resultant values for each leaf in the tree. This is illustrated in Figure 3.10. The classification decision can then be made according to

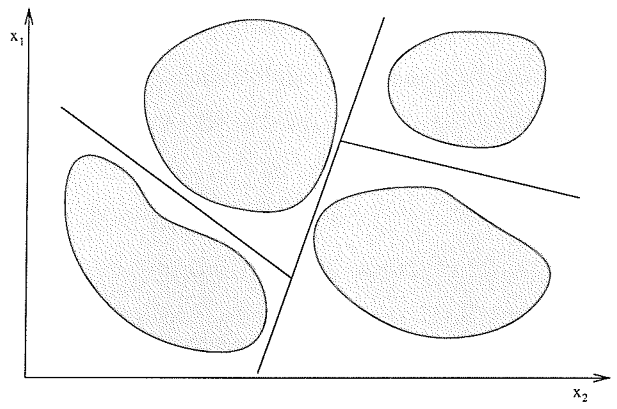


Figure 3.9: Using bivariate splits to discriminate 4 classes. Each shaded region represents the area of input space covered by a single class.

some decision rule on this vector of values. This is known as soft-splitting, or soft decision making.

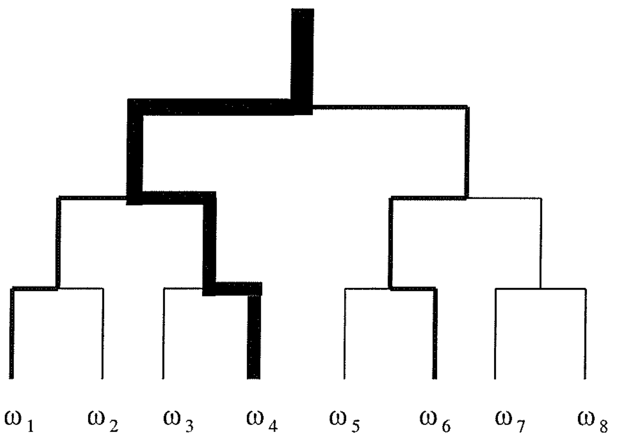


Figure 3.10: A tree classifier using soft splits to discriminate 8 classes. The probability is distributed down the tree, and the class with the highest probability is chosen.

Quinlan (1987) adapted hard decision trees to have a soft threshold about the decision boundary by using a linear ramp function. Probabilities are approximated from these soft thresholds. It is convenient to model the output values at each node as real probabilities, as then we have a rigorous framework on which to base the calculation and combination of these values. Such a probabilistic technique is presented in Schuermann and Doster (1984) and Schurmann (1996) and this technique is adopted in (Bailey and Harris, 1999). A novel formulation of this is given in Section 4.2.

Schurmann presents a probabilistic hierarchical classifier that is motivated from limitations made by model-based classifiers on finite datasets. As the number of classes increase for a single-layer classifier, the complexity of the optimum discriminant functions also increases. The approximations made on the posterior probabilities by insufficient probability models increase and consequently the final error rate increases. The problem is alleviated by breaking the many-class discrimination task into a much simpler subproblem whereby the set of classes is split into two subsets, and discrimination is performed between the two subsets. Each subset can be split similarly resulting in a hierarchy of 2-class discrimination problems. It is not necessary to always split a set of classes into two subsets, an arbitrary number is possible. However all problems can be represented by binary trees using 2-class splits.

The use of probabilistic decisions at each node does indeed have a significant impact on the design of the tree. It is shown in Section 4.4 that a probabilistic hierarchical classifier is asymptotically equivalent to a Bayes' classifier. One of the most significant differences when using this model, as shown in the analysis, is that there can be no overlap between class subsets at any decision node. This always results in a finite tree with $k-1$ nodes for a k-class problem. The search strategies for such a tree is then considerably different than for a tree with overlap which can have exponentially greater numbers of nodes.

One disadvantage (Safavian and Landgrebe, 1991) is that all the branches must now have to be traversed to compute the posteriors for each class. Schurmann (1996) suggests that a minimum threshold value can be used that will allow computation to be stopped prematurely for some branches. It is shown in Section 4.10.2 that indeed a principled method for cut-off can be used that has significant computational advantages.

3.5.4 Non-homogeneous decision nodes

In either formulation of a hierarchical classifier where there is no restriction on the chosen decision rule there is no reason to suppose that the decision rule should be the same for all nodes (Jia and Richards, 1998). In fact there is a compelling reason to choose the decision rule with the lowest affordable complexity for the task at each node. This is elaborated in Section 4.7.1.

3.6 Tree structure

Hierarchical, or tree-based classifiers all have a recursive tree structure in common. However, depending on the algorithm used there are still differences between the types of trees used. This may be in the number of branches allowed from each node, or whether the same class is allowed to belong to more than one branch from a node.

The advantages to be gained by using a tree-based classifier will only be maximised if an appropriate tree structure is found for the classification problem. It is known that an exhaustive search of all possible trees is NP-complete for the case of finite trees with no class overlap, and this search space is even larger for trees in which the same class is allowed on opposite branches of the tree. Algorithms are NP-complete if they belong to the set of problems that cannot be solved in polynomial time, the time taken to solve the problem is usually exponential in the size of the problem. There are many techniques for finding a good tree structure and these are reviewed in this section.

The binary decision tree is by far the most appropriate tree structure to use when designing a tree classifier, since a k-tree - a tree with k branches per node, can always be represented using a binary tree. Constraining the tree design to binary trees simplifies the design procedure.

3.6.1 Class overlap

It is particularly important in this thesis to acknowlegde the difference between tree-based classifiers that allow the same class label to appear on different leaves of the tree, as this is shown in Chapter 4 to be the result of a fundamental difference in the derivation of the tree-based classification algorithm. Existing research is discussed in this section regarding this property of tree-based classifiers.

Kulkarni and Kanal (1976) presented dynamic programming and branch-and-bound as methods for designing hierarchical classifiers where the class label is allowed to appear more than once in the decision tree. The optimisation problem is particularly complex and clustering techniques are used to replace many data points by prototypes to increase the speed of the optimization algorithm.

Ross Quinlan has developed much work on decision trees that allow class overlap (Quinlan, 1993). Many decision tree design issues are addressed in this book, although most of the discussion is restricted to univariate, or axis-orthogonal, splits at each node. The tree structure is generated using a top-down splitting algorithm. This is a greedy search that decomposes the input space into a number of regions using hyperplanes.

The hyperplanes are usually orthogonal to a single input axis and parallel to all others. The position of the hyperplanes is found using a metric called the gain ratio which is a measure of homogenity of class labels of the set of class points on either side of the hyperplane.

This technique usually suffers from overfitting if the tree structure is allowed to grow until the training error is zero. Pruning algorithms approach this problem by removing nodes low down in the tree until a sensible generalisation error is achieved on a separate training dataset. A quasi-probabilistic approach is given in (Quinlan, 1993) where soft decisions are made using probability values estimated from a piecewise linear approximation of the sigmoid function. However this is an approximation of probability values, in Chapter 4 a true probabilistic model is given for a similar decision tree classifier.

A clustering design procedure is presented in (Lin and Fu, 1983) where k-means clustering is used to partition the set of classes into two roughly equal subsets starting from the root to generate a binary tree with class overlap. Criteria are introduced in the design of the tree to ensure well-balanaced trees which are expected to be more computationally efficient when classifying. However the design procedure involves a considerable amount of user interaction to produce an efficient classifier. Feature selection is performed at each node in the tree using sequential forward search with either the Battacharyya or Mahalanobis distances used as performance criteria. A Bayes' classifier and k-NN classifier were used as the decision rule at each node. The k-NN classifier using k-means clustering for feature selection produced the best classification accuracy.

An efficient method for finding the tree structure based on multivariate stepwise regression is given in (Qing-Yun and Fu, 1983). Again a binary tree with multivariate linear decision nodes is used and classes are allowed to overlap branches of the tree. A comparison with the results of Lin and Fu (1983) are made with the author's results showing a slight improvement in both accuracy and computational cost for classification.

Some results from an early paper using a tree without class overlap are given in (Landeweerd et al. 1983). They have a more rigorous approach to generalisation performance, stressing the need for the data to be split into a train and test set. Their model uses Fisher's linear discriminant (Bishop, 1995) as a decision rule with hard splits. Feature selection is done using bilateral Student's t-tests and stepwise covariance analysis. The tree structure was constructed with an agglomerative design procedure using the Mahalanobis distance between class means.

However they report that the binary tree classifier does not improve on the performance on a single-level classifier. This may be due to the fact that the one-shot classifier might be using a more flexible decision boundary. The authors do nevertheless credit the tree classifier with the ability to select fewer features at each node and the reduction of computational effort when classifying.

3.6.2 Class overlap and decision complexity

It is important to note that all techniques (except for Lin and Fu (1983) who use k-NN) use either linear, or quadratic decision boundaries to discriminate between class subsets on either side of each node. These result from maximum likelihood Bayes' classifiers under assumptions of Gaussian distributions with equal, or unrestricted covariance matrices respectively (see Section 2.6). It may well be the case that the data does not adhere to these assumptions. If class overlap is allowed then more than one node can contribute to discrimination between class subsets. This effectively allows a more complex decision boundary to be defined between classes which may overcome the inflexibility of univariate linear, multivariate linear or quadratic decision boundaries. When using univariate linear models then class overlap is essential for good classification accuracy.

If no class overlap is allowed then only one decision boundary can make the distinction between two class subsets before they are further discriminated by lower branches. In this case it is important to use appropriate techniques to ensure that a flexible enough decision boundary is used at each node. This can be achieved in two ways - either by using a more flexible parametric form for the decision boundary, or by using enough features at each node such that a linear decision boundary may be used effectively. These issues are addressed in Chapter 4 which motivates the use of the hierarchical classifier used in this thesis. It is also shown in Chapter 4 that if no class overlap is used then there is a finite number of decision nodes in the hierarchical classifier and by manipulation of the hierarchical structure then the complexity of the decision boundary at any node can be controlled. Due to the fixed size of the tree for a single problem, other tree optimization techniques can be used to find the best tree structure and these are described in Chapter 5.

3.7 Related hierarchical techniques

In the previous section decision tree classifiers were discussed regarding their use as multicategory classifiers. This section describes several techniques of interest that use a hierarchical structure for modelling. Both the hierarchical mixtures of experts and the hierarchical latent variable model described below need to estimate latent, or hidden, variables that represent the mixture coefficients between the submodels in the hierarchy. This is not necessary for the hierarchical classifier in Chapter 4. Indeed a mixture model may be used for each node in the hierarchy but, as explained in Section 4.7.1, the mixture coefficients can be calculated using the training class labels.

3.7.1 Hierarchical mixtures of experts

A hierarchical mixtures of experts (HME) as presented by Jordan and Jacobs (1991) and also in (Jordan and Jacobs, 1994) is a tree-structured architecture for supervised learning. The model is composed of a mixture of generalized linear models, gated by mixture coefficients. A model can be built from a mixture of HME's, thus the hierarchical structure. The architecture effectively splits the feature space by a hierarchy of soft splits (probabilistic as opposed to deterministic). The parameters are specified using a maximum likelihood learning, specifically the Expectation-Maximization algorithm where the mixture coefficients are treated as unknown latent variables. Hong and Harris (2001) within the ISIS Research Group have extended this approach to automatic parismonious construction of hierarchies for preselected knowledge sources.

3.7.2 Hierarchical visualisation

Although not strictly a classification system, Bishop and Tipping's hierarchical latent variable model (Bishop and Tipping, 1998) has much in common with the probabilistic hierarchical classifier as described in this thesis. Using principled probabilistic analysis a projection similar to principle component analysis is used to visualise a dataset such that the axes of largest variance are mapped onto the x-y plane. This reveals the top-level structure of the dataset, which may then be split further into clusters (interactively by the user). Each cluster can then be modelled by a similar projection. The division of points to the chosen sub-models is performed by an algorithm formulated into an Expectation-Maximisation framework where each point is assigned a soft partitioning between each cluster. Clusters can be refined into further clusters resulting in an overall hierarchy of clusters that represents the dataset.

At each node the projected data can be visualised, and if that node has children then the 2-dimensional planes which represent the child clusters can be visualised as rectangles in the original parent space.

3.7.3 Unsupervised hierarchical clustering

Unsupervised hierarchical classification is found both in the statistical and machine learning literature, a good review paper from the statistical side being (Gordon, 1987), and (Ripley, 1996) covers techniques from the machine learning side. Although methods considered in this thesis use supervised learning, there is a wealth of techniques in this field that may aid this process.

Although this thesis is not attempting to imitate human thought processes in any way, it is interesting to note that there is evidence that congnitive models in the brain may be hierarchical in nature. Ambros-Ingerson, Granger and Lynch (1990) show that when simulating primary brain functions through computational neurobiology that there is evidence that the hierarchical clustering is being performed. Also motivated from a human learning perspective, Ealey (Ealey, 1997) proposes a hierarchical tree of self-organizing maps (SOMs) for language learning. Words are categorized according to features on a coarse-level map which is then segmented and each discrete category is then processed by a finer-level SOM using further features. Using a hand-coded dataset that mimics a growing child's increasing ability to perceive the world, Ealey shows that the feature maps encode an intuitive representation for each word.

3.8 Class-dependent features

The analysis of class-dependent features is a new and little studied field. Most classification algorithms strive to find a feature set that provides good discrimination for all k classes in a k-class problem.

In a 1999 paper, Oh, Lee and Suen (1999) claim that their method of class-dependent features is entirely novel. To the author's knowledge this paper is the first to use the term *class-dependent features* although other papers describe the technique of selecting specific feature sets to discriminate between subsets of classes (Jia and Richards, 1998). In fact any algorithm that breaks a k-class problem into a combination of subproblems on a subset of the k classes, which then uses separate feature extractors for each subproblem, is exploiting the fact that features can be class-dependent. However the explicit definition and analysis of class-dependent features is yet to be widely acknowledged in pattern recognition. In this thesis we explore this definition and contribute to the investigation of class-dependent features.

A review of hierarchical techniques has already been given in Section 3.4, however in this section those papers that strictly acknowledge the use of feature selection for different subsets of classes are reviewed.

In the nineteen seventies and eighties there was much research on decision tree algorithms. It was widely recognised that by making decisions on subsets of the entire feature set (Friedman, 1977; Rounds, 1980) that the feature space could be partitioned by a tree structure to separate data patterns associated with different class labels. This is motivated from a probabilistic viewpoint in Section 4.1. This is in effect recognising that specific feature subsets are better at separating some classes than others. However, the motivation was predominantly from the feature space perspective, which was being split to find regions of homogeneous class patterns. A more explicit paradigm that exploits class-dependent features considers the selection of *class* subsets and the feature subsets that can be used to discriminate those specific classes. A probabilistic motivation is given in Section 4.2.

Evidence of hierarchical systems that consider different feature sets for specific subsets of classes can be found in papers spanning the last two decades. These tend to fall into the category of *multi-stage, multi-feature* classification algorithms. These are described below.

In their design of a system for recognising printed Hebrew characters, Kushnir, Abe and Matsumoto (1983) made the observation that certain groups of Hebrew characters were very similar in their characteristics. To avoid confusion in the classification system, these character subsets were discriminated using a coarse classification method based on end-point analysis. Then for one of the subsets a geometrical feature extractor was used to discriminate between members of the subset, and for the others, specific features in a Hough transform space were selected depending on the subset to be classified. This is in principle using class-dependent features to improve performance on a many-class problem. The authors describe the technique as problem dimensionality reduction, as the discrimination tasks within each class subset required fewer features than attempting the whole problem at once.

Zhou and Pavlidis (1994) present a multi-stage process for character recognition where the problem is broken into a two-stage recognition process. Polygonal features are used in the coarse classification phase and then contour features are used to further discriminate the coarse level classifications. Again this is recognising the need for different feature extractors to be used for different stages in the recognition process. This is using the same motivation as for the use of class-dependent features in a hierarchical framework.

The above multi-level classifiers do indeed realise the benefits of considering certain features to be more expressive for certain subsets of classes, however the multi-level architectures that are proposed are all manually designed and consist of no more that 2 levels of classification. The algorithms proposed are particularly application dependent.

Using a more general approach Jia and Richards (1998) show that when using the progressive two-class decision classifier, which is a form of pairwise classifier, that the number of features selected for each node on a 4-class problem is considerably smaller when features are selected independently for each 2-class decision and the classification accuracy is greater, when compared to discriminating all four classes at once. The motivation behind this is that the simpler decision boundaries between pairs of classes are not dependent on the complexity of the most complex class pair, which is the case for single-layer multicategory classifiers. These ideas are built upon in Chapters 4 and 6 and novel analysis is presented for related algorithms.

3.9 Summary

In this chapter a review of multicategory classifiers has been presented, showing the traditional one-of-*n* approach, with criticism. Alternative approaches were reviewed including pairwise classifiers and tree-based or hierarchical approaches. There are many different approaches to designing a hierarchical classifier and the most important of these design decisions have been discussed, such as the form of the decision rule, the form of the tree - notably whether classes are allowed to appear more than once in the tree (allowing class overlap), and the procedure used to design the tree. The prior research presented in this chapter shows that many of the issues of multicategory classification have been investigated and many different algorithms have been proposed over the last few decades but what is lacking is a common framework in which to place these diverse algorithms.

This thesis addresses this problem via a probabilistic formulation that provides a strong framework on which multicategory classifier design can be based. This is presented in the next chapter and provides a consistent paradigm from which many multicategory classification algorithms can be designed. The conceptual difference between allowing class overlap in a hierarchical classifier is brought out clearly, and most notably equivalences between the one-of-*n*, pairwise and hierarchical classifiers is shown under certain assumptions. This then allows these algorithms to be investigated in terms of their complexity as the number of classes increases.

In addition, a review of class-dependent features was presented in this chapter, and although many of the papers outlined do not actually state the use or analysis of class-dependent features, they have been shown to use the same motivation that leads to the definition and use of this term. This is further explored in Chapter 6 using the novel interpretation of conditional independence and experimental results in Chapter 7 show that the use of class-dependent features is advantageous for several classifiers that break the k-class problem into a set of binary problems.

Chapter 4

Multicategory classification

In this chapter a new taxonomy of multicategory classification is offered, based on the formulation of the posterior probabilities and the method of decomposing a complex problem into manageable subproblems.

There are many ways to approach a multicategory classification problem. Two principle approaches exist depending on whether the input space is to be decomposed into managable subregions, or whether the set of classes is decomposed into managable class subsets. This distinction is seldom made and this thesis is concerned with the explicit study of multicategory classification, and more specifically the decomposition of the set of classes, Ω^{all}.

In Chapter 2, the conditioning on the set of classes for a classification problem was made explicit so that the multicategory classification task can be considered as a decomposition of the set of classes. For completeness, one should also condition the posteriors on the region of the input space over which the classification task is valid. For a real-valued feature space of dimension d, this is usually $\mathbb{R}^d$ and since it is constant, it is often omitted from the formulation of the posterior probabilities. However, when the conditioning is made explicit by writing $P(\omega_i|\mathbf{x}, R, \Omega^{\text{all}})$, where R is the set of regions over the input space, one can reason more clearly about classifier design.

Again the problem is to estimate the posterior distribution, then to apply a decision rule. The maximum a posteriori (MAP) decision rule is the only one considered here, which states that once $P(\omega_i|\mathbf{x}, R, \Omega^{all})$ has been found for all i, simply choose the ω_i with the highest posterior. This has the advantage that certain posteriors with low probability need not be computed, however if a different decision rule, or anything other than the 0/1 loss matrix is used then all the posteriors need to be evaluated. This issue is touched on again in Section 4.10.2.

In the next section it is shown that a decomposition of the input space leads to popular classifiers for multicategory classification, and in the subsequent sections it is shown that the set of classes can be decomposed, which leads to other popular multi-category classifier paradigms. These paradigms are then compared and it is shown that the most popular method today is not the best choice in many aspects.

4.1 Decomposing the input space

One method of determining the posterior probabilities from the training sample is to use a local estimate of the posteriors by dividing the input space into regions, and use the observation vector $\mathbf{x}$ to determine which local region is of interest. Then use a simple frequency estimate of the posterior using the labelled training points in that region. In this section, when decomposing the input space the set of classes Ω^{all} is constant and only appears on the right-hand side of the conditioning bar so it may be dropped for brevity.

The input space $\mathbb{R}^p$ is partitioned into a set of regions $R = \{r_q; q = 1, \ldots, m\}$ such that each r_q is distinct:

$$\bigcap_{q=1}^{m} r_q = \emptyset, \tag{4.1}$$

and all r_q form a complete partition of R:

$$\bigcup_{q=1}^{m} r_q = R. \tag{4.2}$$

We can now write

$$\begin{aligned} P(\omega_i|\mathbf{x}, R) &= \sum_{q=1}^{m} P(\omega_i, r_q|\mathbf{x}) && (4.3) \\ &= \sum_{q=1}^{m} P(\omega_i|r_q, \mathbf{x}) P(r_q|\mathbf{x}). && (4.4) \end{aligned}$$

However, $P(\omega_i|r_q, \mathbf{x})$ is not any easier to estimate than the original $P(\omega_i, r_q|\mathbf{x})$ because it still depends on the exact value of $\mathbf{x}$, but it can be sensibly approximated by $P(\omega_i|r_q)$ giving

$$P(\omega_i|\mathbf{x},R) = \sum_{q=1}^{m} P(\omega_i|r_q)P(r_q|\mathbf{x}). \tag{4.5}$$

This is a sensible approximation since as the number of regions increases and their volume decreases, the approximation approaches the exact value for an infinite training set. However, for a finite training set, there will be a point at which the generalisation error is maximised, and increasing the number of regions will thereafter cause overfitting. This is the motivation for pruning in decision tree classifiers.

Since all r_q are defined to be disjoint then $P(r_q|\mathbf{x})$ will be unity only for the region in which $\mathbf{x}$ lies. This is written

$$P(r_q|\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in r_q \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

In this case we need only evaluate one term of the summation in Equation 4.5. This effectively states that via a decomposition of R into the disjoint set of regions $\{r_q, q = 1,\ldots,m\}$ we can approximate the global posterior $P(\omega_i|\mathbf{x},R)$ by a local posterior $P(\omega_i|r_q)$ given that the query point $\mathbf{x}$ lies within region r_q. This is how many popular multicategory classification algorithms work and they are differentiated by the method used to decompose R.

A k-nearest neighbour (kNN) classifier can be said to estimate a local posterior for each class within a region specified about the query point. The region and its complement form a disjoint partition of the input space, but the region's complement need never be considered by virtue of Equation 4.6. The region r is specified by finding the hypersphere in R centered on $\mathbf{x}$ with a radius such that exactly k points from the training sample lie within the hypersphere. The value of k is chosen using an appropriate estimate of the generalisation peformance such as cross-validation. The local posteriors are estimated within this region by $P(\omega_i|r) = n_i/k$ where n_i is the number of points for class ω_i within the region. The class with the highest posterior is chosen.

A decision tree classifier also operates on a decomposition of the input space but in this case each region r_q in the initial partition is decomposed further. In Equation 4.4, the value $P(\omega_i|\mathbf{x}, r_q)$ was approximated, however an alternative method is to form a further partition of r_q. Each region r_q is then partitioned into a set of regions $r_q = \{r_{qp}; p = 1,\ldots,m_q\}$ such that each r_q is again distinct, $\cap_{q=1}^{m_q} r_q = \emptyset$ and all r_q form a compact partition, $\cup_{p=1}^{m_q} r_{qp} = r_q$. Again we can write

$$P(\omega_i|\mathbf{x}, r_q) \quad = \quad \sum_{p=1}^{m_q} P(\omega_i|r_{qp}, \mathbf{x})P(r_{qp}|\mathbf{x}). \tag{4.7}$$

For a continuous space R, each region may be partitioned further until a final approximation is made as in Equation 4.5. The relationships between the regions and their partitions can be represented as a tree diagram as in Figure 3.10. A classifier of this type is known as a decision tree classifier (DTC) using soft decision nodes and a great deal of research has gone into DTCs. A good review paper is (Safavian and Landgrebe, 1991).

It is widely acknowledged that one advantage of this hierarchical decomposition is that the partitioning of a region need not be dependent on every element of the input vector $\mathbf{x}$. In other words, the boundaries between partitions may be parallel to one or more of the axes of the input space. It is often the case that a severe restriction is made which requires that the boundaries are orthogonal to one axis and parallel to all others. The nature of the partitions is shown in Figure 3.8 for a two dimensional space. This type of decision tree has the advantage that the model can be written as a set of rules of one variable. This is said to be interpretable by the user because they can understand the reasons behind the final classification output by reading the rules that contributed to that decision. However the restriction of axis-orthogonal splits can firstly lead to an unmanagable number of rules for a complex problem, and secondly can impose too high a model bias and render the models outputs inaccurate.

The important issue here is that the input space is being partitioned, and the posterior probabilities for each class is estimated for a defined region of this space. These posteriors can be calculated by a simple frequentist approach, given the training points that fall in that region. It is often the case in such a restricted region the posteriors for many classes are zero and a decision is made between only a few classes. In fact the algorithms presented in (Quinlan, 1993) strive to find a tree-based partition such that all the training points within that region belong to the same class. This is a result of partitioning and not through direct manipulation of the set of classes. In the next section the alternative approach is shown whereby the set of classes is decomposed to solve the multicategory classification problem.

4.2 Decomposing the set of classes

Returning to the original problem of estimating $P(\omega_i|\mathbf{x}, R, \Omega^{\text{all}})$; we can follow one of several approaches that, instead of decomposing the input space into regions, uses the principle of decomposing the set of classes, Ω^{all}, into subsets and then estimating the posteriors via these decompositions. By far the most straightforward approach is to estimate the posteriors for each class directly, and this can be interpreted as a set of k models, one for each class, where one class, ω_i, is discriminated against the remaining classes, $\Omega_i^{\text{others}} = \{\omega_j; \forall j \neq i\}$. The ith model is parameterised to discriminate between ω_i and Ω_i^{others}, and outputs the posterior probability for ω_i. Again the class with the highest posterior probability can be chosen.

This method was adopted in the early days of artificial neural networks as the generalisation of a 2-class problem to a k-class problem when using perceptrons, which have since been shown to be equivalent to linear discriminants. This approach was upheld as multi-layer perceptrons were developed, and still holds for the standard formulation of a multi-layer perceptron today. In this context, the method of discriminating one class from n classes is called the *one-of-n* decomposition. However other methods have existed for some time, such as the hierarchical decomposition (not to be confused with decision tree classifiers as mentioned in the previous section), and pairwise discrimination. However, the relationships between these techniques have been largely ignored. A consistent framework using statistical pattern classification is presented here that draws these techniques together.

These fall into three categories:

- One-of-n - separate into k problems of one class against the rest.
- Hierarchical - separate into $k-1$ recursive binary partitions.
- Pairwise - separate into $k(k-1)/2$ problems between all pairs of classes.

One of the most important issues here is that in each case a k-class problem can be reduced to a set of 2-class problems. Many classification algorithms are specifically designed to solve 2-class problems, and these may be used effectively. Principled approaches to the decomposition of the many class problem into binary decisions and the recombination of models are given for each case in the sections following.

Considering the decomposition of the input space, presented in Section 4.1 and the decomposition of the set of classes presented in this section, a novel taxonomy of multicategory classifiers is presented. This is shown in Figure 4.1.

4.3 One-of-n partitioning

If one were then to use a set of binary classifiers to distinguish each class in a k-class problem, then the most common choice would be to train a set of k classifiers such that

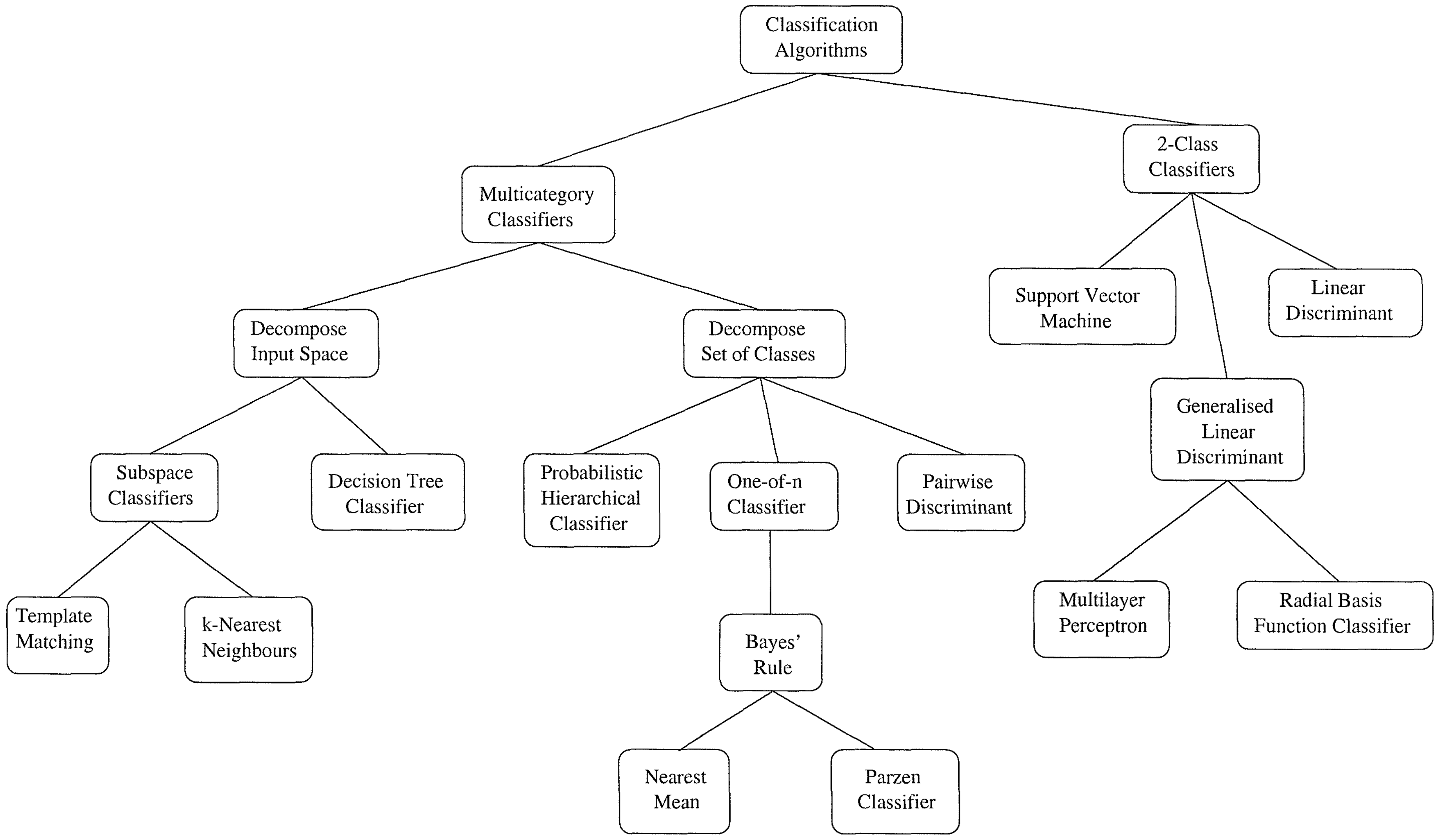


Figure 4.1: Classifier taxonomy

each one discriminates one class from the remaining set of classes.

In this case the subsets Ω_i^l and Ω_i^r, $i = 1, \ldots, k$, for the k models are defined as follows:

$$\begin{aligned} \Omega_i^l &= \{\omega_i\}, && (4.8) \\ \Omega_i^r &= \{\omega_j; \forall j \neq i\}. && (4.9) \end{aligned}$$

This can be thought of as distributing the probability value of $P(\Omega^{\text{all}})$, which is always unity, between the individual class elements, $P(\omega_i|\mathbf{x})$, according to the observation, $\mathbf{x}$. Since all classes are treated similarly this is represented by a 'flat' structure as in Figure 4.2.

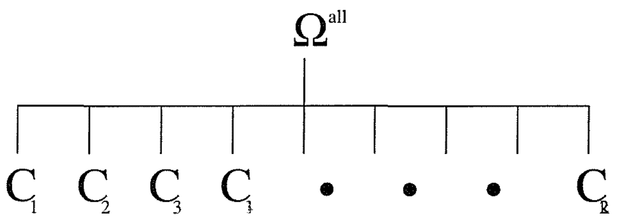


Figure 4.2: Using a single node to discriminate k classes

It must be stressed that there are two approaches to the one-of-n decomposition, firstly whether a separate model is used for each of the k problems, or secondly, whether a single model is used to map the input space to a new 'hidden' feature space and then k model outputs are used to solve the one-of-n problem. This is the same as training k MLPs with one output, or training one MLP with k outputs respectively It might be quicker to train a single MLP, but it will be argued in Chapter 6 that different features should be used for each different model, and by the same argument, different hidden feature spaces should be used for each model. This advocates the use of k different models with one output each.

4.4 Hierarchical partitioning

Hierarchical partitioning is achieved by splitting Ω^{all} into subsets and then splitting these subsets until no further splitting is possible. This is described in this section.

This set of classes is partitioned (for the moment arbitrarily) into s subsets, $\Omega_q^{\text{sub}}, q = 1, \ldots, s$. The partition is disjoint,

$$\bigcap_{q=1}^{s} \Omega_q^{\text{sub}} = \emptyset, \tag{4.10}$$

and complete,

$$\bigcup_{q=1}^{s} \Omega_q^{\text{sub}} = \Omega^{\text{all}}. \tag{4.11}$$

Once again we can write

$$\begin{aligned} P(\omega_i|\mathbf{x},\Omega^{all}) &= \sum_{q=1}^{s} P(\omega_i,\Omega_q^{\text{sub}}|\mathbf{x}) & (4.12)\\ &= \sum_{q=1}^{s} P(\omega_i|\Omega_q^{\text{sub}},\mathbf{x})P(\Omega_q^{\text{sub}}|\mathbf{x}) & (4.13)\end{aligned}$$

and since the class subsets are non-overlapping

$$P(\omega_i|\Omega_q^{\text{sub}},\mathbf{x}) = \begin{cases} 1 & \text{if } \omega_i \in \Omega_q^{\text{sub}} \\ 0 & \text{otherwise.} \end{cases} \tag{4.14}$$

This allows us to reduce the summation to a single term:

$$P(\omega_i|\mathbf{x},\Omega^{all}) = P(\omega_i|\Omega_q^{\text{sub}},\mathbf{x})P(\Omega_q^{\text{sub}}|\mathbf{x}), \text{ where } \omega_i \in \Omega_q^{\text{sub}}. \tag{4.15}$$

If we set $m = 2$ then $P(\Omega_q^{\text{sub}}|\mathbf{x})$ can be represented by a binary classifier. Then, if Ω_q^{sub} has only two class members then $P(\omega_i|\Omega_q^{\text{sub}},\mathbf{x})$ can be modelled by a binary classifier, otherwise it can be further partitioned until all problems can be represented by binary classifiers. This follows much the same argument as the tree-based classifier described in Section 4.1, except no approximations have been made, and the depth of the tree is finite and depends on the number of classes. In fact it is straightforward to show that the number of binary models needed is always $k-1$.

In our formulation, no suggestion has been made how to model the posterior probabilities, and the partitioning of classes into subsets was chosen arbitrarily. In fact if the probabilities can be modelled perfectly then then the choice of the partitioning into a class hierarchy is arbitrary, since no approximations were made in the above analysis. In this case the resulting posterior probabilities are equivalent to

k	N_k
2	1
3	3
4	15
5	105
6	945
7	10,395
8	135,135
9	2,027,025
10	34,459,425
11	654,729,075
12	13,749,310,575

Table 4.1: Number of possible trees for problems of k classes

those generated using the one-of-n partitioning. However, in the more realistic case of imperfect models and finite data then the choice of the class hierarchy will have some effect on the evaluation of the final posterior probabilities. The next section considers the complexity of choosing an appropriate class hierarchy. Chapter 5 deals with this structure identification problem in detail.

Although in the case of infinite data the class hierarchy is arbitrary, when presented with a real problem the choice of class hierarchy plays an important role in the accuracy of the hierarchical decomposition. The number of permutations of the hierarchy can be computed by a simple product of the odd integers.

The simplest tree has two nodes. There are then three ways of adding a new node to this tree, either adding it on the left branch, to the right branch, or by creating a new root node with the new class on one side and the existing tree on the other. It can be shown that for a general tree of k classes, that there are $2k-1$ ways of adding a new node. The number of possible trees for a problem of k classes is then:

$$N_k = \prod_{n=1}^{k-1} 2n-1.$$

The number of permutations for $k = 2,\ldots,12$ are shown in table 4.1.

Obviously for problems of many classes this number grows very quickly. Asides from an exhaustive search there are no general methods of finding the optimal tree.

Methods of finding near-optimal trees in the space of all trees are discussed in Chapter 5.

4.5 Pairwise discrimination

The pairwise discriminant classifier works on the simple principle that each class should be compared to each other class separately and the class that has the most favourable comparisons is chosen as the output class. Many versions of this algorithm (Friedman, 1996; Jia and Richards, 1998; Kressel, 1999) use hard classifiers that have only binary outputs and the choice of combination scheme varies. Here we use probabilistic outputs.

In this case the subsets Ω_{ij} for the $k(k-1)/2$ class subsets are defined as follows:

$$\Omega_{ij} = \{\omega_i, \omega_j\} \tag{4.16}$$

where i and j are chosen such that all class pairs (except where $i = j$) are considered. Only half of the $k(k-1)$ models need to be trained since $P(\omega_i|\Omega_{ij},\mathbf{x}) = 1 - P(\omega_i|\Omega_{ji},\mathbf{x})$.

	ω_1	ω_2	ω_3	ω_4
ω_1	n/a	Ω_{12}	Ω_{13}	Ω_{14}
ω_2	Ω_{21}	n/a	Ω_{23}	Ω_{24}
ω_3	Ω_{31}	Ω_{32}	n/a	Ω_{34}
ω_4	Ω_{41}	Ω_{42}	Ω_{43}	n/a

Table 4.2: Pairwise sets of classes for a four class problem

The pairwise classifier does not lend itself to the same analysis as the hierarchical decomposition because the class subsets no longer form a disjoint partition due to the fact that there is overlap between the subsets.

Nevertheless we can write

$$P(\omega_i|\mathbf{x},\Omega^{all}) = P(\omega_i|\mathbf{x},\Omega_{ij})P(\Omega_{ij}|\mathbf{x},\Omega^{all}). \tag{4.17}$$

This can be averaged over all the class pairing between class ω_i and the other classes as,

$$P(\omega_i|\mathbf{x},\Omega^{all}) = \frac{1}{k-1}\sum_{j\neq i} P(\omega_i|\mathbf{x},\Omega_{ij})P(\Omega_{ij}|\mathbf{x},\Omega^{all}) \quad (4.18)$$

Remarkably in practice a very accurate classifier can be implemented by simply using the average of the posteriors given by each model for all pairings of a class, given by

$$P(\omega_i|\mathbf{x},\Omega^{all}) = \frac{1}{k-1}\sum_{j\neq i} P(\omega_i|\mathbf{x},\Omega_{ij}) \quad (4.19)$$

It is shown here that such a classifier results in the optimum decision boundaries for neighbouring classes. This explains the exceptional results shown by the pairwise classifier in practice, though at the cost of training a number of models in the order of k^2.

If we ignore the fact that a classifier should output a posterior probability value and write the output of the pairwise classifier simply as a function of $\mathbf{x}$ for each class ω_i as

$$f_i(\mathbf{x}) = \frac{1}{k-1}\sum_{j\neq i} P(\omega_i|\mathbf{x},\Omega_{ij}), \quad (4.20)$$

then the decision boundary between classes ω_1 and ω_2 is the locus of points where

$$f_1(\mathbf{x}) = f_2(\mathbf{x}). \quad (4.21)$$

Since $P(\omega_i,\Omega_{ij}|\mathbf{x}) = P(\omega_i|\mathbf{x})$ and $P(\Omega_{ij}|\mathbf{x}) = P(\omega_i|\mathbf{x}) + P(\omega_j|\mathbf{x})$, it is true that

$$\begin{aligned} P(\omega_i|\mathbf{x},\Omega_{ij}) &= \frac{P(\omega_i,\Omega_{ij}|\mathbf{x})}{P(\Omega_{ij}|\mathbf{x})} \\ &= \frac{P(\omega_i|\mathbf{x})}{P(\omega_i|\mathbf{x})+P(\omega_j|\mathbf{x})} \end{aligned} \quad (4.22)$$

This allows us to write 4.21 as

$$P(\omega_1|\mathbf{x})\sum_{p\neq 1}\frac{1}{P(\omega_1|\mathbf{x})+P(\omega_p|\mathbf{x})} = P(\omega_2|\mathbf{x})\sum_{q\neq 2}\frac{1}{P(\omega_2|\mathbf{x})+P(\omega_q|\mathbf{x})}. \quad (4.23)$$

This is true when $P(\omega_1|\mathbf{x}) = P(\omega_2|\mathbf{x})$, which of course describes the true decision boundary between the two classes when using the MAP decision rule. This shows that the model given in Equation 4.20 is equivalent to the optimal model, given that the individual class posteriors $P(\omega_i|\mathbf{x})$ are modelled correctly.

4.6 Comparison of techniques

It has been shown that many class problems can be decomposed via different means into binary class problems where individual models can be learnt and their output combined in a principled manner. However it is important to consider the nature of those binary class problems to be able to select suitable classifier models at each node. It is of course quite acceptable to select different types of model for any of the binary classifications, but for the moment the assumption is made that the models are homogeneous across all partitions.

It is shown below that the hierarchical and pairwise decompositions can linearly separate any problem that a one-of-*n* model can linearly separate. Then in the next section, using a more intuitive analysis, the separation properties of each decomposition is considered. This also illustrates the effective decision boundaries constructed by each global model.

It is useful to define linear separability formally to investigate how each of the three class decompositions can separate a set of *k* classes.

Two sets of classes Ω^l and Ω^r are linearly separable if a vector $\mathbf{w}$ and a bias b can be found such that:

$$\mathbf{w}^T\mathbf{x}_i > b,\ \forall \mathbf{x}_i \in \Omega^l \tag{4.24}$$

$$\mathbf{w}^T\mathbf{x}_i \leq b,\ \forall \mathbf{x}_i \in \Omega^r. \tag{4.25}$$

For a single node classifier using linear discriminants to achieve perfect classification, each class must be linearly separable from the rest. We can define a function S such that:

$$S(\Omega^l,\Omega^r) = \begin{cases} 1 & \text{if } \Omega^l \text{ and } \Omega^r \text{ are linearly separable} \\ 0 & \text{otherwise.} \end{cases}$$

Then a one-of-*n* classifier can only achieve perfect classification using linear discriminants if a class ω_i and the set of remaining classes $\Omega_i^{\text{others}} = \{\omega_j, \forall\omega_j \neq \omega_i\}$, are linearly separable for all classes:

$$S(\omega_i, \Omega_i^{\text{others}}) = 1,\ \forall\omega_i \in \Omega^{all}. \tag{4.26}$$

It is straightforward to show that if Equation 4.26 holds for a single-node classifier then the hierarchical classifier can also separate the data. Simply split Ω^{all} into two subsets, Ω^l and Ω^r such that:

$$\Omega^l = \omega_i \quad \text{and} \tag{4.27}$$
$$\Omega^r = \{\omega_j, \forall \omega_j \neq \omega_i\}. \tag{4.28}$$

The subset Ω^r can then be split further by virtue of Equation 4.26, leading to a full hierarchical decomposition.

However the converse is not true, a set of classes which can be linearly separated by a hierarchy cannot always be separated by a single-node classifier. This is evident from Figure 4.3. The classes ω_1 to ω_4 can be linearly separated for each class and the remaining classes. However, if the class ω_5 is introduced then this is no longer the case. A hierarchical classifier can still separate these classes linearly as shown.

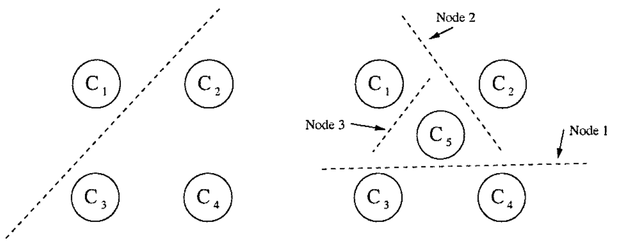


Figure 4.3: Example of linear separability where the one-of-n model fails. Left: Four linearly separable classes, with a single one-of-n decision boundary, Right: Five classes separated by a hierarchical decomposition, the decision boundary for each node is labelled.

This also follows for the pairwise decomposition which by a similar argument can also linearly solve any problem that the one-of-n decomposition can solve using linear decisions. Again, if Equation 4.26 holds, then ω_i must be linearly separable from each individual $\omega_j, j \neq i$.

It must then be the case that simply by using a hierarchical decomposition, we can linearly separate a greater number of datasets than separating them using a one-of-n decomposition. This is investigated further in the next section. In fact for classes of one point it can be shown that any set of classes can be separated by the hierarchical decomposition of points.

4.7 Linear separability for many classes

It is important to analyse how linear separability is affected as the number of classes increases for both the one-of-n output encoding and the hierarchical decomposition.

Analysis of the pairwise classifier is considered in Section 4.7.1 since it is difficult here to illustrate the individual decision boundaries for each of the $k(k-1)/2$ classifiers. Firstly the simple problem where each class is represented by a single point is examined since this avoids problems of class overlap and uncertainty. In fact any two classes of one point are linear separable, unless they lie on the same point, in which case they may aswell be the same class. It will be shown that in the linear case the hierarchical decomposition is by far the more flexible model in that it can represent all problems, whereas the one-of-n encoding can only represent a small subset of problems.

In the next section this will be developed further for the family of effective non-linear models known as generalised linear discriminants.

When considering classes of only one point, the problem of linear separability can be investigated solely in terms of the number of classes without being distracted by the linear separability issues that arise from the overlap of classes with many points. The following argument is illustrated in Figure 4.4.

For each row in Figure 4.4 there are three diagrams that illustrate, from left to right, possible problems which can be separated using a one-of-n decomposition, then problems that cannot be separated in such a way, and finally the same problem as solved by a hierarchical decomposition. Decision boundaries are drawn as solid lines, and for illustrative purposes the hierarchical decision boundaries for nodes closer to the root of the tree are drawn as longer lines.

Each row represents a problem with more classes. As the number of classes increases it can be seen that the one-of-n decomposition can only solve problems where each of the classes forms part of the convex hull of all classes. If a new class falls inside the convex hull (row 3), or is added such that another class falls inside the convex hull (row 4) then the problem cannot be solved by a linear one-of-n output encoding. This is a severe restriction and it is unlikely that in spaces of low dimension that all classes will form part of the convex hull. Thus it is only likely that a linear discriminant will only be effective for many classes in a high-dimensional space since the convex hull property is more likely if the dimensionality of the input space is increased.

However all problems of this type can be solved using a hierarchical decomposition such that the set of classes is recursively partitioned into two subsets without necessarily increasing the dimensionality.

The limitations of the one-of-n decomposition are a result of the restriction of linear separability in low dimensions. It is shown below that this can be overcome by simply transforming the space non-linearly into a feature space of higher dimension. To overcome the linear one-of-n problem in the previous section it is sufficient to define

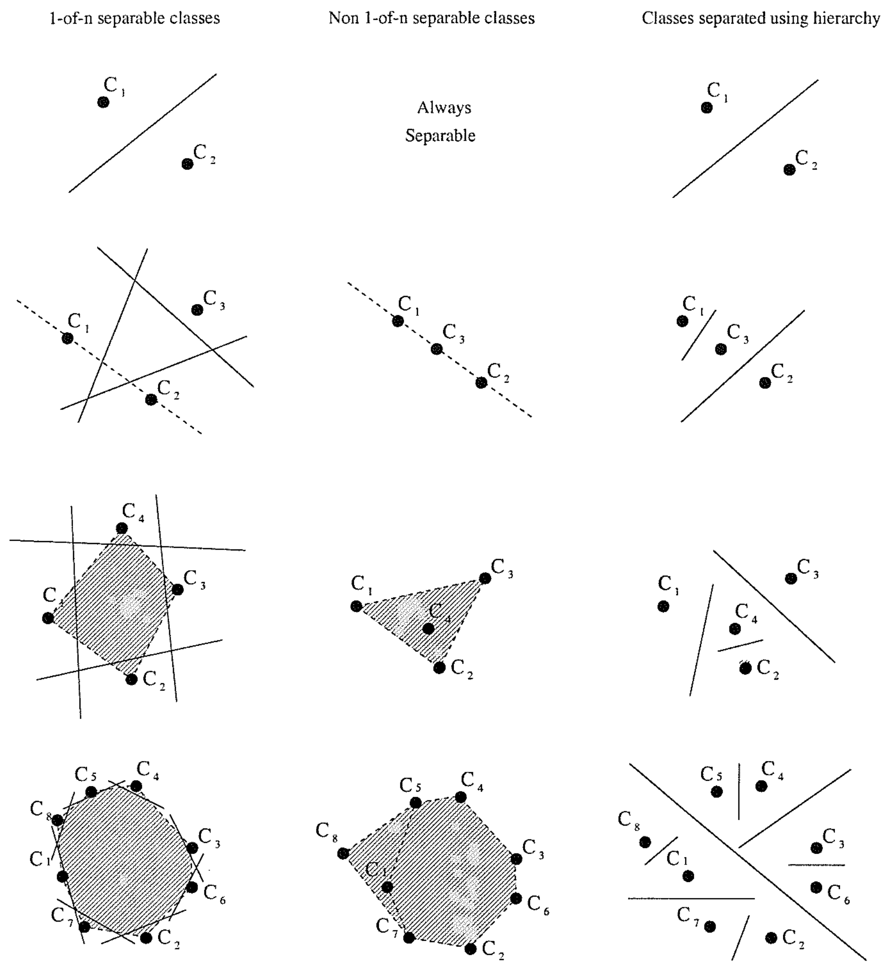

1-of-n separable classes
Non 1-of-n separable classes
Classes separated using hierarchy
Always
Separable
C1
C2
C3
C4
C5
C6
C7
C8

Figure 4.4: Separating classes of one point using one-of-n and hierarchical linear models

a set of basis functions of the form:

$$z_i = d(\mathbf{x} - \mathbf{c}_i) \text{ for } i = 1, \ldots, k, \tag{4.29}$$

where k is the number of classes, $\mathbf{x}$ and $\mathbf{c}_i$ are respectively the input point and the single class points in the original input space, and $d(.)$ is a monotonically increasing distance function. An example of a typical such basis function is given by:

$$z_i = e^{-|\mathbf{x}-\mathbf{c}_i|^2} \text{ for } i = 1, \ldots, k, \tag{4.30}$$

which is known as a radial basis function.

This transforms the problem into a space with k dimensions that guarantees that each point in z-space forms part of the convex hull so the one-of-n output encoding can be used to separate the classes. (This can be seen by the fact that each z_i will be equal to its maximum value of 1 only for its respective class ω_i. Thus each class must lie on the convex hull in k-dimensional space).

The disadvantage of this approach is that learning now has to be done in a much higher dimensional space, which will increase as the number of classes increases. This is not the case for the hierarchical model which has the advantage that the problems learnt at each node are only as complex as necessary, in this case linear models can be used in the original input space, providing simpler and more transparent models.

4.7.1 Classes of more than one point

Once the number of points in each class increases the problem becomes more complex as there is the possibility that the points lie in such a way that any two classes are not linearly separable. It may even be the case the the spread of class points overlap for two of more classes in such a way that no model can separate them and there is genuine ambiguity between the classes. In this case a probabilistic approach is an appropriate form of uncertainty where posterior probabilites for each class are output before using a decision rule to decide which class to choose. Probability density functions can be used to describe each class and if these density functions are assumed to be Gaussian and of equal covariance for all classes, then the analysis is equivalent to the single-point classes, where the single point is the mean of the distribution.

To illustrate the behaviour of the algorithms using linear and non-linear models, the decision boundaries resulting from each class decomposition are shown in Figures 4.5 and 4.6 for increasing numbers of classes. It is plainly shown that when using linear discriminant classifiers that the resultant multi-class decision boundaries degenerate for the one-of-n decomposition, where the hierarchical and pairwise decompositions cope adequately, although each one does impose its own 'flavour' to the nature of the decision boundaries.

It is interesting to study the decision boundaries resulting from each decomposition using a Bayes' classifier under the assumption of Gaussian distributed classes. This is shown using the same dataset as previously in Figure 4.6. (The data was generated using a procedure detailed in Section 7.2.1.) In this case the assumptions for the classifiers used in the one-of-n and pairwise decompositions are not being broken, and this is seen by the similarity between the resulting decision boundaries. The hierarchical decision boundaries are however different, and this can be readily

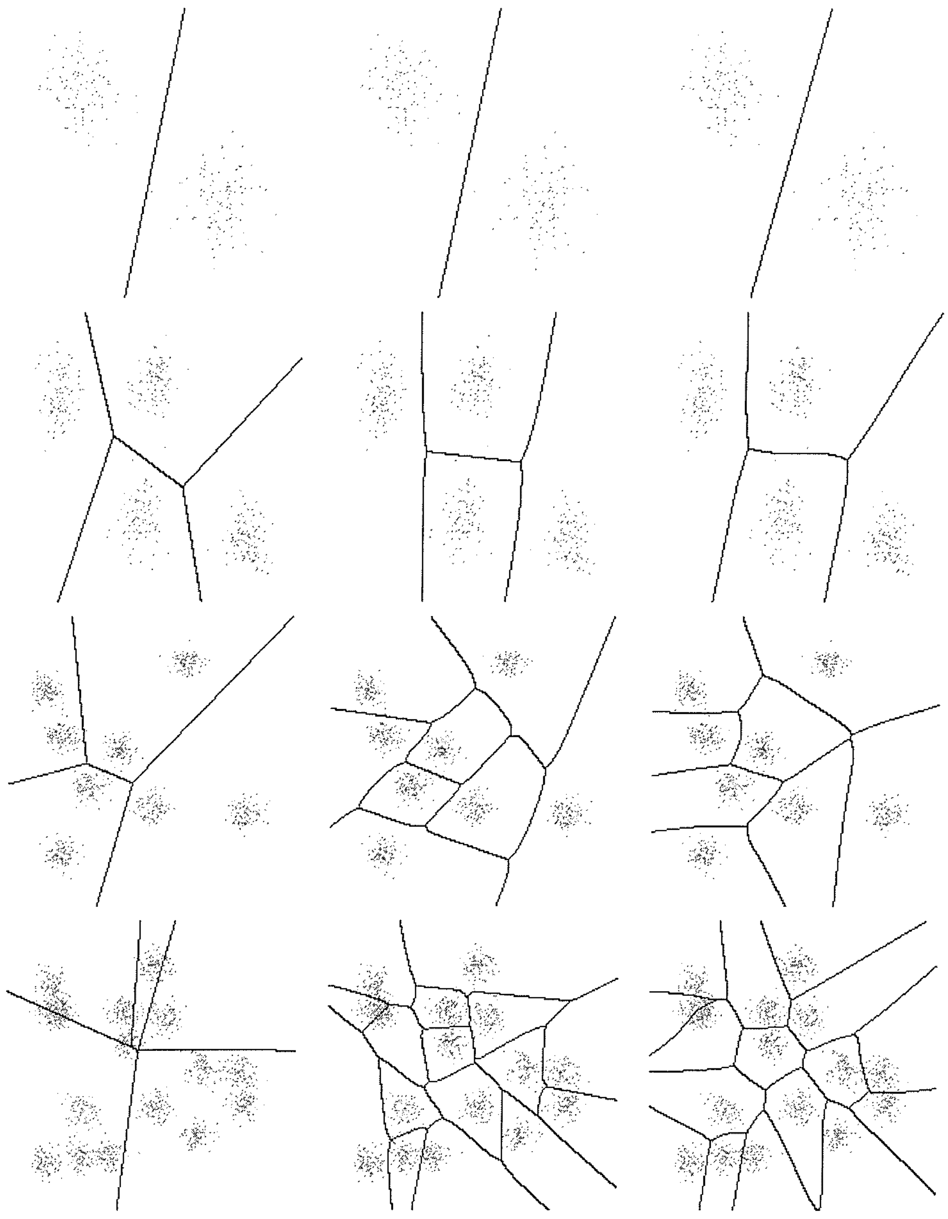


Figure 4.5: Decision boundaries using linear nodes

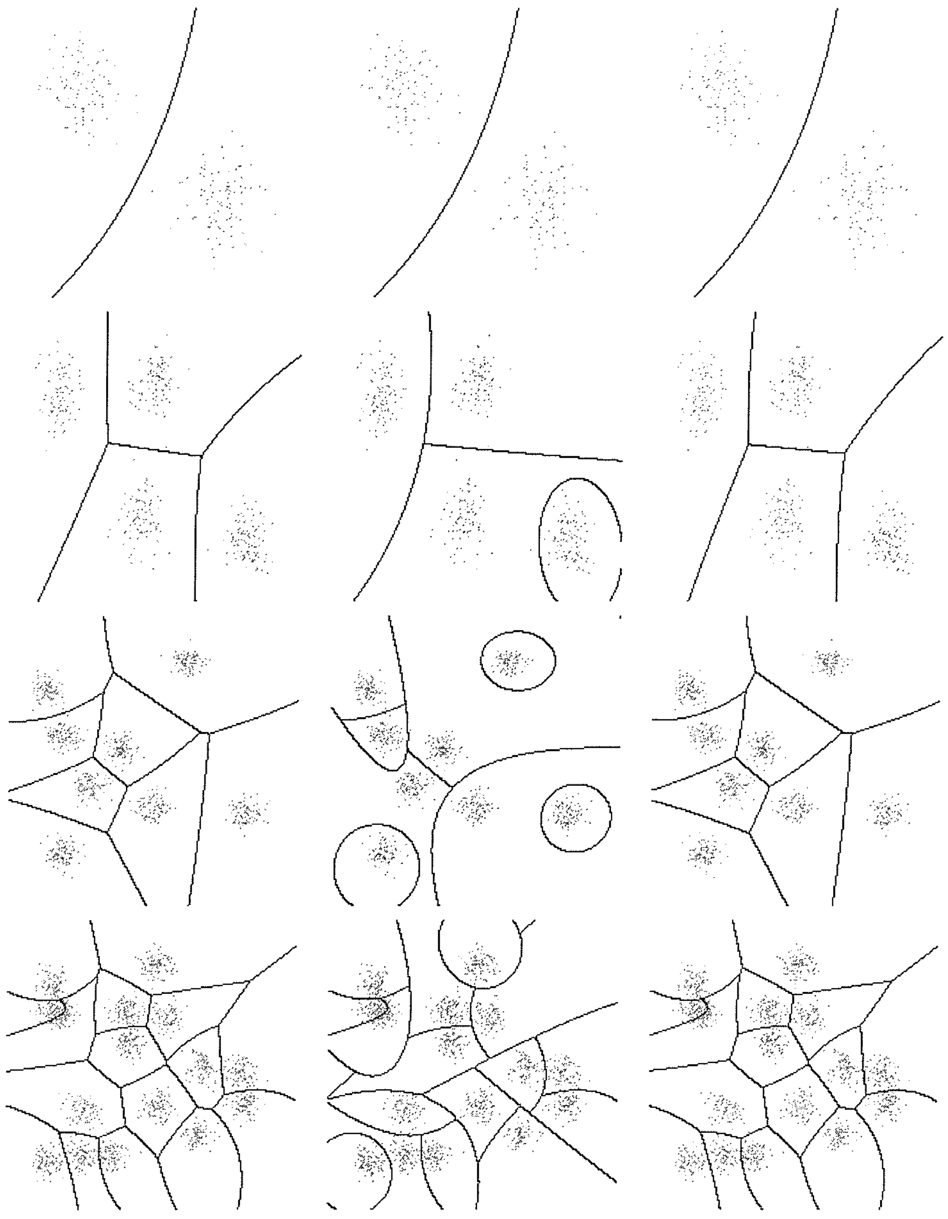


Figure 4.6: Decision boundaries using Gaussian nodes

explained.

The hierarchical classifier using Gaussian Bayes' classifiers will attempt to estimate the mean and covariance of the left and right class subsets, Ω_l and Ω_r via Equations 2.7 and 2.8. When Ω_l and Ω_r contain only one class each, then this results in the optimum classifier for the two classes, since each class has been generated from an unknown Gaussian distribution. However, for classifiers in the hierarchical decomposition closer to the root, there will be multiple classes in Ω_l and Ω_r. In this case the assumption of Guassian distributed class points is being broken, since the class points are most accurately modelled by multimodal Gaussian distributions, otherwise known as a Gaussian mixture model.

In this case, the optimum class-conditional density estimate $P(\Omega_l|\mathbf{x},\Omega_q)$ can be calculated directly. Usually when estimating the parameters for a mixture model, the mixture coefficients are unknown. But here the mixture is known since we are estimating the density of a set of classes, where each class point is labelled. The mixture density can be written

$$\begin{aligned} P(\Omega_l|\mathbf{x},\Omega_q) &= \sum_{\omega_i\in\Omega_l} P(\omega_i|\mathbf{x},\Omega_q) && (4.31) \\ &= \sum_{\omega_i\in\Omega_l} \frac{P(\mathbf{x}|\omega_i,\Omega_q)P(\omega_i|\Omega_q)}{P(\mathbf{x}|\Omega_q)}. && (4.32) \end{aligned}$$

In this case the $P(\omega_i|\Omega_q)$ are the mixture coefficients and these are readily calculable as

$$P(\omega_i|\Omega_q) = \frac{P(\omega_i)}{\sum_{\omega_j\in\Omega_q} P(\omega_j)}. \tag{4.33}$$

If the classifiers used to generate the decision boundaries for the hierarchical decomposition were mixture density classifiers using Equation 4.32 then the hierarchical classifier would be equivalent to the one-of-n classifier and show the same decision boundaries.

4.7.2 Summary

It must be stressed that only a subset of problems really are linearly separable, and in practice it is seldom the case. More flexible models are needed, and these will be considered in the next section, however the need for more flexible models should be properly understood before selecting an appropriate model for the problem.

There are two basic needs for more flexible models in classification:

- To overcome the non-linearly separable nature of problems between two classes.
- To overcome the deficiencies of the one-of-n decomposition.

It has been shown that when there is only one point per class (i.e. perfectly linear separable models and no ambiguity) that the decomposition of the set of classes is important. Hierarchical and pairwise methods are by far more suitable for many-class problems than the traditional one-of-n decomposition. One might argue that using a more flexible model such as one of the family of non-linear discriminants can overcome this problem. The author suggests that one should use more flexible models to overcome the problems of non linear separability between pair of classes and that the hierarchical or pairwise decomposition should be used to solve the many-class problem.

4.8 Generalised linear discriminants

The transformation of the input space via a set of basis functions given in Equation 4.30 is an example of a generalised linear discriminant, in this case specifically a radial basis function classifier. Generalised linear discriminants are an important family of models and it is important to investigate the effects of using these models in each class decomposition paradigm.

Generalised linear discriminants are so called because they all use a linear discriminant as a final output layer. This output layer acts on a space which is generated by the action of a non-linear transform of the inputs. The different types of generalised linear discriminants result from using different non-linear transforms. In this section, the issues resulting from multicategory classification using multi-layer perceptrons are shown, a similar argument follows for other type of generalised linear discriminants such as radial basis function classifiers.

A multi-layer perceptron (MLP) usually consists of a input layer, a hidden layer, and an output layer. The hidden layer transforms the inputs into a 'hidden' feature space via ridge functions (of the same form as a linear discriminant) and then performs the final output using a linear discriminant on the hidden space. (The choice of activation function may differ depending on the specific type of MLP used).

A simple analysis of how a MLP behaves for multicategory classification can be done by analysing simple two-class and three-class problems. The two-class problem is shown in Figure 4.7. The diagram shows the outlines of two classes (all class points lie within the outline) and the two ridge functions needed to separate the two classes. In this section the notation C_i is used to label the ith class, to avoid confusion between ω_i previously used for classes and w_{ij} used for weights.

This example is intended to illustrate the fact that a single MLP with many outputs is over parameterising a many class problem. The same many class problem can be solved by a hierarchy of many MLP models each with a single output. A non-linearly separable problem between two classes is shown to be solved by a MLP with two hidden nodes (see Figure 4.7). Two hidden nodes are needed due to the non-linear nature of the decision boundary between the classes. This is not related to the number of classes; were the classes to be linearly separable then a single hidden node would suffice, and were the decision boundary between them more complex then more hidden nodes would be needed.

When this problem is extended to three classes, (see Figure 4.8), a single extra hidden node can be added to solve the problem because the new class is linearly separable from the the other two classes. Again this is not related to the number of classes, but to the complexity of the decision boundaries between the classes.

Now in the three class case, there is a choice between a single three-output MLP and a hierarchical combination of two single-output MLP models. It will be shown that the hierarchy is the simpler model.

The MLP is defined by the action of the hidden nodes and the output node. The hidden nodes are given by:

$$\phi_1^h(\mathbf{x}) = \frac{1}{1+e^{-(\mathbf{w}_1^h\mathbf{x}+b_1^h)}}, \tag{4.34}$$

$$\phi_2^h(\mathbf{x}) = \frac{1}{1+e^{-(\mathbf{w}_2^h\mathbf{x}+b_2^h)}}, \tag{4.35}$$

where $\mathbf{w}_i^h$ is the weight vector and b_i^h is the bias term for the ith hidden node. We shall assume that the weights of the hidden nodes are adjusted such that they describe the ridge functions shown in Figure 4.7. It is then sufficient to determine values of the hidden to output weights that can discriminate the two classes in the new 'hidden' feature space.

The action of the output node is given by:

$$y_1 = g(a_1) \tag{4.36}$$

$$a_1 = w_{11}^o\phi_1^h(\mathbf{x}) + w_{12}^o\phi_2^h(\mathbf{x}) + b_1^o, \tag{4.37}$$

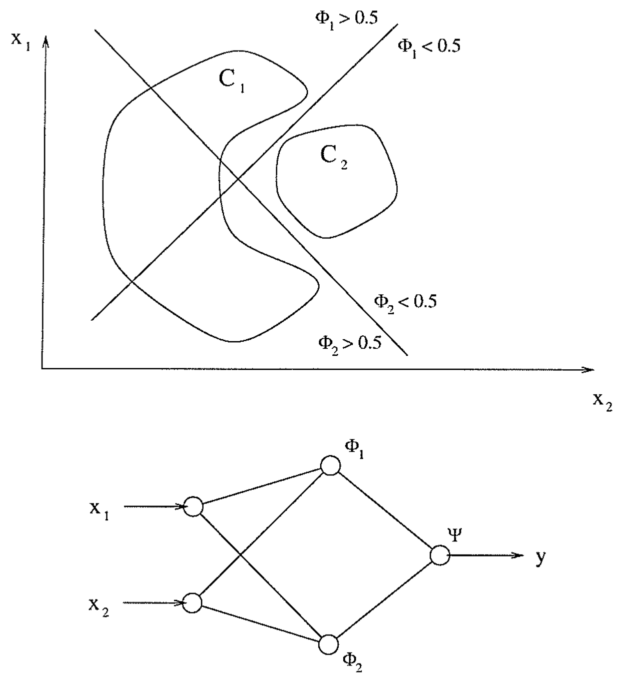


Figure 4.7: Separating two classes using a multi-layer perceptron

where $g(a) = \frac{1}{1+e^{-a}}$, w^o_{ij} is the jth element of the ith output weight vector and b^o_i is the ith output bias term. (Here there is only one output vector). The output can be interpreted as the posterior probability of class C_1, $P(C_1|\mathbf{x}) = y$ and $P(C_2|\mathbf{x}) = 1-y$, if the MLP is trained using the mean-squared error function (Bishop, 1995).

The three class problem is shown in Figure 4.8. An extra ridge function is needed to separate the new class in input space and an extra two outputs are needed to compute the one-of-n output encoding for more than two classes.

The extra hidden node which defines the new ridge function is given as:

$$\phi^h_3(\mathbf{x}) = \frac{1}{1+e^{-(\mathbf{w}^h_3\mathbf{x}+b^h_3)}}, \tag{4.38}$$

and the outputs are now formulated as:

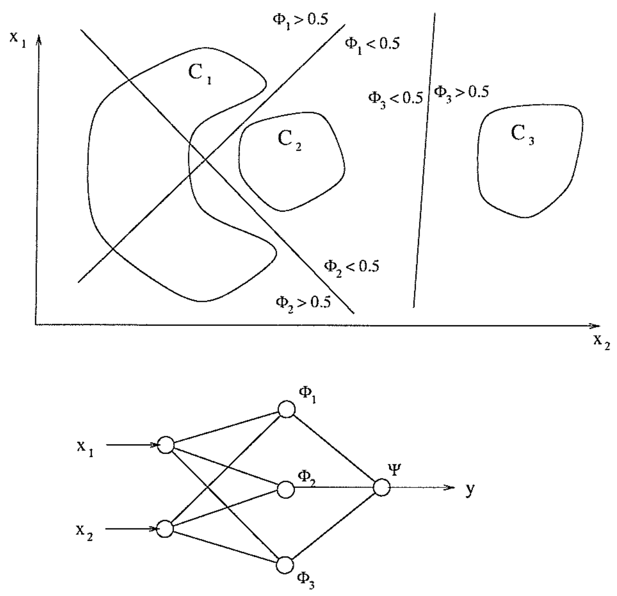


Figure 4.8: Separating three classes using a multi-layer perceptron

$$
\begin{aligned}
y_1 &= g(a_1) && (4.39)\\
y_2 &= g(a_2) && (4.40)\\
y_3 &= g(a_3) && (4.41)\\
a_1 &= w^o_{11}\phi^h_1(\mathbf{x}) + w^o_{12}\phi^h_2(\mathbf{x}) + w^o_{13}\phi^h_3(\mathbf{x}) + b^o_1, && (4.42)\\
a_2 &= w^o_{21}\phi^h_1(\mathbf{x}) + w^o_{22}\phi^h_2(\mathbf{x}) + w^o_{23}\phi^h_3(\mathbf{x}) + b^o_2, && (4.43)\\
a_3 &= w^o_{31}\phi^h_1(\mathbf{x}) + w^o_{32}\phi^h_2(\mathbf{x}) + w^o_{33}\phi^h_3(\mathbf{x}) + b^o_3, && (4.44)
\end{aligned}
$$

As stated in Section 4.7.2, the need for non-linear models results fundamentally from the complexity of the decision boundaries between neighbouring classes in input space and the one-of-*n* decomposition unnecessary complicates the nature of the decision boundaries in each of the *k* models needed to solve a *k*-class problem. This is shown in the examples given in Figures 4.7 and 4.8. It can be seen that classes C_1 and C_2 need two hidden nodes to be separated, but to separate C_3 from either C_1 or C_2 only one hidden node is needed. Therefore the maximum required complexity of the between-class decision boundaries is 2 hidden nodes. The one-of-*n* MLP shown in Figure 4.8 needs a model of 3 hidden nodes to solve the problem, whereas the hierarchical MLP shown in figure 4.9 uses models no greater than the maximum required complexity of 2 hidden nodes. In this case the hierarchical model reduces

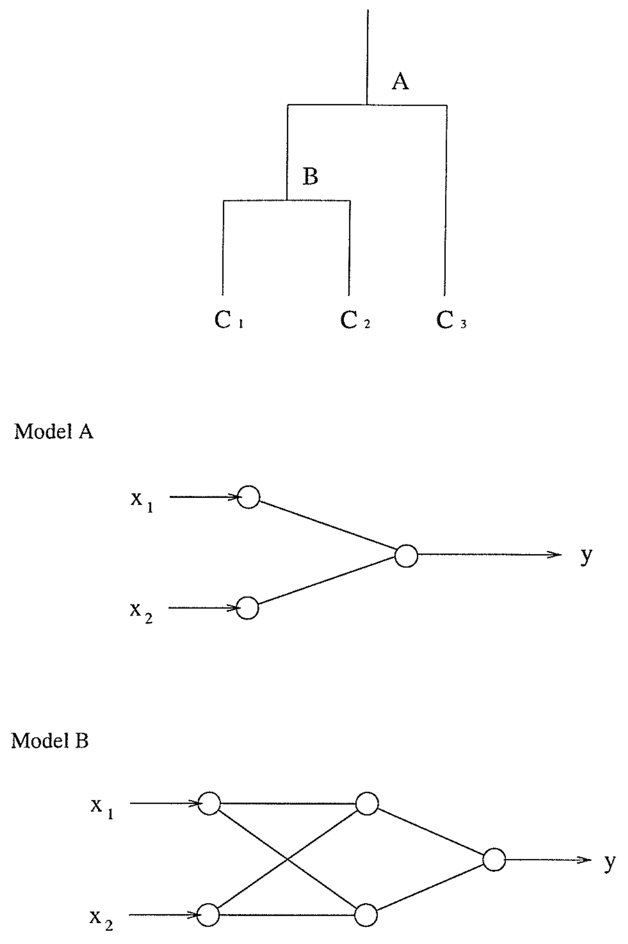


Figure 4.9: Separating three classes using a hierarchical classifier with a linear discriminant node and a multi-layer perceptron node.

the complexity of each model used. For problems of k classes, the complexity of the decision boundary is expected to be of the order of k for the one-of-n decomposition, where it is expected to be constant for the hierarchical and pairwise decompositions.

This is illustrated in Figures 4.10 to 4.13. Figure 4.10 shows the decision boundaries using MLPs for each 2-class problem in the one-of-n, hierarchical, and pairwise decompositions. Since the data is generated from Gaussian distributions of equal covariance it is known that the optimum decision boundaries between any two classes is of linear complexity. The number of hidden nodes for each MLP is set to 1, and this is equivalent to a linear model. The one-of-n model shows poor decision boundaries since it requires a greater model complexity for each model in the decomposition. The hierarchical and pairwise models show well-formed decision boundaries. For Figures

4.11, 4.12, and 4.13, the number of hidden nodes is increased, and it can be seen that the one-of-n decomposition gradually improves the decision boundaries, while the hierarchical and pairwise remain relatively constant, showing that the decision boundaries for linear models need little improvement.

4.9 VC-dimension analysis

A fundamental concept in learning theory is the VC-dimension. An upper bound on generalisation performance can be formulated using the VC-dimension for a particular family of models. The generalisation of a model can be represented by the difference between the output of the model trained on a finite sample of N data points, and the output of the hypothetical model that represents the true distributions of the data. The theorem due to Vapnik and Chervonenkis (Vapnik, 1998) gives an upper bound on the probability of this difference exceeding a given value ε.

This, stated for the set of classification functions with VC dimension h, is:

$$\Pr(\max_y |P(\mathbf{w}) - \nu(\mathbf{w})| > \varepsilon) < \left(\frac{2eN}{h}\right)^h \exp(-\varepsilon^2 N) \qquad (4.45)$$

where $P(\mathbf{w})$ is the probability of classification error, or risk functional, and $\nu(\mathbf{w})$ is the training error on a finite dataset, or empirical risk functional (Haykin, 1999). The vector $\mathbf{w}$ is the parameter vector of the model, e is the base of the natural logarithm and N is the number of training points.

It can also be shown that, for a linear discriminant decision rule, i.e.

$$\phi(\mathbf{w}^T\mathbf{x}+b) \qquad (4.46)$$

$$\text{where} \quad \phi(v) = \begin{cases} 1, & v \geq 0 \\ 0, & v < 0, \end{cases} \qquad (4.47)$$

the VC-dimension, h, is given by:

$$h = m+1, \qquad (4.48)$$

where m is the dimension of the input space of the discriminant, (i.e. $\mathbf{x} \in \mathbb{R}^m$).

The important issue behind Equation 4.45 is that the tightness of the bound determines how useful the relationship is. If for a certain family of models the bound

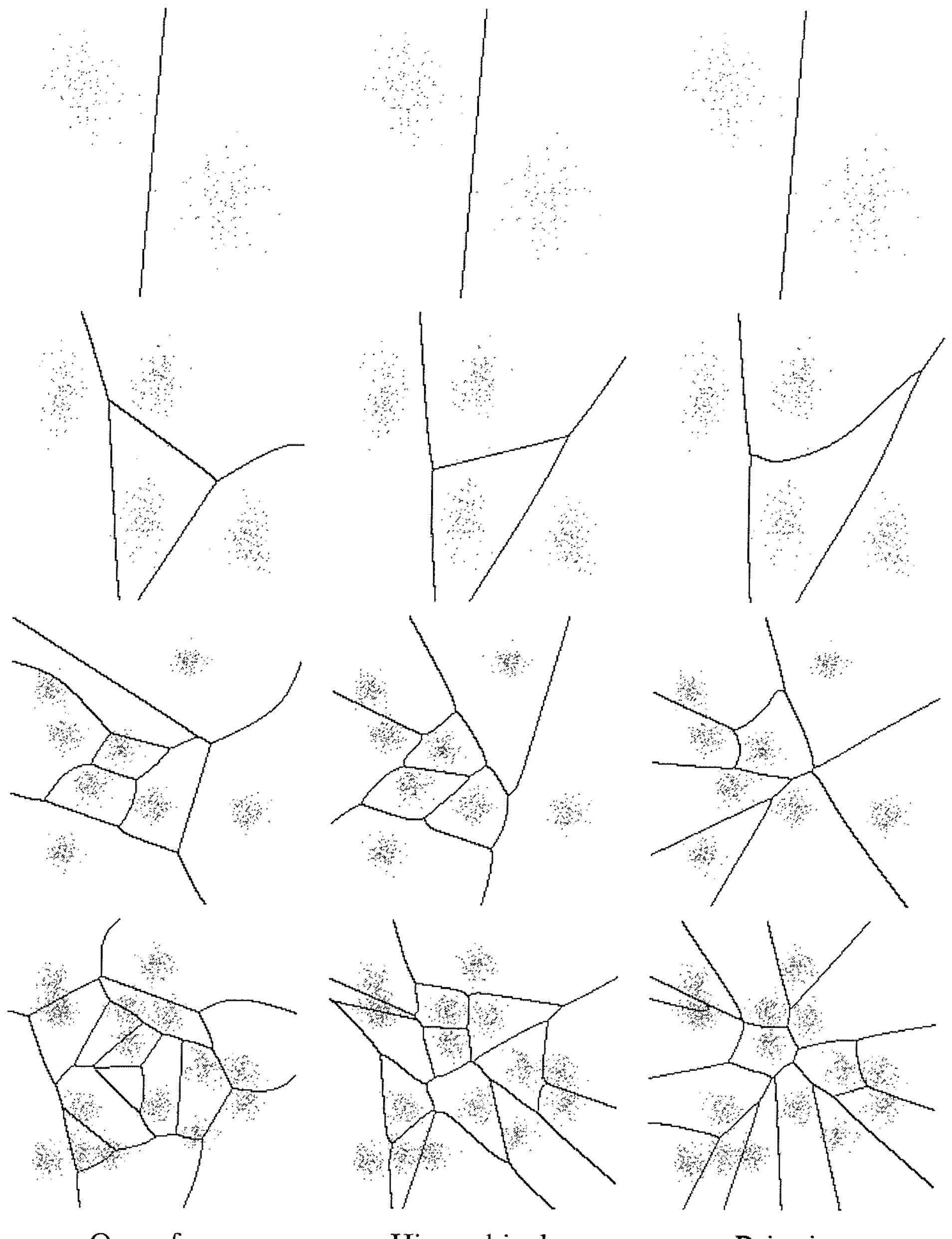


Figure 4.10: Decision boundaries using MLP nodes with 1 hidden node and sigmoid output activations

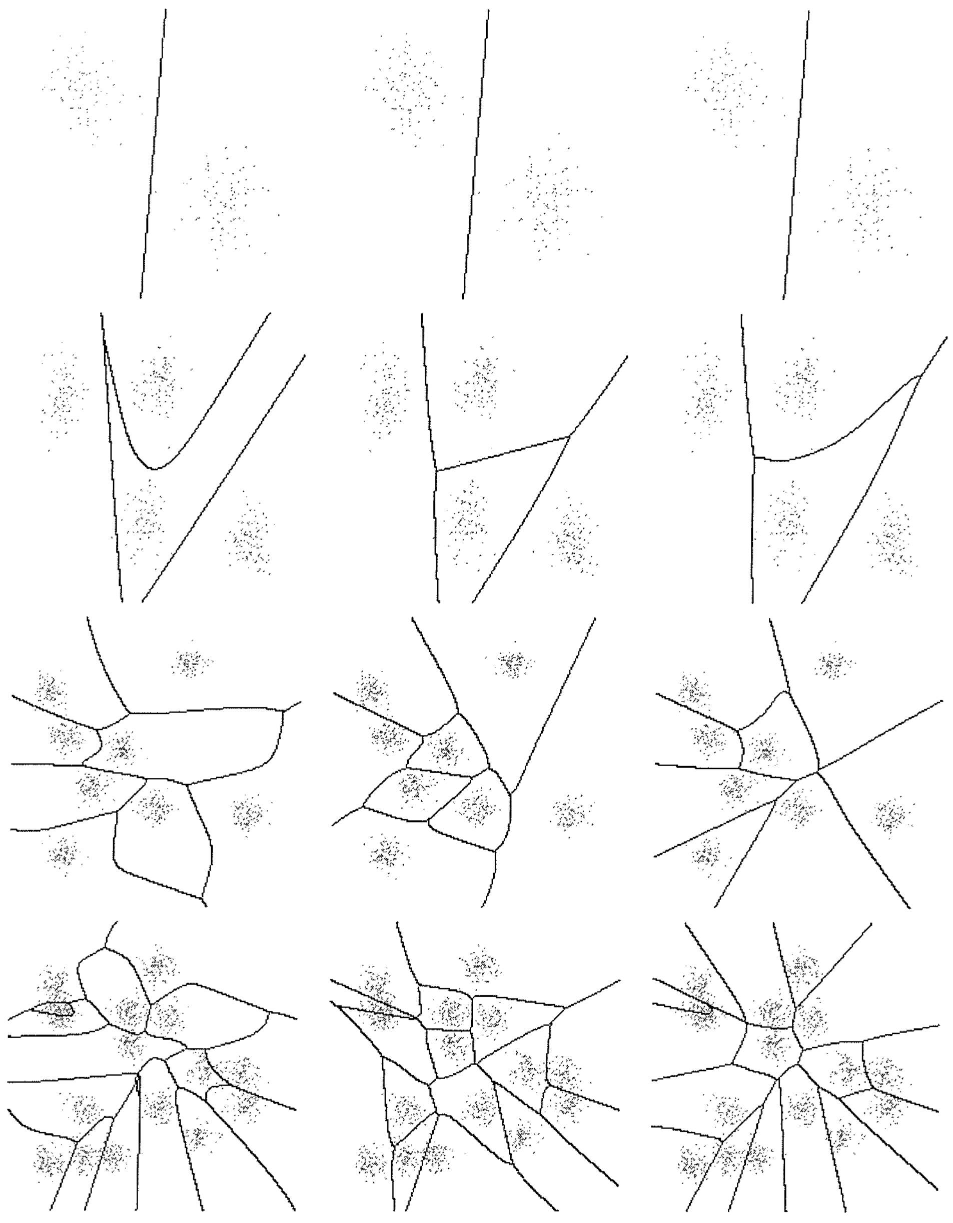


Figure 4.11: Decision boundaries using MLP nodes with 2 hidden nodes and sigmoid output activations

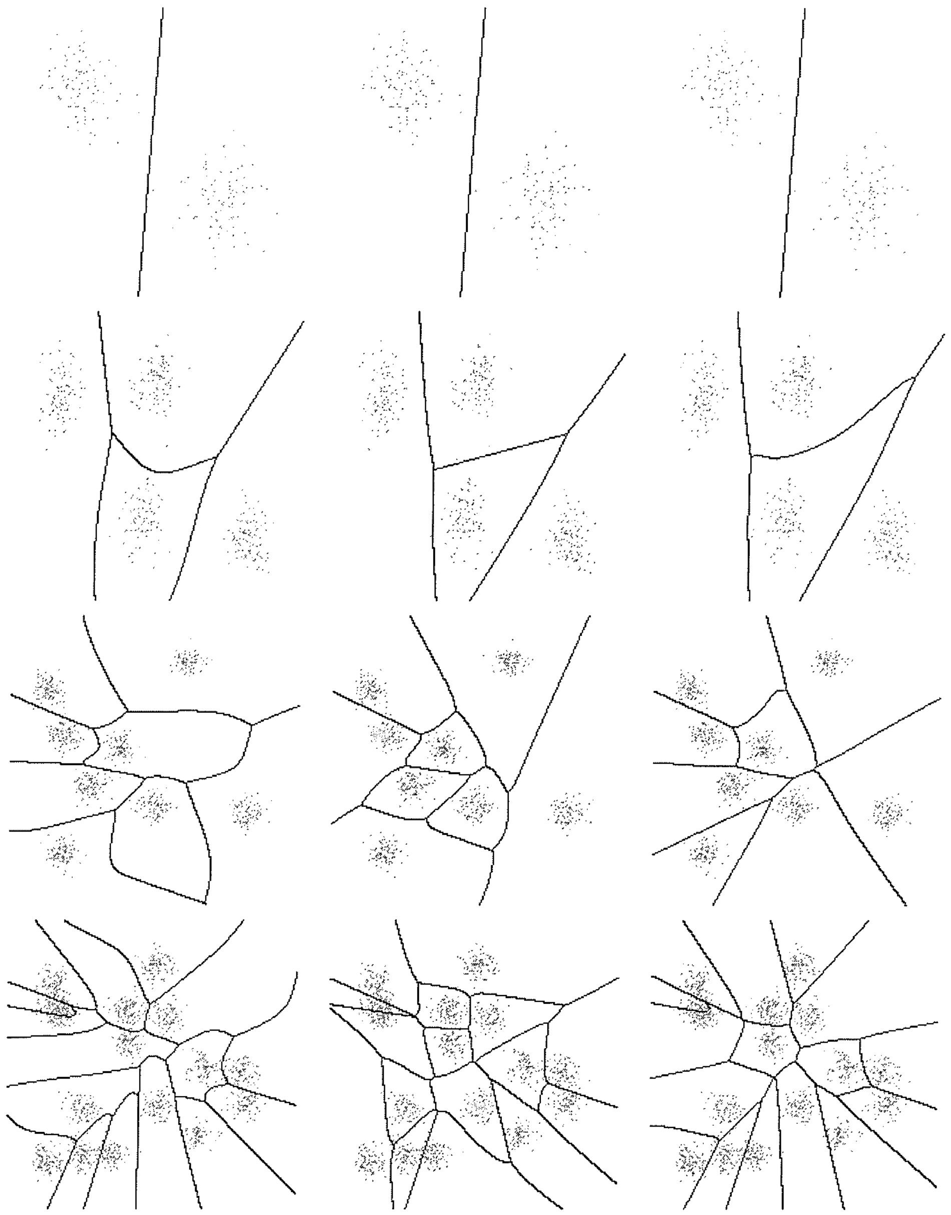


Figure 4.12: Decision boundaries using MLP nodes with 3 hidden nodes and sigmoid output activations

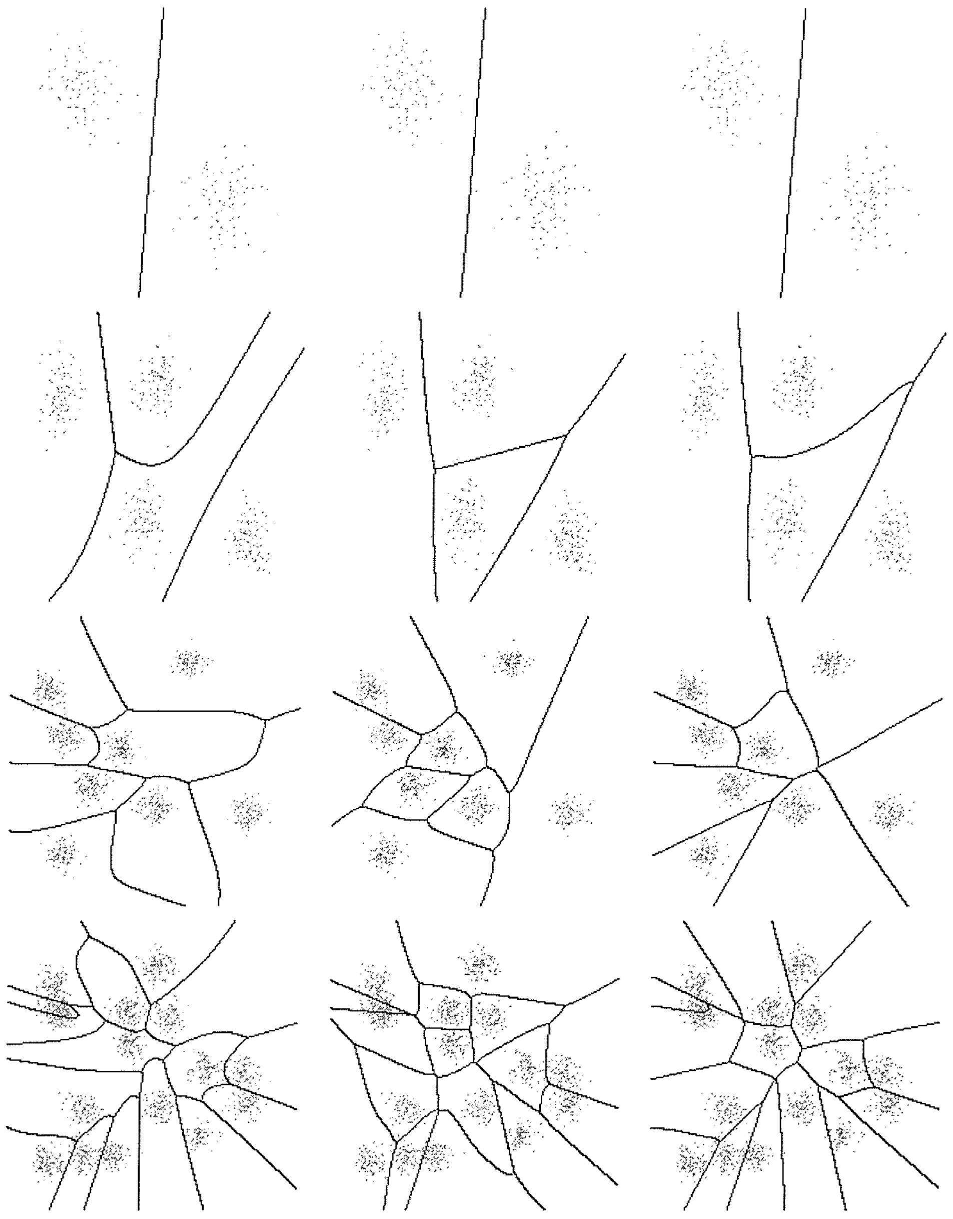


Figure 4.13: Decision boundaries using MLP nodes with 4 hidden nodes and sigmoid output activations

can be proven to be tight then it is a representative upper limit for the generalisation performance of that family of models. In the case where the bound is not tight for whole family of models then it could be said that the bound serves no purpose for selecting models within that family. However, the VC-dimension still proves to be a good guide for the generalisation performance for many models in statistical learning theory. In fact for the case of non-zero empirical risk the aim is to choose the model with the minimal the VC-dimension (Burges, 1998).

It has already been shown that by using a hierarchical decomposition, for the same number of points, fewer parameters per model need to be learnt to achieve the same results. Thus a hierarchical classifier is, on average, expected to achieve better classification performance than a one-of-*n* output classifier.

4.10 Computational issues

Another considerable advantage of the fact that the hierarchical classifier decomposes the classification problem into a set of subproblems is that each subproblem from the root to the leaves becomes smaller in terms of the number of classes, and subsequently the number of data points to be considered. Also the size of the feature space will be significantly smaller at each node due to only a subset of the features being relevant to each subproblem. Also the number of training points for each model is significantly reduced for the pairwise classifier. These issues are addressed in this section.

4.10.1 Computational cost of training

It is interesting to analyse the computational cost of training each decomposition. It is reasonable to approximate the computational cost with the total number of training points used to train all the models in each decomposition, as the computational cost for most learning algorithms is proportional to the number of training points. The assumption is made that there are an equal number of training points per class, denoted by n_t. Consequently there are $n_t k$ points in the whole training set. This is a reasonable assumption since the size of the training set will usually scale with the number of classes.

For a one-of-*n* decomposition there are k models to be learnt, and each model uses all the points to distinguish one class from all the others. This leads to the total number of training points as:

$$N_t^o = n_t k^2. \tag{4.49}$$

For the hierarchical decomposition we will assume for simplicity that the hierarchy is balanced. In other words each node in the tree has an equal number of left and right children. In this case the number of levels is $\log_2 k$. At each level the number of points used in training is halved (assuming there are an equal number of points per class), but there are always twice the number of nodes for each level. In this way all $n_t k^2$ points are used once at the root, and $n_t k/2$ points are used at the two secondary nodes etc. The total number of points is then:

$$N_t^h = n_t k \log_2 k. \tag{4.50}$$

Consequently the number of points used in training is significantly smaller for the hierarchical decomposition than the one-of-n decomposition, increasingly so for large numbers of classes (increasing k).

The pairwise decomposition has $k(k-1)/2$ models which each compare $2n_t$ points. This leads to the total number of training points as:

$$N_t^p = n_t k(k-1). \tag{4.51}$$

It is plain that the hierarchical decomposition results in the least training, being of the order $O(k\log_2 k)$, where the others are $O(k^2)$, although this does not take into account the effort required to optimise the class hierarchy.

4.10.2 Computational cost of classification

To exactly calculate the k posteriors for each $\omega_i, i = 1,\ldots,k$, for a single unclassified point $\mathbf{x}$ all the models must be evaluated, regardless of the decomposition used. It is evident that the hierarchical and one-of-n decompositions will be faster by an order of k than the pairwise decomposition. However, for the hierarchical and pairwise decompositions there are methods that allow the computational complexity of classification to be reduced without affecting the MAP decision, although the posteriors will be approximated for classes with low probability.

When classifying an unseen point there can be a significant advantage to using a hierarchical decomposition of the problem. If one is using the MAP decision rule then it may not be necessary to calculate the posterior probability for each class. It is sufficient to calculate the posterior probabilities for the left and right subsets, i.e. $P(\Omega^l|\mathbf{x})$ and $P(\Omega^r|\mathbf{x})$, at each node and disregard the left or right subtree if the probability for that branch falls below a well-defined threshold. This technique is mentioned in (Schurmann, 1996), but a method for choosing the threshold is not given. A novel method for choosing a threshold is given here.

This threshold can be computed easily by considering the decision to be made at a node in the hierarchy. One must remember that the probability is being distributed between the left and right branches of the tree.

We wish to know the range of the posterior probability values for the classes in the left and right class subsets. Since the left posteriors (for the left class subset Ω_l) are constrained by:

$$\sum_{\omega \in \Omega^l} P(\omega|\mathbf{x}) = P(\Omega_l|\mathbf{x}),\ P(\omega|\mathbf{x}) \geq 0; \tag{4.52}$$

then we know that the maximum posterior probability for classes in the left subtree,

$$\alpha_l = \max_{\omega \in \Omega^l} [P(\omega|\mathbf{x})], \tag{4.53}$$

must lie within the range:

$$\frac{1}{|\Omega_l|} P(\Omega_l|\mathbf{x}) \leq \alpha_l \leq P(\Omega_l|\mathbf{x}); \tag{4.54}$$

where $|\Omega_l|$ is the number of classes in the left subset. So it is evident that if $P(\Omega_l|\mathbf{x}) < \frac{1}{|\Omega_r|} P(\Omega_r|\mathbf{x})$ then there is no need to compute the left branch any further. This relationship is of course symmetric and applies in a similar fashion for the right branch.

Now, the MAP decision rule assumes a 0/1 loss function, and if the posterior probabilities are required to be transformed by a loss matrix then the above will not apply, and all the posteriors will have to be calculated for every branch in the hierarchy.

Although there is a cost in calculating the posterior probabilities from the features at each node, the advantage gained by such computational cutoff will be more significant when the computational cost for extracting particular features becomes high. It is expected that fewer features will have to be calculated to make the final classification than any one-of-*n* classifier.

Similarly it is the case that when using a pairwise classifier it is not necessary to evaluate all the models. This is mentioned in (Jia and Richards, 1998), but a formal algorithm for selecting those models to evaluate is not given.

4.11 Training cost for a fixed-size problem

In the previous sections, the complexity of each class decomposition has been considered for an increasing number of classes. In reality the size of a classification problem is fixed, and the real problem is to reduce the training cost for a given number of classes and a fixed number of data points. This problem is considered in this section and it is indeed shown that the hierarchical and pairwise decompositions have an advantage over the one-of-n decomposition due to the reduced complexity of the decision boundaries for each submodel.

Consider first the pairwise decomposition. In many cases the decision boundary between any two classes will be linear or near linear. There will of course be some neighbouring classes which may have complex decision boundaries due to the particular application complexity, but on the whole, if any two classes are chosen they will have a simple decision boundary. This is the most attractive feature for the pairwise classifier.

In this case, it would be a sensible idea to attempt to train a linear model to discriminate between each class pair. Then if this linear model was found to be of insufficient complexity, then a non-linear model should be trained instead. In this way, the non-linearity is used only where it is needed, reducing the overall training complexity. To make this process more efficient the effort spent training the linear model should be conserved by initialising the non-linear model with the solution learnt by the linear model. This is readily achieved by initialising a MLP from a linear discriminant. If the decision boundary is only slightly non-linear, then the initial parameters of the untrained non-linear model will most likely be very close to their optimal values already. Thus the number of training cycles used for the complete pairwise classifier is expected to be reduced significantly.

The method used to initialise a MLP from a linear discriminant is quite simple, since the hidden nodes of an MLP are logistic linear functions in the input space, as is the logistic linear discriminant. The weight vectors for each of the hidden nodes can be initialised by

$$\mathbf{w}_i^h = \mathbf{w} \tag{4.55}$$
$$b_i^h = b, \tag{4.56}$$

where $\mathbf{w}_i^h$ is the weight vector for the ith MLP hidden node, b_i^h is the bias vector for the ith MLP hidden node, $\mathbf{w}$ and b are the weight vector and bias for the linear discriminant respectively. The hidden to output weights for the MLP then need to be initialised such that the MLP has the same output as the linear discriminant. This is done by setting

$$w_{1j}^o = \frac{2}{n^h} \tag{4.57}$$
$$b^o = -1, \tag{4.58}$$

where n^h is the number of hidden nodes, w_{1j}^o is the jth element of the one and only output node, and b_1^o is the bias for that node. In practice some random noise should be added to Equation 4.55 to allow each hidden node to diversify with further training.

This principle can be used effectively for the hierarchical classifier also. Given that the class hierarchy has been found such that the subproblems in the hierarchical decomposition are of linear or near linear complexity, then the same procedure can be applied. The agglomerative clustering procedure for finding a class hierarchy which is described in Chapter 5 has this property. This is expected to reduce the number of training cycles needed for the complete hierarchical classifier.

However, this principle is not as efficient for the one-of-n classifier, since the subproblems are less likely to be linear, or even near linear. It has been shown in this chapter that the complexity of the decision boundaries for each subproblem in the one-of-n decomposition increases with the number of classes, where it is constant for the pairwise and hierarchical decompositions. For a fixed-size problem, it is expected that many more of the subproblems with be of non-linear complexity than for either of the pairwise or hierarchical decompositions. Non-linear models may be initialised with the parameters of a linear model, but this would not be expected to significantly reduce the number of training cycles needed to specify the parameters of the non-linear model.

4.12 Summary

As solutions to a multicategory classification problem, three methods of decomposing the set of classes have been compared. Table 4.12 summarises the properties discussed in this chapter.

	One-of-n	Pairwise	Hierarchical
Number of models	k	$k(k-1)/2$	$k-1$
Permutations	1	1	$\prod_{n=1}^{k-1} 2n-1$
Number of points used in training	$n_t k^2$	$n_t k(k-1)$	$n_t k \log_2 k$
Number of models evaluated when classifying	k	$\leq k(k-1)/2$	$\leq k-1$
Suitable with linear techniques	No	Yes	Yes
Computational cut-off possible	No	Yes	Yes

Table 4.3: Summary of properties for the three class decompositions

A new taxonomy of multicategory classification has been presented and in particular the paradigm of decomposing the set of all classes into binary subproblems has been explored and three methods that appear separately in the classification literature have been defined and analysed under this paradigm.

An equivalence has been shown between the three decompositions when the required class conditional distributions are known perfectly. The relationships between the three methods have then been compared using linear discriminant models, showing that the hierarchical and pairwise decompositions have a considerable advantage over the one-of-n decomposition due to their ability to represent a greater proportion of problems of many classes using simple linear decision nodes.

A comparison in terms of the number of points needed to train each submodel and the number of comparisons needed in classification was also made. The hierarchical decomposition has the greatest advantage here, but the problem of designing the class hierarchy has not been addressed.

It has also been stressed that the need for non-linear models should only be justified if the binary submodels in each decomposition require it. In this case the one-of-n decomposition requires non-linear models where the pairwise and hierarchical decompositions do not, thus the need for non-linear models can be reduced to a minimum by adopting either of these decompositions, and the one-of-n decomposition unnecessarily complicates the form of the binary subproblems.

Chapter 5

Hierarchical structure identification

In Chapter 4 the complexity of the submodels in a class decomposition classifier was investigated. It has been argued that the complexity of the submodels in a pairwise classifier are constant for an increasing number of classes, that the one-of-n classifier has the undesirable property that the complexity of each submodel increases with the number of classes, and that the complexity of the submodels for the hierarchical classifier can be controlled by the design of the class hierarchy. This chapter describes methods that can be used to design the class hierarchy to minimise the complexity of each submodel in a hierarchical classifier with a critical comparison of their advantages and disadvantages.

An important difference from other tree-based classifiers such as decision tree classifiers is that the class hierarchy is finite and of constant size for a fixed number of classes, k. There are always $k-1$ nodes in the tree and each class is represented by the k leaves in the tree. This allows combinatorial optimisation techniques to be used to optimize the class hierarchy, where in decision tree classifiers the tree can be infinite and this restricts the choice of design procedure to a top-down divisive algorithm where the tree is designed greedily starting from the root. In this chapter, top-down divisive algorithms, bottom-up agglomerative algorithms and more general combinatorial optimisation techniques are discussed.

Each node in the hierarchical classifier breaks its input class subset into two output subsets and gives probabilities for each output subset:

$$P(\Omega_s^{\text{out}}|\Omega^{\text{in}},\mathbf{x}),\ s=1,2. \tag{5.1}$$

Criteria can be defined that evaluate a particular splitting of a class subset at a particular node, and these may then be combined to evaluate the whole class hierarchy.

5.1 Objective functions on individual nodes in the hierarchy

Given k classes contained in Ω^{in} that must be separated into two subsets Ω_1^{out} and Ω_2^{out}, there are $2^{k-1}-1$ ways of achieving this.

Each class is represented by a series of points in feature space with an associated class label, $[\mathbf{x_i}, \omega_k]$ for i points and k classes. Given a class subset, Ω_s^{out}, the points for each class may be aggregated according to whether the class label ω_k falls in the class subset.

Distance measures may be defined on the points in each class subset individually, or on the whole classes represented by their means.

5.1.1 Distance measures

Given a set of points, $\{\mathbf{x}_i\}, i = 1,\dots,N$, where $\mathbf{x}_i$ is a vector in P dimensional input space, a selection of distance measures can be applied between any two points (Friedman and Rubin, 1967; Basseville, 1989):

The Minowski metric is a general form of several well-known distance measures:

$$d_{ij} = \left[\sum_{k=1}^{d} \left|x_{ik} - x_{jk}\right|^{r}\right]^{1/r}, r \geq 1 \tag{5.2}$$

this leads to the Manhattan metric when $r = 1$, the Euclidean distance when $r = 2$ and the infinity norm when $r = \infty$.

Examples of these distance measures are described further below, with specific properties outlined:

- Squared Euclidean distance between two points:

$$S_{ij} = \sum_{p=1}^{P} (x_{ip} - x_{jp})^2 = (\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j) \tag{5.3}$$

  where $\mathbf{x}_i$ is the ith point and x_{ip} is the pth component of the same vector. This distance measure is invariant under any orthogonal transformation of the points in input space.

- The Mahalanobis distance between two points is a generalization of the sum of squares which is invariant under any non-singular transformation of the input space.

$$M_{ij} = (\mathbf{x_i} - \mathbf{x}_j)^T \Sigma^{-1} (\mathbf{x}_i - \mathbf{x}_j) \tag{5.4}$$

  where Σ is the covariance matrix for the points in input space. This has the disadvantage that the size of the covariance matrix increases with the number of input dimensions, and the inverse may involve significant computation in high-dimensional spaces.

If we assume that each class is normally distributed with equal covariance matrix then we may use the above distance measures directly on the means of each class distribution. If we assume that each class has equal covariance which is a multiple of the identity matrix (i.e. a spherical distribution) then we can use the sum of squares as a measure of distances between classes:

$$S_{ij} = \sum_{p=1}^{P} (\mu_{ip} - \mu_{jp})^2 = (\mu_i - \mu_j)^T (\mu_i - \mu_j) \tag{5.5}$$

where μ_i is the mean vector for the ith class and μ_{ip} is the pth component of the same vector.

If the assumption of equal class covariances is too strong it can be relaxed by using a form of the Batacharrya distance where Σ_i is the covariance matrix for the ith class,

$$B_{ij} = \frac{1}{8}(\mu_i - \mu_j)^T \left[\frac{\Sigma_i + \Sigma_j}{2}\right]^{-1} (\mu_i - \mu_j) + \frac{1}{2} \ln \frac{\left|\frac{\Sigma_i + \Sigma_j}{2}\right|}{\sqrt{|\Sigma_i||\Sigma_j|}}. \tag{5.6}$$

The first term reflects the separation due to the mean difference between the two classes and the second term reflects the difference due to the covariance (Kim and Landgrebe, 1991). Again this requires an inverse and now three determinants to be calculated, which may involve significant computational cost for high-dimensional spaces.

5.1.2 Entropy measures

Schurmann (Schurmann, 1996) suggests that an entropy measure may be used to evaluate the suitability of a choice of class subsets.

$$H(P(\Omega_s^{\text{out}}|\Omega^{\text{in}},\mathbf{x})) = \sum_{s=1}^{2} P(\Omega_s^{\text{out}}|\Omega^{\text{in}},\mathbf{x}) \log\left(\frac{1}{P(\Omega_s^{\text{out}}|\Omega^{\text{in}},\mathbf{x})}\right). \tag{5.7}$$

This is at its maximum when the probabilities are evenly divided between the classes, and a minimum when the majority of the probability mass is assigned to one class subset. This is similar in motivation to the metrics used in the C4.5 algorithms for the design of decision tree classifiers (Quinlan, 1993).

5.1.3 Fisher's criterion

The Fisher criterion can be used as a measure of separability between two classes at each level, it is defined as follows (Bishop, 1995):

$$J(\mathbf{w}) = \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_W \mathbf{w}} \tag{5.8}$$

where S_B is the between class covariance matrix:

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \tag{5.9}$$

μ_i being the mean of class i and S_W the total within class covariance matrix:

$$S_W = \sum_{n \in C_1} (\mathbf{x}_n - \mu_1)(\mathbf{x}_n - \mu_1)^T + \sum_{n \in C_2} (\mathbf{x}_n - \mu_2)(\mathbf{x}_n - \mu_2)^T \tag{5.10}$$

The weight vector $\mathbf{w}$ is given by:

$$\mathbf{w} = S_W^{-1}(\mu_2 - \mu_1). \tag{5.11}$$

Fisher's criterion is a measure of the distance between class means, normalized by the within class scatter along the direction of $\mathbf{w}$. This is an acceptable measure of separability of two classes, under the usual assumptions of Gaussian class distributions, but again requires a matrix inverse.

5.2 Objective functions on the complete hierarchy

The metrics described in the previous section measure the suitability of the splitting of a set of classes into two subsets. This can be used to determine the suitability of a single node in the tree. However, if the search for the most appropriate class hierarchy is done as a combinatorial optimisation on the set of all complete class hierarchies, then metrics that determine the suitability of the complete hierarchy must be used. This may be formulated as a combination of the above single-node criteria, or by way of the final classification rate. These are discussed below.

Any of the criteria described in Section 5.1 can be used to describe the whole tree by simply combining them for each node in the tree. The simplest way to do this is to do this is to calculate the sum of the criteria over each node. These may be weighted according to the depth of each node in the tree to allow the nodes nearer the root to have more of an effect on the total value. If each node is given equal weighting it should be noticed that the leaf nodes will dominate the total value simply because they are more numerous.

If accuracy were the only concern when training a classifier, then the classification rate is the most representative measure of this accuracy. Though sometimes other criteria such as interpretability are required from the final solution which are not always best represented by a measure of accuracy. Unfortunately, classification rate is the most computationally expensive of all measures since it involves feature selection and training to be performed on the classifier every time the measure is required. The distance measures given above are commonly used since they offer a less computational approximation to the classification rate.

Another possible criterion can be computed from the confusion matrix resulting from the complete trained k-class classifier (Schurmann, 1996). The confusion matrix is defined as:

$$\begin{aligned}\text{confus}[k,j] &= \text{relative frequency of patterns coming from class } k \\ &\quad \text{and being recognized as members of class } j \end{aligned} \tag{5.12}$$

A total error criterion is the combination of errors from classes in opposite class subsets across all nodes in the hierarchy. If N is the set of all nodes in the hierarchy and Ω_1^n and Ω_2^n are the two class subsets for node n:

$$\text{error} = \sum_{n \in N} \left(\sum_{k \in \Omega_1^n \wedge j \in \Omega_2^n \vee k \in \Omega_2^n \wedge j \in \Omega_1^n} \text{confus}[k, j] \right). \tag{5.13}$$

This value is to minimized when choosing a suitable class hierarchy. However this suffers from the same problem that the hierarchical classifier needs to be fully trained each time.

Given a suitable criterion for selecting a particular choice of class subset it is possible to recursively choose optimal class subsets to form a class hierarchy. When the number of classes is small this may be done systematically. However, when the number of classes becomes too large it is impractical to enumerate all possible choices of class subsets, even for a binary split. In this case an iterative clustering procedure can be used that optimizes a criterion by estimating cluster centres for the two class subsets, and then choosing subsets to be those classes best represented by each respective cluster centre. Clustering is presented in the next section.

5.3 Clustering

There are two main approaches to forming a hierarchy using clustering methods, either by combining singleton classes into class subsets and constructing the hierarchy from the bottom up, or by splitting the complete set of classes into class subsets, starting from the root, and building a hierarchy from the top down. These are known as *agglomerative*, and *divisive* methods respectively (Kim and Landgrebe, 1991) and are described below.

5.3.1 Agglomerative clustering

When designing a class structure from the bottom up, the goal is to group the classes together according to a similarity measure. The two most similar classes are joined to form a class subset. Each new class subset replaces its members and the process is repeated until the last two class or class subsets are joined. This technique is also used in an unsupervised manner when no class information is available by using individual points as initial sets and combining sets of points.

The similarity measures used can be chosen depending on the assumptions made on the class distributions. A suitable distance measure may be chosen from those described in section 5.1.1.

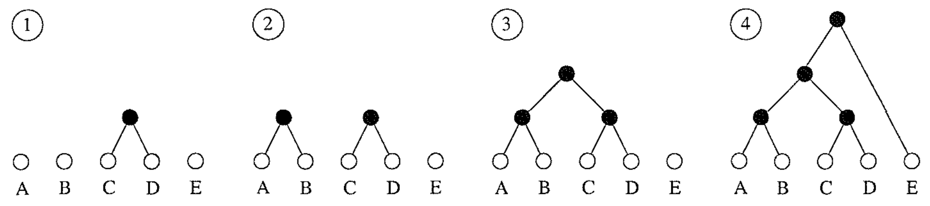


Figure 5.1: Example of agglomerative tree formation

These measures are used to compare sets of classes for joining and allow the agglomerative clustering algorithm to produce a class hierarchy.

The time complexity of this algorithm is dependent on the the number of comparisons that have to be made between pairs of classes. For a problem of k classes there will always be $k-1$ agglomerative steps in this algorithm. The first step requires $k(k-1)/2$ comparisons between pairs of classes and the second step then requires $(k-1)(k-2)/2$ comparisons. The last step requires no comparisons since there will be only two nodes to join. The number of comparisons is the sum of comparisons for each level:

$$N_{\text{agg}} = \sum_{m=0}^{k-2} (k-m)(k-m-1)/2. \tag{5.14}$$

The time complexity is then in the order of k^2 comparisons.

5.3.2 Divisive clustering

The top-down approach to structure design works in an opposite fashion by considering the dataset as a whole initially and then splitting it into smaller groups according to a performance criteria (Lin and Fu, 1983). Each group can then be considered for splitting itself and a hierarchy is built in a recursive manner.

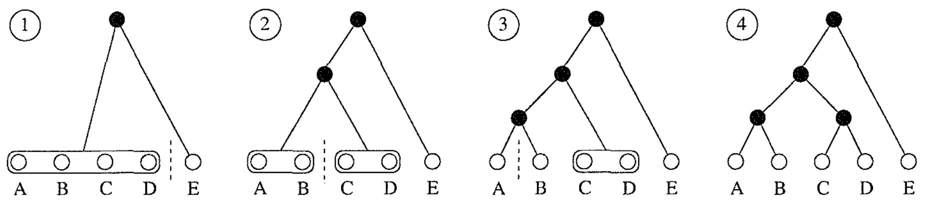


Figure 5.2: Example of divisive tree formation

The technique of splitting data into homogeneous groups is well researched in the literature on clustering, however it is dominantly an unsupervised learning task.

Although in this case the class labels are present and known, and this information should be used in designing the tree structure. The classes need to be grouped into class subsets that are most separable using the fewest necessary features. There are $k-1$ divisions when using divisive clustering, but the computational complexity of the clustering procedure performed for each division is not known as it will involve an unknown number of iterations depending on the random initialisation of the clustering procedure.

Existing clustering methods, and also, mixture methods for estimating mixture distributions may be used to determine clusters of interest in the data. Once clusters have been defined a decision must be made as to how to group the class into two distinct subsets using the clustering information.

5.4 Combinatorial optimisation

In the previous sections, the design of the class hierarchy has been described as a greedy search, where once a step in the procedure has been taken, it cannot be undone. Top-down and bottom-up procedures that involve $k-1$ divisive or agglomerative steps that construct the hierarchy level by level were described. In this section, an alternative method is proposed that takes advantage of the fact that there is a finite set of possible class hierarchies. This is combinatorial optimisation, whereby the set of possible class hierarchies is searched according to an objective function.

Combinatorial optimisation is a problem whereby the optimum solution is to be found in space which is non-linear and non-convex. Gradient information may not be available or necessarily useful in such problems hence the unsuitability of typical gradient-based algorithms. An objective function must be given that defines an ordering between states in the search space. Each specific algorithm defines how the state space is searched to find the mimimum-valued solution according to that objective function.

In terms of finding a solution to the class hierarchy, an objective function from Section 5.2 may be chosen. It has already been mentioned that if the number of classes becomes too large then a systematic, or exhaustive search is impractical. There are many potential algorithms that are designed to produce near-optimal solutions in such a situation, namely directed graph-searching techniques such as branch and bound, and statistical techniques such as simulated annealing and genetic algorithms.

The search strategies presented in this section strive to find a class hierarchy that gives a minimum value of the objective function. It is expected that there will be many local minima due to the discrete nature of the search space (the set of possible class hierarchies). There are several combinatorial optimisation techniques that have become standard in the literature, some of which are designed to overcome such local minima. As ever, there is no one algorithm which can guarantee the global minimum is found, but techniques can be used that increase the chance of finding a good result.

The discrete optimisation problem can be defined as finding a solution from the set of all possible states that minimizes a given objective function within any given constraints. The set of all possible states (state space) may be viewed in some cases as a graph, given operators that transform one state to another. Such graphs tend to grow exponentially with the size of a problem and optimal search techniques are NP-compete, that is the solution time increases exponentially with problem size for all algorithms. However, heuristic algorithms exist that can find sub-optimal solutions in polynomial time. Examples are directed depth-first search (DFS), cost-bounded DFS (IDA*), depth-first branch-and-bound (DFBB) and best-first search (BFS). Each of these has their merits and are discussed below.

The search algorithms mentioned so far are all deterministic. They attempt to overcome the problem of local minima by backtracking to previously unsearched paths in the graph that may potentially lead to better solutions. Alternative algorithms escape from local minima by using stochastic methods to search the graph. Simulated annealing allows all possible paths to be chosen with a given probability, hence backtracking is allowed. Genetic algorithms use stochastic crossover and mutation operators to explore the state space. These are also discussed below.

Jain and Zonker (1997) provide a taxonomy for a similar set of algorithms, with an empirical comparison using the Mahalanobis distance as criterion for a feature selection problem.

The next few sections describe discrete optimisation search techniques, starting with the deterministic graph searches, then stochastic graph searches, and finally genetic algorithms.

5.4.1 Deterministic graph search

Points in a discrete state space can be represented as node on a graph. The arcs between the nodes represent valid transformations from one state to another. Given an initial state as a starting point, a problem may be solved by traversing the graph until a valid solution is found. For some applications the graph may be a special case without loops. i.e. a tree. However, any graph may be searched as a tree if nodes are visited only once. This is the preferred method, and hence a graph search can be considered equivalent to a tree search.

Exhaustive tree-searching techniques such as depth-first search are simple, but for many applications this is impractical due to the exponential increase in time with increasing problem size. Methods are used to 'prune' branches of the tree to avoid unnecessary computation. These may be optimal, whereby the optimal solution can be proved to not lie in the pruned branch, or heuristic, where the chances of discarding good solutions, including the optimal one, is minimized.

The most extreme and least computationally expensive pruning technique is sequential forward search whereby only the best successive state, according to the objective function is chosen, and all others discarded. This does not allow any backtracking and the first minima found is offered as the solution. In the likely event of many local minima this will seldom find the global solution. This is also known as hill-climbing.

To introduce backtracking, instead of discarding successive states and consequently the paths resulting from those states, they are ordered in terms of the likelihood of offering the best solution. The second best may be visited once the best has been evaluated. This is also known as *directed-DFS*. Without pruning this is again an exhaustive search. Evidently, branches may be pruned after the *n*th best solution has been visited.

An alternative is to limit the depth of the search, in *depth-bounded DFS* the depth is limited and possibly extended if no satisfactory solution is found. This may be further improved by introducing a formal 'cost' criterion to replace the depth measure. This is known as *cost-bounded DFS*, a popular algorithm in this vein is IDA*.

Branch and bound techniques may be applied to DFS to prune branches of the search tree. A lower bound on the cost of a solution path that passes through a given state can be computed. If this lower bound of a certain node is larger than the cost of the best solution so far, then the node can be pruned. This is known as depth-first branch-and-bound (DFBB).

An alternative to depth-first search is to consider the most promising node of all nodes currently on the search frontier, across all branches, as opposed to concentrating on only one branch. This search technique is called best-first search (BFS). However this entails the overhead of storing the search frontier at all times, which may grow exponentially with search depth. Again bounds may be estimated to prune nodes, giving rise to best-first branch-and-bound (BFBB), also know as the algorithm A*. Many of these algorithms may be implemented on a parallel processor architecture (Grama and Kumar, 1999).

5.4.2 Simulated annealing

Simulated annealing is a combinatorial optimisation technique that borrows its motivation from the physical process of cooling a molten metal. In terms of hierarchical structure design, it is a stochastic search procedure that allows positive changes in the objective function. Alterations to the structure are chosen at random and if the change is negative the alteration is accepted, otherwise it is only accepted with a probability according to the size of the increasing step ΔJ and a temperature coefficient T:

$$P(A) = \begin{cases} e^{-\Delta J/T} & \text{if } \Delta J \geq 0 \\ 1 & \text{if } \Delta J < 0 \end{cases} \tag{5.15}$$

where $P(A)$ is the probability that step A is accepted.

This tolerance for steps of decreasing objective allows the algorithm to 'jump' out of local minima at high temperatures, where nearly all steps will be taken, and as the temperature is 'cooled' the algorithm will settle on a minimum, hopefully global.

Given a suitable objective function and operators that perform the 'steps' from one candidate solution to the next, then simulated annealing is easy to implement and relatively efficient, although many iterations of the algorithm need to be run. It can be difficult to know when to terminate the algorithm since it is never known when the solution found is the global optimum. Usually several runs of the algorithm are performed with different random initialisations in an attempt to avoid local minimum. Choices have to be made on the form of the random step, and the cooling schedule.

5.4.3 Genetic programming

Genetic programming is a branch of genetic algorithms that uses a tree structure to encode a set of actions (possibly a computer program) and evolves an initial random population of such structures (Koza, 1992). Those structures giving the best performance according to a fitness function (or objective function) are selected to generate the next population via operators such as crossover and mutation. Unlike hill-climbing the operators are not guaranteed to generate a valid solution and some computation will be wasted on evaluating garbage structures.

The set of possible trees used for class hierarchies is quite restricted, since each class must be represented as a leaf node. There must be no repetitions of classes, and none left out. To define a genetic encoding that allows crossover and mutation, the two fundamental operators of genetic algorithms, without generating excesses of garbage trees would be a complicated task.

Naturally the operators from hill-climbing can be used in genetic programming to provide a structure constrained to be correct, although this reduces to a parallel hill-climbing exercise, similar to performing many randomly initialized hill-climbs and has little genetic motivation.

Additionally, due to the fact that genetic techniques are population based, the number of evaluations is proportional to the size of the population. Evaluating a hierarchical classifier with many nodes is already quite computationally expensive, and a genetic algorithm would be expected to increase this by a factor of the population size for each step in the search.

5.5 Objective functions for a discrete search

In the previous section it was suggested that the class hierarchy of a probabilistic hierarchical classifier could be found by discrete optimisation. This is further investigated in this section. The problem is discussed in terms of the ideal solution, which will eventually lead to an objective function and search constraints.

It is proposed that the class hierarchy can be found by a search in solution space (i.e. the set of all possible class hierarchies). The search strategy must allow for four specific requirements on the eventual solution sought. That is:

- Accuracy - The final classifier should produce accurate classification results.
- Degrees of freedom - The number of parameters for the final model should be controlled to prevent over-fitting and the curse of dimensionality.
- Smoothness - The models must adhere to predefined constraints on the structure to prevent unbalanced trees.
- Prior knowledge - If a prior model is given, the information represented in that prior must be respected.

These are further explained in the sections below.

5.5.1 Accuracy

To measure the accuracy of a classifier the most direct measure is the misclassification rate, as this truly reflects the desired performance. However this can be costly to compute, as it involves the complete training of each classifier node in the hierarchy.

Criteria for evaluating individual nodes are given in section 5.1.1 and also criteria for evaluating complete tree are given in section 5.2. However, care should be taken when combining single node criteria to represent a complete class hierarchy. Simple summations will be dominated by the values for the leaves, simply because they are more numerous. If the nodes closer to the root are more important then they should be weighted as such when adding the separate node criteria.

5.5.2 Degrees of freedom

The dangers of over-fitting and the curse of dimensionality have been known for some time (Bishop, 1995). Controlling the effective number of parameters in a model is important, especially in terms of small datasets where the parameters need to be specified by a few data points in a manner that is statistically significant.

The number of nodes in the class hierarchy is fixed, due to the binary nature of the tree and the requirement that all classes should be present. The complexity of the model can be controlled by using the optimum minimal feature set at each classifier node.

However, distance measures such as in section (5.1.1) can only be used to compare features sets of the same dimensionality. This presents a problem since performing feature selection for each state to be evaluated in solution space will result in models of different dimensionality across nodes, rendering the summation meaningless. Again a solution to this would be to use the final classification accuracy as a performance measure but this is too computationally expensive. As an initial investigation, the search is performed on models containing all features, allowing such distance measures to be used.

Feature selection and training can be performed on the model given by the final class hierarchy.

5.5.3 Smoothness

For a class hierarchy to be meaningful and interpretable, it is likely that is should be well balanced (i.e. no particular branch should be significantly deeper than the others). A constraint on the maximum allowable depth of a single branch can be imposed on the required solution.

5.5.4 Prior knowledge

The data is not usually the only available source of information on the desired model. Prior knowledge should be used wherever possible to aid the construction of a good model. If a class hierarchy is suggested through domain knowledge then it may be desirable to find a solution that does not differ significantly from the given hierarchy.

Tree comparison algorithms are available, but can be computationally expensive. In some cases the distance between two trees is described by the number of transformations required to transform one tree to the other. This distance is effectively the search depth if the search is initialized with a prior hierarchy. This can be easily and efficiently incorporated as a constraint on the search.

The discrete optimisation problem can be defined as finding a solution from the set of all possible states such that minimizes a given objective function within the given constraints. The constraints may be incorporated as hard constraints on the states allowed by the search procedure, the search rejects any states that do not adhere to the constraint, or they may be incorporated as soft constraints by introducing extra terms in the objective function. Hard constraints used are the restriction on the search depth, or the amount of backtracking, effectively reducing the search space. Soft constraints can require free parameters to be estimated as coefficients for each term in the objective function which can be costly for large search spaces.

5.6 Operators on class hierarchies for discrete search

It has already been described in section 5.4.1 that a discrete space can be represented as a graph when an operator is defined that maps one state to another. This graph then becomes searchable if an ordering is imposed on the states via an objective function.

For the set of unordered binary trees that are used to represent the class hierarchy, one simple operator can transform one tree to another such that all possible trees may be reached from any starting tree by successive applications of the operator. A simple such operator is described here. A branch may be 'shifted' to the left or the right. A right shift is illustrated in Figure 5.3.

This operator may be applied to the left or right, and on most nodes in the tree, it can be interpreted simply as moving a member of an non-terminal node to the opposing non-terminal node. This may involve shifting a whole branch with many descendants. Any valid hierarchy may be generated from any other valid hierarchy using combinations of this operator. For simplicity only binary hierarchies are used.

One attractive property of this operator is that if a hierarchical classifier is trained,

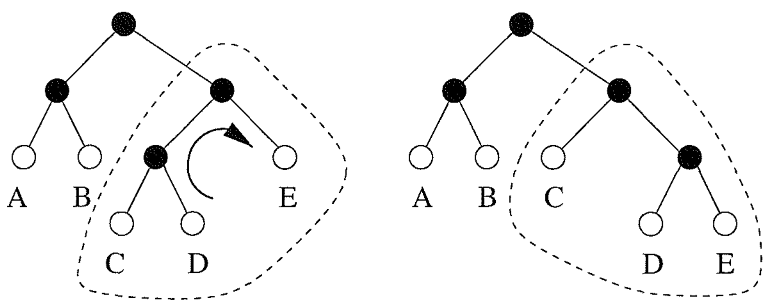


Figure 5.3: Example of branch shifting operator

and then the operator applied once, then only two nodes in the tree need to be retrained to retrain the whole classifier. The models for all other nodes are unaltered and do not need retraining. This leads to an efficient evaluation of the classification rate which may allow the classification rate to be used as the search objective in a discrete optimisation. The retraining time complexity does not depend on the number of classes in this case, whereas normally the training time complexity is in the order of k.

5.7 Summary

This chapter has described techniques used for choosing a specific class hierarchy, whether through construction, or a discrete search. The use of a discrete optimisation search on defining the class hierarchy is infrequently investigated in the literature, although it is mentioned very briefly by Shurmann (Schurmann, 1996). Potential algorithms have been described in this chapter. Although the specification of the class hierarchy can be of importance when using linear models, the impact of the class hierarchy diminishes for more flexible non-linear models, since for a perfectly flexible model the class hierarchy can be assigned arbitrarily without degrading the classification performance. There is a trade-off between the complexity of the model at each node and the effort required to optimise the class hierarchy. In Chapter 7 comparisons are made between a class hierarchy found using the reliable agglomerative clustering procedure using Euclidean distance measures, and a random class hierarchy to illustrate the impact on the hierarchical classifier.

Chapter 6

Class-dependent features

In Chapter 4 it was shown that one method of solving a many-class problem is to decompose the set of classes to produce a combination of 2-class problems. Each 2-class problem is solved conditioned on a reduced set of classes. In this chapter the novel idea is developed that certain features are only good discriminants when conditioned on a particular set of classes. For example the position of a loop in a handwritten character can only be useful for discriminating the set of characters that have loops from those that do not.

With the correct class subsets and well designed class-dependent features, the two concepts are mutually compatible in providing a good interpretable classification scheme for problems with many classes. Since class-dependent features are particularly application dependent it is useful to illustrate many of the points raised in this chapter with further examples from the chosen application in this thesis, namely handwritten character recognition.

In particular, this has implications for the hierarchical decomposition. With the one-to-one relationship between classes and leaves in the hierarchical decomposition (unlike decision tree classifiers), it is much more straightforward to interpret the structure of the tree. Each leaf node represents a complete and meaningful class concept, and each non-terminal node represents a group of classes that have common properties.

6.1 Definitions

Class-dependent features are defined as features whose discrimination ability varies significantly depending on the classes which are to be discriminated. This is unfortunately a rather vague definition since it is likely that the discriminative ability of any feature will vary to some extent due to variations between classes. A more rigorous definition is given later in Section 6.3 derived from the overlap of the class-conditional probability densities of a feature given a subset of classes.

The degree to which different features are class-dependent will in this case vary depending on the nature of each feature, but a further distinction can be made between *strong* and *weak* class-dependence. These are defined as follows:

- Weak class-dependence: the feature has a observable value for each class, and the feature and the class variable are conditionally independent given a set of classes.

- Strong class-dependence: when conditioned on the class variable, the feature may not have an observable value.

The weak sense of class-dependence can be easily demonstrated by showing the class-conditional distributions for each class and observing the overlap for different sets of classes (as illustrated later in Figure 6.4). The strong sense of class dependence is less amenable to standard analysis and can be explained by the following example from digit recognition.

Suppose a feature extractor were designed to detect loops in images. If a loop were present then it would select the minimum set of pixels that form the loop. If no loop is present then set of loop pixels remains empty. This is given as follows:

$$L = \phi(X) = \begin{cases} \{\text{set of pixels in loop}\} & \text{if loop is present} \\ \emptyset & \text{otherwise,} \end{cases}$$

where L is the set of pixel locations $L = \{(x_n, y_n) : n = 1, \ldots, N\}$, where N is the number of pixels in the loop. An example of the set of pixels selected for a character with and without a loop is shown in Figure 6.1.

The number of pixels selected, N, could be used as a feature to distinguish classes with loops from those without. This feature is an example of a class-dependent feature in the weak sense since it can distinguish ones from zeros, but not sixes from nines. If one wanted to distinguish a six from a nine, then one could take the average vertical position of all the loop pixels in the set L. This is defined as:

$$\hat{y} = \frac{1}{N} \sum_{(x,y) \in L} y.$$

This feature contains useful discriminatory information since an image with a loop that was above the centre of the image would most likely be a nine and one with

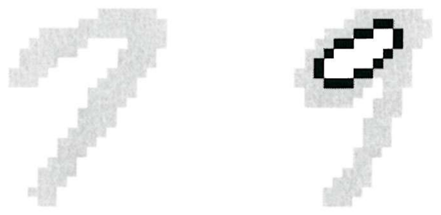


Figure 6.1: Example of a pixel-wise loop detector on the digits seven and nine

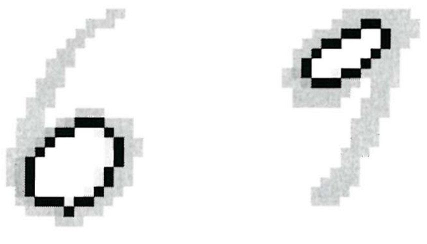


Figure 6.2: Example of a pixel-wise loop detector on the digits six and nine

the centre below would be a six (see Figure 6.2). However this feature only exists for those images from which a loop was successfully extracted. For those images that do not contain loops, the value of the feature is not only ineffective for discrimination, it does not even have a value.

Such undefined values for features are particularly problematic for statistical pattern recognition. Probability densities cannot be defined simply since the value of the feature is both categorical and real (i.e. either $\hat{y} = \{\text{undefined}\}$ or $\hat{y} \in \mathbb{R}$). These problems could be avoided by discretizing the feature into defined sets of values including the undefined value, but the resulting categorical variable would have to be unordered since there is no meaningful ordering for the undefined value. The existing ordering information would be lost.

An alternative would be to divide the image into distinct cells and accumulate the number of pixels in each cell that contributes to a loop. The value for each cell would be a positive integer and the positional information of the loop pixels could be used. This avoids the estimation of $\hat{y}$ by using values derived from the original N.

Although strongly class-dependent features as defined here are of interest and the author feels that analysis of such features would be useful for pattern classification with many classes, this thesis will concentrate on weakly class-dependent features since their analysis is more tractable. The analysis of strong class-dependent features is left to future work in Chapter 8.

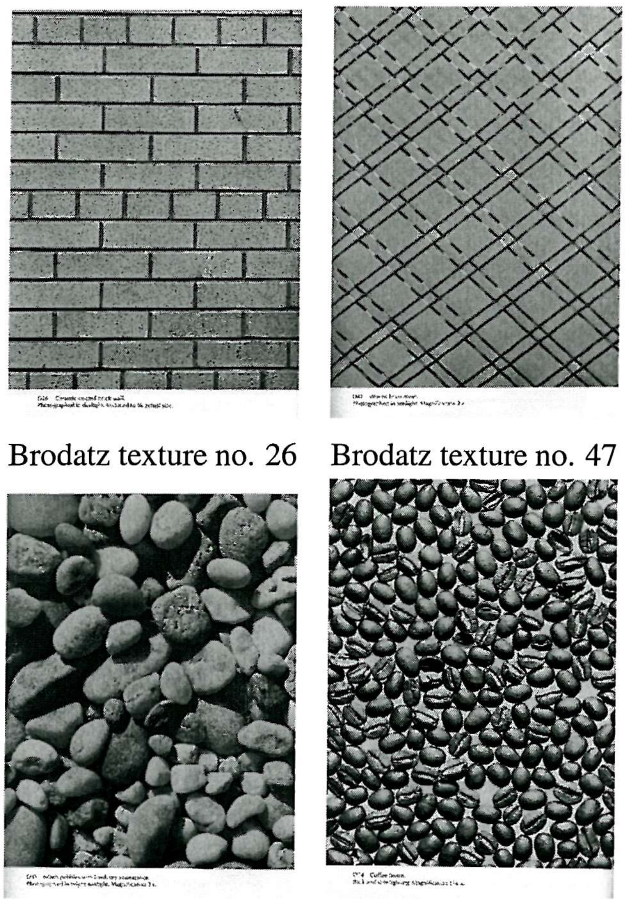
6.2 Evidence for class-dependent features in real-world data

There are many applications that show evidence of class-dependent features that may be exploited in a multicategory classifier. The handwriting example above is but one. A good example is the Brodatz texture album, which is used as a standard for comparing texture classification algorithms. The album consists of 112 photographs of both natural and man-made materials, ranging from cloth and fabrics to wood and stone textures. In the texture classification literature, many feature extraction techniques have been developed to provide features that give good discrimination between all the classes (Ng, Nixon and Carter, 1998). However, it may not be sensible to expect all feature extractors to have meaningful values for all textures.

Consider the four textures taken from the dataset in Figure 6.3. While global feature descriptors such as grey-level histograms can be sensibly generated from all four textures, it is clear that texture numbers 26 and 47 have properties not present in texture numbers 31 and 74, and vice versa. The former two textures have dominant lines that define the texture, whereas the latter two have rounded objects that have circular properties such as radius. Simply the confirmation of the presence of such features would allow the two pairs to be discriminated, and then the values for the relevant feature may be used to discriminate between textures within each pair.

However features described as such may be very difficult to design. To extract high-level circular features from the textures in Figure 6.3 would involve complex image processing algorithms such as the Hough transform, which may require significant fine tuning to be robust over different textures with similar properties.

An example of application that demonstrates class-dependent features, but one which does not rely on complicated feature extractors, is document classification. A document of text can be reliably described by the set of n most frequently occurring



Brodatz texture no. 26 Brodatz texture no. 47

Brodatz texture no. 31 Brodatz texture no. 74

Figure 6.3: Textures showing linear and circular features

words, excluding common words such as ‘and’ and ‘the’. Documents then may be distinguished by the occurrence of certain recognized words. This is a case where the context of the document in question greatly affects how it is categorized. Particular words will be present for particular subject areas, and the presence of these words can be used as class-dependent features.

For example a hierarchy of documents could be created by distinguishing different categories of subject matter. Suppose one high-level subject was ‘sailing’. Once documents had been classified as having a subject relating to sailing, then further classification could be done by searching for the words, ‘dingy’, ‘yacht’, and ‘ship’. Without the pre-conditioning on the documents about sailing, one would not expect these words to be common enough to be able to be useful discriminants.

The next section describes a method for measuring the class-dependent nature of features using the overlap of class-conditional density functions.

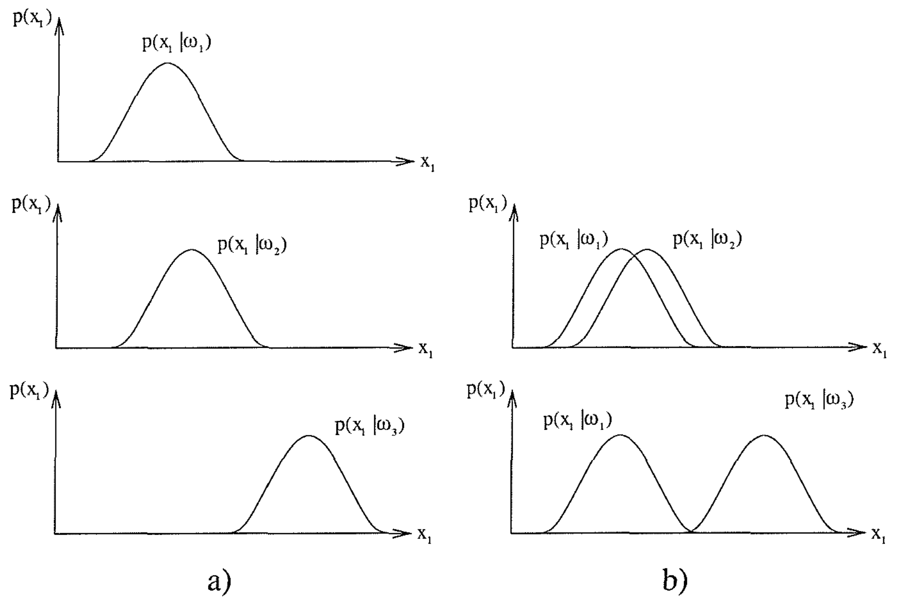


Figure 6.4: Class conditional distributions for three classes; a) the individual distributions for each class, b) the overlap between class pairs $\{\omega_1, \omega_2\}$, and $\{\omega_1, \omega_3\}$.

6.3 Conditional independence

Although there is compelling evidence that class-dependent features may be of use in a many-class classification problem, a formal definition and analysis of class-dependent features is needed to be able to quantify the possible benefits and reason about them clearly. A good starting point is the definition of class-dependent features using conditional independence.

This is an important concept regarding class-dependent features since it can be formulated as:

$$P(\omega|x_i, \Omega) = P(\omega|\Omega).$$

This means that the class variable, ω, and a particular feature, x_i can be independent conditioned on the set of classes, Ω. In other words, the dependence between features and classes may vary given a set of classes as a context.

Here the class-conditional probabilities are drawn as normal distributions. If we let the class label be described by a random variable W where $W \in \{\omega_i; i = 1, \ldots, k\}$ and the feature be represented by a random variable X where $X \in \mathbb{R}$. Then W and X are independent if $P(W = \omega_i | X = x) = P(W = \omega_i)$. (This is written as $P(\omega_i|x) = P(\omega_i)$

for brevity). This states that they are independent if knowing the value of X does not change our belief in the value of W.

Figure 6.4 shows class-conditional distributions for three classes that are not independent. As x_1 changes then we can make some inferences about the probable value of ω. If we condition the posterior probabilities on the subset $\Omega = \{\omega_1, \omega_2\}$ then it can be seen that x_1 and ω are now independent. Knowing the value of x_1 does not affect the probability of the class label significantly because the class-conditional distributions are very similar for the two classes. Therefore if we are making a decision between classes ω_1 and ω_2 then this feature is of little use and can be omitted. Conversely a measure of dependence is useful to select those features which are useful for discriminating between specific sets of classes. If we condition the posterior probabilities on the subset $\Omega = \{\omega_1, \omega_3\}$, x_1 and ω are now dependent. The value of x_1 now tells us with certainty the value of ω (low values of x_1 means the class label is ω_1, high values mean the class label is ω_3).

The degree of dependence between the class variable and specific features given opposing subsets of classes can be calculated via the area of overlap of the two class conditional distributions. If Ω_l and Ω_r are the two class subsets to be distinguished then the area of overlap between the two class-conditional distributions is:

$$O_{x_n}(\Omega_l, \Omega_r) \quad = \quad 1 - \frac{1}{2}\int_{-\infty}^{\infty} |p(x_n|\Omega_l) - p(x_n|\Omega_r)|\, dx_n. \tag{6.1}$$

If the overlap is 1 then the class variable is independent of the feature x_n. If there is no overlap then the feature would be a good discriminant between the two class subsets.

This measure would be a good indicator for a feature selection algorithm whose task was to select a set of features with good discrimination properties. The important issue is that the measure of discrimination is done between two sets of classes, and these two sets might not represent all the classes in the problem. There might be some classes whose interaction with the feature in question is of no importance. This matches the class decomposition paradigm where the models are learnt on subsets of classes, and it is argued in the next section that this overlap metric is particularly useful in determining the features for each submodel of a class decomposition classifier.

6.4 Applicability to many-class problems

It has been shown in Chapter 4 that advantages can be gained by breaking down many-class problems of a set of k classes, $\Omega = \{\omega_i; i = 1, \ldots, k\}$, into a combination of binary problems between two subsets of classes, Ω_l and Ω_r. The class subsets may both be single-element sets where $\Omega_l = \{\omega_i\}$ and $\Omega_r = \{\omega_j; j \neq i\}$. This leads to pairwise discrimination. Or the class subsets may be chosen such that $\Omega_l = \{\omega_i\}$ and $\Omega_r = \{\omega_j; \forall j \neq i\}$, this in turn leads to the popular 1-of-n classification. Or alternative Ω may be split by successive binary partitions of the classes that lead to a hierarchical classifier. This is discussed at length in Chapter 4.

By whatever method the classes are split into left and right subsets, feature selection should be performed for each separate binary decision. The analysis of class-dependent features is concerned with selecting different feature sets for different decisions between sets of classes. This is called *local feature selection* and is achieved via the overlap metric defined in the above section. Section 6.7 describes how this metric may be used for feature selection.

Although studied for over three decades now, the process of selecting the optimal feature set from a large set of possible features has proved a largely unsolved problem due to the combinatorial explosion of the number of possible feature subsets when the number of features is large. Information on class-dependent features can aid the feature selection process to guide the search for the optimum features sets in a many-feature, many-class classification problem.

Several class separability metrics are well-known and used as the objective function in feature selection, however none explicity define the sets of classes that are being discriminated. This information is of course *implicit* in the method used for feature selection and part of the problem definition whenever feature selection is being performed for any classification problem. The difference here is that the choice of class subsets is being stated clearly as part of the metric.

6.5 Previous work

There is little citable work using the term *class-dependent features* except for Oh, Lee, and Suen of the Department of Computer Science at the Chonbuk National University in South Korea (Oh, Lee and Suen, 1998; Oh, Lee and Suen, 1999). The reader is referred to Section 3.8 for a discussion on other work related to class-dependent features.

The parallel work by Oh *et al* mirrors some of the work in this thesis and is described below. The work presented in Chapter 7 extends the work described here by showing the effect of class-dependent features across the set of class decomposition models described in Chapter 4.

A class-dependent feature is defined as a feature that has different merits to different classes in terms of discriminating power. This is entirely in accordance with the definition given in this thesis, however the more formal definition using conditional independence is preferred. The notation has been changed from Oh's original paper to fit the standard statistical pattern classification notation used in this thesis.

Oh *et al* define metrics on the separability of classes and class subsets given a particular feature x_n. These are defined below. A separation between two classes is defined as:

$$S^{cc}_{x_n}(\omega_i,\omega_j) = \int_{-\infty}^{\infty} \left| p(x_n|\omega_i) - p(x_n|\omega_j) \right| dx_n. \tag{6.2}$$

Then the class separation for a group of classes is formulated as:

$$S^{g}_{x_n}(\Omega) = \sum_{\omega_i\in\Omega} \sum_{\omega_j\in\Omega, j\neq i} S^{cc}_{x_n}(\omega_i,\omega_j). \tag{6.3}$$

Similarly, the class separation between a single class and a group of classes is given as:

$$S^{cg}_{x_n}(\omega_i,\Omega) = \sum_{\omega_j\in\Omega} S^{cc}_{x_n}(\omega_i,\omega_j). \tag{6.4}$$

A non-parametric kernel-based density estimator is used to evaluate $p(x_n|\omega_i)$. Equation 6.2 represents the degree of overlap between the two class-conditional probability densities for classes ω_i and ω_j.

The method used to select a particular feature set to discriminate one class ω_p from the set of all other classes $\Omega = \{\omega_j, \forall j \neq i\}$ simply selects the set of features with the highest values of $S^{cg}_{x_n}(\omega_i,\Omega)$ for all features, $x_n, n = 1,\ldots,d$.

A modular neural network is used to perform the k-class classification using a set of k multi-layer perceptrons, $M_i, i = 1,\ldots,k$. Each MLP, is trained using the specific feature set selected for the class ω_i. The outputs for the ith model (O_{i1}, O_{i2}) are trained to distinguish between *positive samples* which belong to class ω_i where $(O_{i1}, O_{i2}) = (1.0, 0.0)$, and *negative samples* which belong to any of the other classes

	training set	test set
CGD	98.92	95.10
DDD	98.97	97.30
class-common	99.42	97.60
class-dependent	99.47	97.85

Table 6.1: Comparison of recognition rates for CENPARMI database from Oh, Lee and Suen (1999)

	training set	BS test set	good BS test set
CGD	99.52	96.50	98.15
DDD	99.49	96.55	98.42
class-common	99.68	96.98	98.59
class-dependent	99.71	97.27	98.73

Table 6.2: Comparison of recognition rates for CEDAR database from Oh, Lee and Suen (1999)

in $\Omega_i = \{\omega_j, \forall j \neq i\}$ where $(O_{i1}, O_{i2}) = (0.0, 1.0)$. The final output of each classifier is the difference $y_i = O_{i1} - O_{i2}$. The final class is chosen as ω_i for the model with the greatest output y_i.

In actual fact the models M_i can simply be used to output the posterior probability of class ω_i given the input vector $\mathbf{x}$, $P(\omega_i|\mathbf{x})$, and the class with the greatest posterior probability chosen. This is due to the fact that an MLP correctly trained will approximate (Bishop, 1995) O_{i1} and O_{i2} as $P(\omega_i|\mathbf{x})$ and $P(\Omega_i|\mathbf{x})$ respectively, where $\Omega_i = \{\omega_j, \forall j \neq i\}$.

Since $P(\omega_i|\mathbf{x}) = 1 - P(\Omega_i|\mathbf{x})$, then y_i as above reduces simply to $2P(\omega_i|\mathbf{x}) - 1$ which is proportional to $P(\omega_i|\mathbf{x})$.

The results show that a small but significant increase in classification performance can be achieved on a handwritten digit dataset (10 classes) when using class-dependent features. This is shown in Tables 6.1 and 6.2 replicated from Oh, Lee and Suen (1999). In these tables, CGD refers to the use of Contour-based Gradient Distribution features which are represented by a feature vector of 256 real values that are derived from edge detection gradients in each image. DDD refers to the use of Directional Distance Distribution features which are represented by a vector of 256 real values which represent distances between pixels in the image. Details of these algorithms can be found in

(Srikantan, Lam and Srihari, 1996; Oh, Lee and Suen, 1998). The results in the rows labelled *class-common* give the classification rates when selecting the same common 256 features from the 512 combined CGD and DDD features for each class. The results in the rows labelled *class-dependent* give the classification rates when selecting different sets of 256 features for each of the 10 classes. The CENPARMI database is a dataset of handwritten digits which consists of 4000 training samples and 2000 test samples. The CEDAR database consists again of handwritten digits with 18,468 training samples and 2,711 test samples in the BS test set and 2213 test samples in the good BS test set. The good BS test set was constructed using the well-segmented samples from the BS dataset (Oh, Lee and Suen, 1999). A consistent improvement is shown when using class-dependent features.

This work is extended in this thesis by considering local feature selection for the set of class decomposition models. The modular neural network used in Oh's work is an example of the one-of-*n* class decomposition classifier, and results are shown in Chapter 7 that show the same results can be obtained for pairwise and hierarchical decompositions. In Oh's work half of the available features are selected, where in Chapter 7, results are shown when varying the fraction of features selected, for both linear and non-linear models to show that this trend is consistent across all models and fractions of features selected.

It is interesting to note a very early paper (Swain and Hauska, 1977) suggests a manual method for selecting class-dependent features (although not by that name) via a 'coincident spectral plot', which plots for each feature the ranges of values for each class. In this case the features are defined as specific ranges in a spectral band of satellite signals. From this diagram it can easily be seen that for particular features and some subsets of classes there is little or no overlap in feature space, showing that that feature would be a good discriminant for those particular classes. This is in fact a visual interpretation of the separation measures given in the section above.

6.6 Consequences for classifier design

One important consequence of using class-dependent features is that when describing a class by a set of features, one no longer needs to restrict the choice to features realisable for all classes. In fact it would be more advantageous to actively choose features that represent most specifically that class without concern for the applicability of that feature to the other classes. In this way, the class-dependent features will be a more accurate representation of each class.

This could have an impact for classifier design and the choice of measurements taken, and the choice of features extracted. If certain features are expected to have discriminatory information for only a subset of classes by design, then this can be reflected in the choice of features for specific submodels in a class decomposition classifier. This information will aid the feature selection for each submodel. When using the hierarchical classifier, there is flexibility in the choice of the class hierarchy, and this could also reflect any grouping of classes by common properties which are known a priori at the design stage.

The fundamental issue is that since each class is represented as a whole concept for all of the class decomposition classifiers, and the set of classes is manipulated with this in mind, then the classifiers are more interpretable. The relationships between features and classes can be expressed by the classifier, and this property can be used to either aid the design through prior knowledge, or aid the understanding of the problem by interpreting an automatically designed classifier. This automatic design process involves feature selection at each submodel, called *local feature selection*, before adapting the parameters of the each final submodel. This is discussed in the next section and contrasted with the alternative method of *global feature selection* where each submodel uses the same feature set.

6.7 Global and local feature selection

In the history of pattern classification and feature selection it has more often been the case that a set of features is selected for the problem in general, and there is seldom a formalism for selecting features for decompositions of the problem. However, given the decomposition of the set of classes as described in Chapter 4, it is straightforward to apply feature selection to the subproblems within each model in the decomposition, regardless of the decomposition chosen.

The distinction is made here between *global* feature selection, whereby the set of features is selected for the whole problem, and each submodel shares the same feature set, and *local* feature selection where a separate feature set is selected for each submodel independently of all other submodels.

If the cost of evaluating features is high then the hierarchical classifier is particularly useful when used with computational cut-off since for all but the ambiguous classification queries, very few features will have to be evaluated (see Section 4.10.2). A typical application would be automatic target recognition (Dodd, Bailey and Harris, 1998; Harris, Bailey and Dodd, 1998).

It is expected that for a fixed size of feature set, that if features were selected locally for each model in any of the class decompositions then this would result in a higher classification rate than selecting a single feature set of the same size to be used globally for all models. However the classification system using local feature selection will effectively be using more features overall than the system with global feature selection. If the purpose was to reduce the number of features to be evaluated due to a high cost of feature evaluation then it would seem there would be no gain in using local feature selection.

However this is not the case if not all the models in the class decomposition need to be evaluated. Computational cut-off has already been discussed for hierarchical and pairwise decompositions in Chapter 4. In the case of hierarchical decomposition all but the ambiguous points can be classified using a minimal set of nodes in the decomposition. This means that the number of features evaluated for the majority of points will be minimised. This is in contrast to the one-of-n decomposition where all k models must be evaluated. For the hierarchical case, assuming a balanced tree, only $\log_2 k$ models must be evaluated. This is a considerable saving on the number of features evaluated for each point.

Feature selection algorithms tend to be greedy forward searches or backwards eliminations, or a combination of both (Jain and Zonker (1997) and Dash and Liu (1997) review feature selection algorithms). A metric needs to be defined that orders the features in terms of the expected discrimination ability for the classification problem. The metric may apply to single features at a time or a combination of features whereby correlations between features are taken into account, and not just the correlation between the feature and the class label. For class-dependent feature selection the metric needs to be conditioned on a specific subset of the classes. The overlap metric defined in Section 6.3 is suitable for this purpose, although feature to feature correlations are not measured. A selection algorithm needs to be chosen that selects a suitable subset of features given a specific class subset. If there are many features then the number of possible feature sets is prohibitively large for an exhaustive search, so a practical solution is to simply order the features using the defined metric and select the p best features. This is the technique used in the Chapter 7 for both local and global feature selection.

6.8 Summary

In this chapter the concept of class-dependent features has been formally defined and motivated from examples of real-world data. It is expected that in problems of many classes and many features that not all features will be good discriminants for all the classes. This is expressed by the concept of conditional independence of the predicted class label and the feature, given a subset of classes to choose from. A metric can be calculated via the overlap of the class-conditional densities for each feature for the classes in the subset. Features can be selected for the submodels in a class decomposition classifier using this metric, with the purpose of reducing the number of features for each submodel.

This is a novel approach, although there has been work in parallel that mirrors some of the work presented here (Oh, Lee and Suen, 1999). The development of multicategory classification in conjunction with class-dependent features is important. The use of class-dependent features to select feature sets for submodels in a combination of 2-class classifiers is studied in this thesis for several different class decomposition paradigms. This further develops the work in (Oh, Lee and Suen, 1999), where only the one-of-n paradigm is explored.

These issues are explored in practise by considering a handwritten recognition problem. This is described in Chapter 7 where experiments using real and simulated data are detailed.

Chapter 7

Analysis of simulated and real-world data

A number of experiments were carried out to evaluate the effectiveness of the algorithms presented in this thesis. Several issues are under evaluation here and they are:

- comparison of one-of-n, pairwise, and hierarchical decompositions for many class problems for increasing numbers of classes,
- demonstration of linear and non-linear models for each decomposition,
- comparison of global and local feature selection for each decomposition,
- demonstration of the performance of each decomposition on real-world data, and
- demonstration that the class hierarchy can aid problem understanding.

To allow ample control over the experiments and to aid the understanding of the performance of each algorithm a number of simulated datasets are used in the experiments. Peformance is also measured in experiments on real-world data. For the hierarchical classifier, the class hierarchy was found by agglomerative hierarchical clustering as described in Section 5.3.1 or chosen randomly.

All the experiments were carried out in software, and the first section of this chapter describes the design of the software that was written specially for the experiments.

7.1 Software design

The design and implementation of the software used to generate the decision boundary diagrams in Chapter 4 and the experimental results in this chapter is detailed here.

The code was developed using an object-oriented design methodology in C++. Class diagrams are presented using the Unified Modelling Language (UML) which is an industry standard for object-oriented modelling descriptions. This section presents an outline of the software design and is not intended as a complete description of the source code, so most of the detail of the specific class methods and attributes is omitted for clarity.

In all, over 18,000 lines of original code was written for the evaluation of the results in this thesis. The programs used were compiled using the g++ compiler and ran under Linux, outputting the results to text and image files. During the course of the PhD, and additional to the lines of code mentioned above, both X-Windows and Microsoft Windows interfaces were developed, but they are not documented here as they do not play a part in the final results.

The object-oriented design played a significant role in the development of the theory and vice-versa. The object hierarchical shown in Figure 7.1 has similarities to the classifier taxonomy shown in Figure 4.1. Through implementation of the different classification algorithms, the similarities and differences of the one-of-n, pairwise and hierarchical classifiers are highlighted. This is borne out in the object hierarchy.

7.1.1 The classifier object hierarchy

The most fundamental concept in the software design is the classifier and this is reflected in the implementation by a set of C++ classes derived from the single **Classifier** base class. The base class and its derived classes are shown in Figure 7.1. The base class defines the attributes and methods for a generic classifier as shown in Table 7.1. The **Classifier** base class supplies virtual methods for firstly initialising a binary classifier or multiclass classifier, and secondly for training and classification, and these are implemented appropriately by the classes that inherit from the base class.

The **LinearDiscriminant**, **BayesClassifier**, **MLPClassifier**, and **Compound-Classifier** are all direct specialisations of the **Classifier** base class. The **OneOfN-Classifier**, **PairwiseClassifier**, and **HierarchicalClassifier** are specialisations of the **CompoundClassifier** class. This is due to the common behaviour of these compound classifiers (in the rest of the thesis these are called class decomposition classifiers), since they all define a list of **Classifier** objects within them. They differ in the initialisation procedure, and in the classification procedure used to calculate the final posterior vectors. This is reflected by the fact that each of the three compound classifiers need only define the *initialise_multiclass()* and *classify()* methods. The *train()* method is the same for all three and involves simply training the list of classifiers that have been

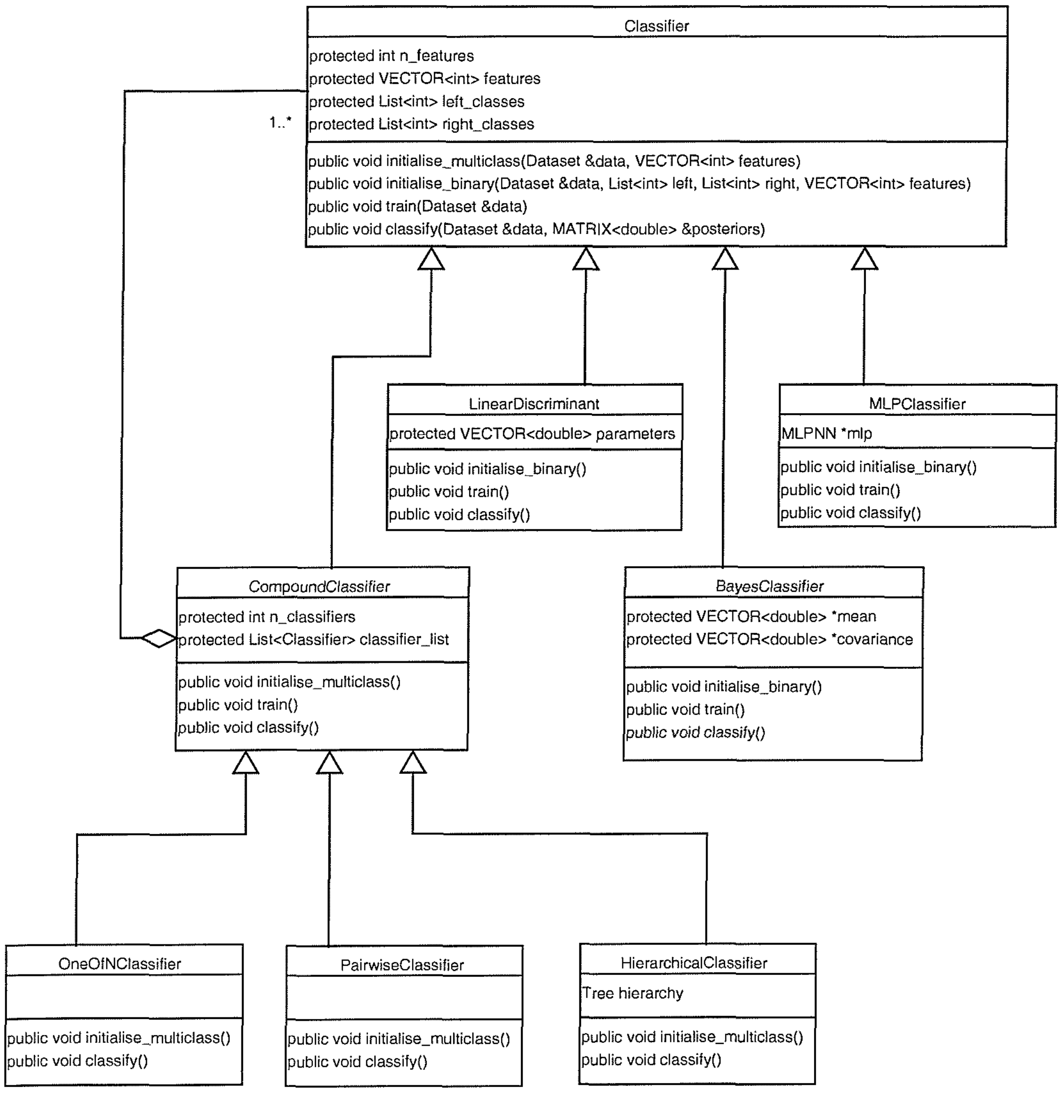

Classifier
protected int n_features
protected VECTOR<int> features
protected List<int> left_classes
protected List<int> right_classes
public void initialise_multiclass(Dataset &data, VECTOR<int> features)
public void initialise_binary(Dataset &data, List<int> left, List<int> right, VECTOR<int> features)
public void train(Dataset &data)
public void classify(Dataset &data, MATRIX<double> &posteriors)
1..*
LinearDiscriminant
protected VECTOR<double> parameters
public void initialise_binary()
public void train()
public void classify()
MLPClassifier
MLPNN *mlp
public void initialise_binary()
public void train()
public void classify()
CompoundClassifier
protected int n_classifiers
protected List<Classifier> classifier_list
public void initialise_multiclass()
public void train()
public void classify()
BayesClassifier
protected VECTOR<double> *mean
protected VECTOR<double> *covariance
public void initialise_binary()
public void train()
public void classify()
OneOfNClassifier
public void initialise_multiclass()
public void classify()
PairwiseClassifier
public void initialise_multiclass()
public void classify()
HierarchicalClassifier
Tree hierarchy
public void initialise_multiclass()
public void classify()

Figure 7.1: The classifier class diagram

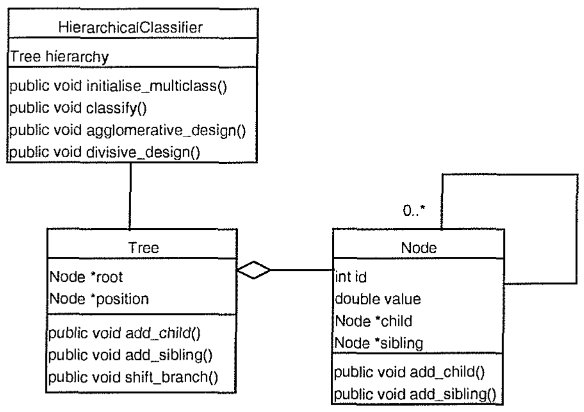


Figure 7.2: The hierarchical classifier class diagram

setup, and this method is supplied by the **CompoundClassifier** base class.

7.1.2 The hierarchical classifier

The **HierarchicalClassifier** class is a little more complicated than the other two compound classifiers since it involves the implementation and manipulation of the class hierarchy using a tree data structure. These are implemented via the **Tree** and **Node** classes as shown in Figure 7.2. The *id* attribute in the **Node** class associates the node in the tree with one of the binary classifiers defined in the **CompoundClassifier** class. Extra methods are included in the **HierarchicalClassifier** class to implement algorithms to design the class hierarchy.

7.1.3 The multi-layer perceptron classifier

The **MLPClassifier** class is in fact a wrapper for the **MLPNN** class kindly supplied by Steve Gunn and Jasvinder Kandola. Originally designed for regression problems, the code was adapted to model classification problems and called from within the **MLPClassifier** methods. The **MLPNN** class uses a **ScaledConjugateGradient** class written by Steve Gunn to adapt the model parameters.

7.1.4 Associated classes

Other important structures are described in this section, most notably the **Dataset** class which is used to store and manipulate the application data for training and testing of

Attributes	**Description**
features	A vector of integers that define which features from the dataset are to be used for training and classification
left_list	A list of left class indices for binary classifiers
right_list	A list of right class indices for binary classifiers
Methods	**Description**
initialise_binary()	Initialises classifier parameters prior to training to solve a binary classification problem
initialise_multiclass()	Initialises classifier parameters prior to training to solve a multicategory classification problem
train()	Adapts the model to fit the training data
classify()	Outputs posterior probabilities for the test data
nonparametric_feature_select()	Selects features for the classifier
global_nonparametric_feature_select()	Selects features globally for the classifier
calculate_mse()	Calculates the mean squared error from the dataset and posteriors
calculate_classification_rate()	Calculates the classification rate from the dataset and posteriors
output_confusion_matrix()	Outputs the classification results for a trained classifier as a confusion matrix
draw_boundary_to_dmatrix()	Draws decision boundary diagrams for a trained classifier

Table 7.1: Attributes and methods for Classifier class

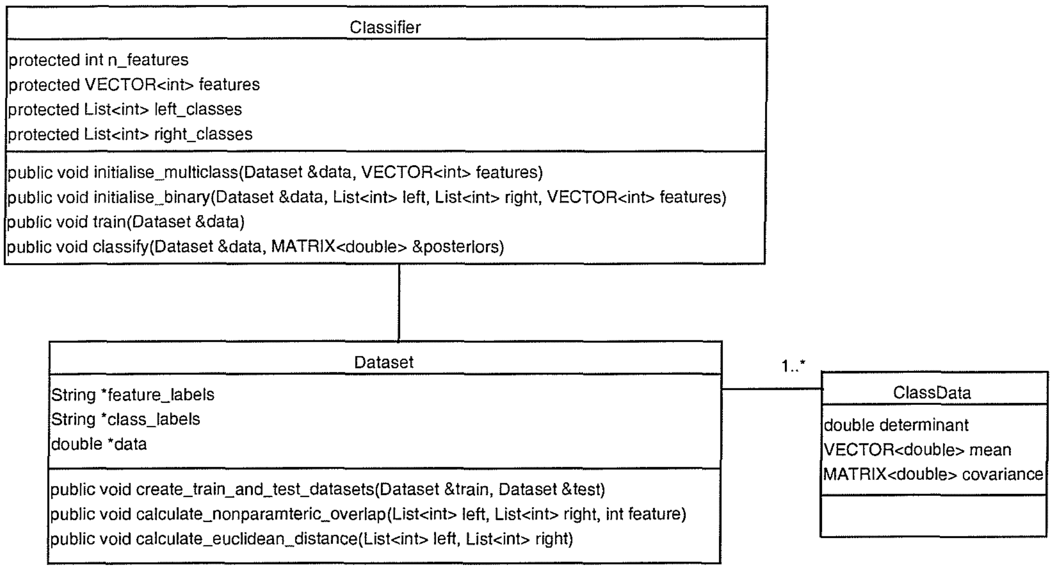

Classifier
protected int n_features
protected VECTOR<int> features
protected List<int> left_classes
protected List<int> right_classes
public void initialise_multiclass(Dataset &data, VECTOR<int> features)
public void initialise_binary(Dataset &data, List<int> left, List<int> right, VECTOR<int> features)
public void train(Dataset &data)
public void classify(Dataset &data, MATRIX<double> &posteriors)
Dataset
String *feature_labels
String *class_labels
double *data
public void create_train_and_test_datasets(Dataset &train, Dataset &test)
public void calculate_nonparamteric_overlap(List<int> left, List<int> right, int feature)
public void calculate_euclidean_distance(List<int> left, List<int> right)
1..*
ClassData
double determinant
VECTOR<double> mean
MATRIX<double> covariance

Figure 7.3: The dataset class diagram

the classifiers.

An important and fundamental class is the **Tensor** class which is used to implement the *VECTOR* and *MATRIX* data types. This has been implemented and refined over time by Steve Gunn. Vector and matrix operations used from this class are the addition and multiplication of vectors and matrices, and the determinant and inverse operators on matrices.

Other utility classes implemented were **List**, and **ListPtr**. While there are standard libraries for this type of class, at the outset it was felt that for portability between compilers these classes should be implemented specifically for the PhD software. With the implementation of the C++ Standard Template Libraries (STL) for most compilers now, the STL classes would be used for future projects.

7.1.5 Summary

This section has described the class structure for the software used to implement the algorithms needed to perform the experiments in this thesis. The remainder of the chapter describes the experiments and the results, starting with the generation of simulated datasets.

7.2 Simulated datasets

Simulated datasets have been used to confirm the theory presented in Chapter 4 regarding the performance of the three decompositions as the number of classes increase. The use of simulated datasets is promoted as the complexity of the problem can be controlled and the distributions of the data known. This allows the experiments to investigate assumptions of normally distributed classes without any unknown external effects. Datasets were generated with an increasing number of classes. These were then classified using one-of-n, pairwise, and hierarchical decompositions. Two types of dataset were generated, one to show the performance of each decomposition, and the second to show the effect of feature selection for the hierarchical model in particular.

7.2.1 Normally distributed classes

The first of the two types of simulated dataset generated is characterised by normally distributed classes. The points for each class were generated as such; 200 points were generated for each class according to a multivariate Gaussian distribution. The covariance matrix for each Gaussian was fixed as σI where I is the identity matrix. The mean for each Gaussian was generated as a uniform vector in the unit hypercube, which was fixed at 2 dimensions. Example plots of typical datasets generated using this method are shown in Figures 4.5 and 4.6.

7.2.2 Best-case hierarchical datasets

A second set of datasets was generated to illustrate the effect of feature selection for the hierarchical classifier. To illustrate the example, a dataset of three inputs and four classes is used. The class-conditional distribution for each input is shown in Figure 7.4. The classes are grouped such that $\Omega_1 = \{C_1, C_2\}$ and $\Omega_2 = \{C_3, C_4\}$. A sample dataset in 3 dimensions from this distribution is shown in Figure 7.5. Although this type of dataset is particularly contrived, it does illustrate the best-case scenario for the hierarchical classifier.

7.3 Comparisons of algorithms on simulated data

Datasets were generated as described in Section 7.2.1. The variance σ was defined such that there was little overlap between points of different classes. Four types of classifier were compared, namely the one-of-n classifier, the pairwise classifier, and

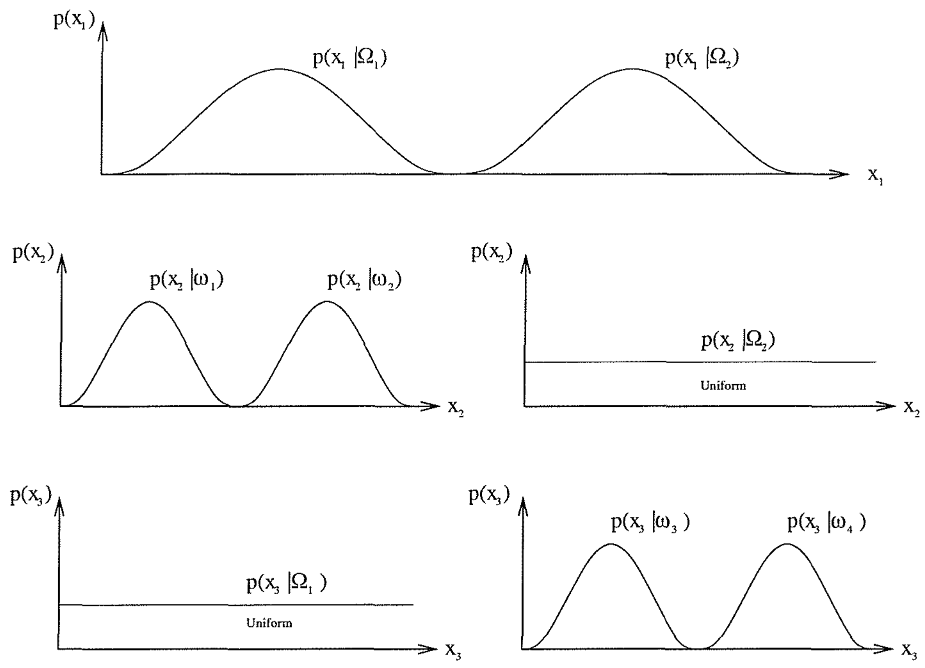


Figure 7.4: Probability distributions for three inputs and four classes

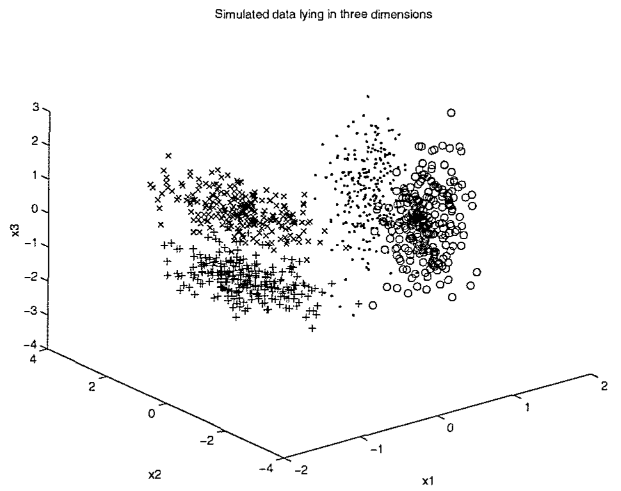


Figure 7.5: Example of a simulated dataset with four classes generated such that subsets of classes can be discriminated using a single input variable

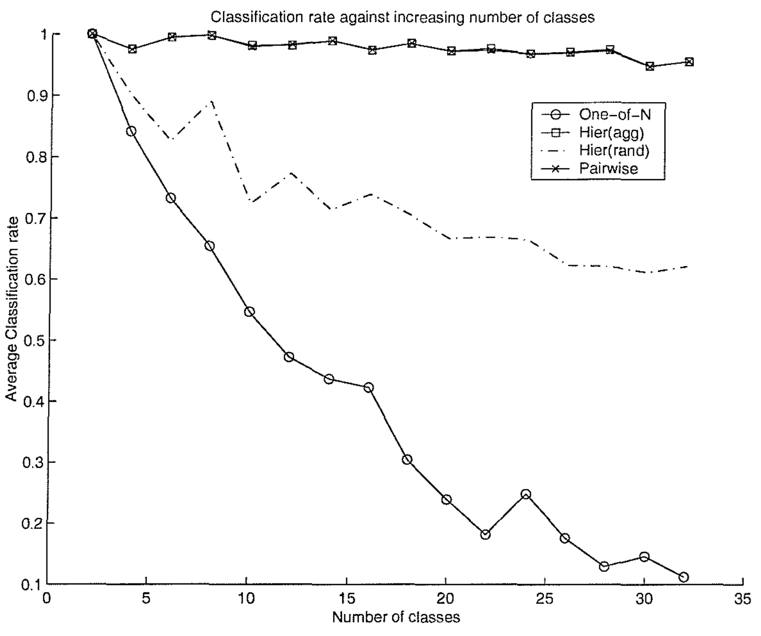
the hierarchical classifier using both a random class hierarchy, and a class hierarchy found using the agglomerative clustering procedure using an Euclidean distance metric as described in Section 5.3.1. The form of the submodels used within each of the above class decompositions was also varied. Firstly the classification rates using a logistic linear discriminant are shown in Figure 7.6. Secondly the classification rates using a multi-layer perceptron are shown in Figure 7.7. Thirdly the classification rates using a Bayes plug-in classifier using Gaussian class-conditional density estimation (see Section 2.6) are shown in Figure 7.8.

The multi-layer perceptrons each had one hidden layer of 5 nodes and were batch trained using scaled conjugate gradients with a maximum of 400 training cycles. They were first initialised using the class means for the left and right class sets (each MLP submodel is always a 2-class classifier). Weight decay regularisation was used to control overfitting.

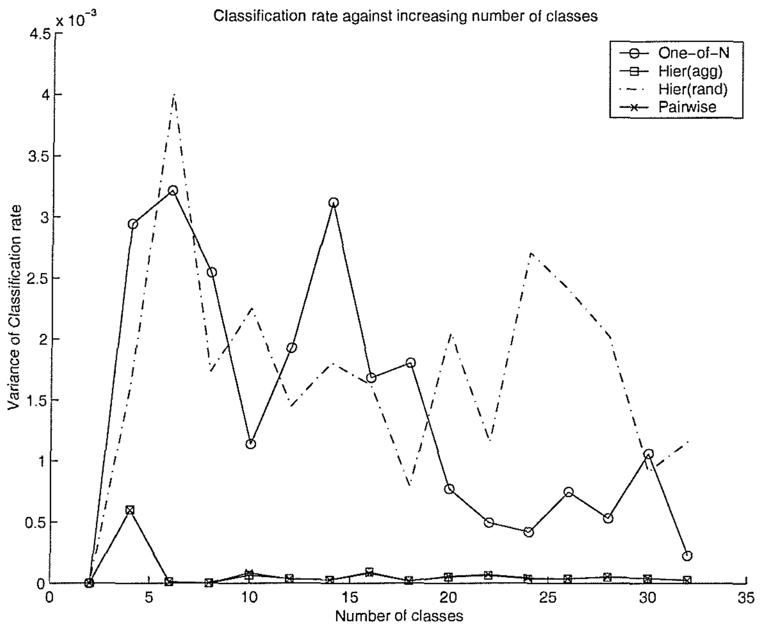
Generalisation classification rates were collected by splitting the datasets into 100 training points and 100 testing points, and classifying the trained models on the unseen testing points. The number of classes was varied between 2 and 32 classes and the whole process was repeated 10 times and averaged for each point on the graph. Graphs are given for the average classification rate and also for the statistical variance of the classification rate as an indication of the statistical significance of the results. This is preferred over the use of error bars as the graphs would be too cluttered using error bar plots. The statistical variance of the classification rate should not be confused with the concept of model variance which is an indication of the flexibility of a model.

7.3.1 Assumptions made and assumptions broken

The logistic linear discriminant classifiers have the underlying assumption that the two classes being discriminated have Gaussian distributions with equal spherical covariance matrices (see Section 2.7). Using the simulated data, this is the case for each individual class, but it must be remembered that discrimination is being done between class subsets that are superpositions of spherical Gaussians. In the hierarchical and one-of-n cases the assumptions for linear discrimination are being broken. In the case where each class is assumed to be normally distributed, using a Bayes plug-in classifier with Gaussian distributions, no assumptions are being broken for the pairwise and one-of-n classifiers. The hierarchical classifier is expected to perform less well due to the Gaussian assumptions being broken. This is readily explained in Section 4.7.1. The MLP makes no explicit assumptions on the form of the class-conditional distributions, but its flexibility is affected by the number of hidden nodes.

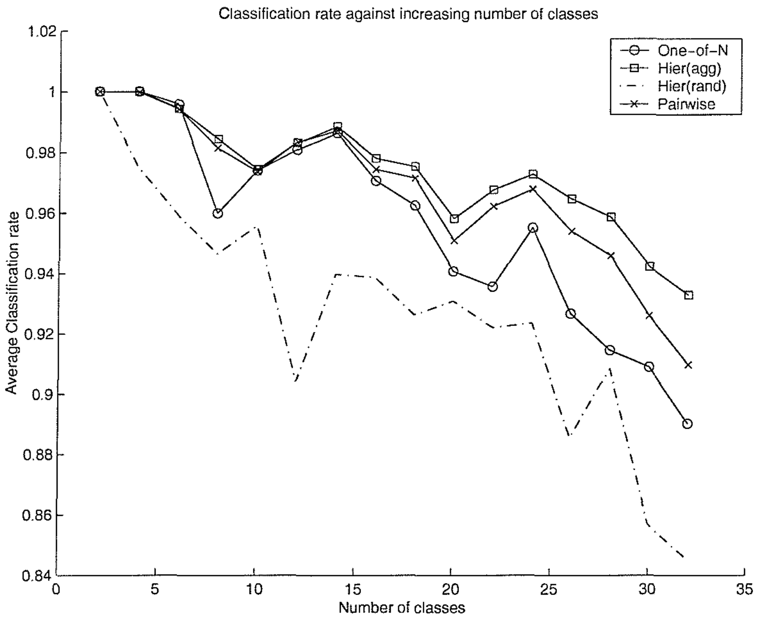


(a)

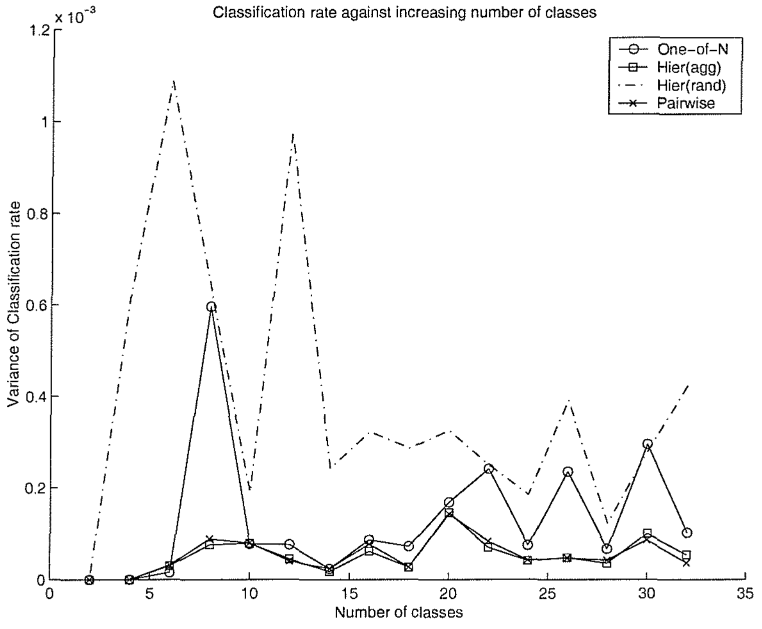


(b)

Figure 7.6: (a) Average classification rates over 10 runs for linear submodels on 2-dimensional Gaussian data distributions. The one-of-n classifier is shown to perform badly while the hierarchical and pairwise classifiers classifiers show consistently good results. Even when using a random class hierarchy the hierarchical classifier outperforms the one-of-n decomposition. (b) Variance of the classification rates.

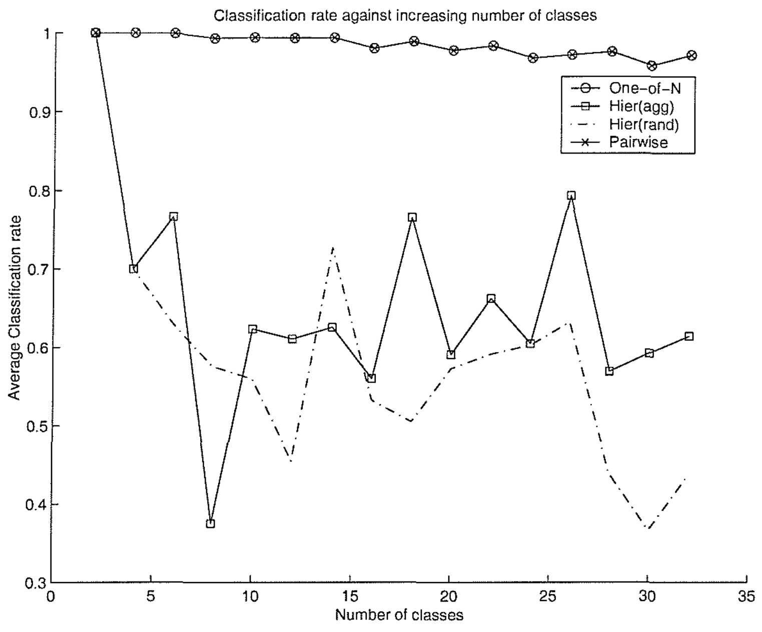


(a)

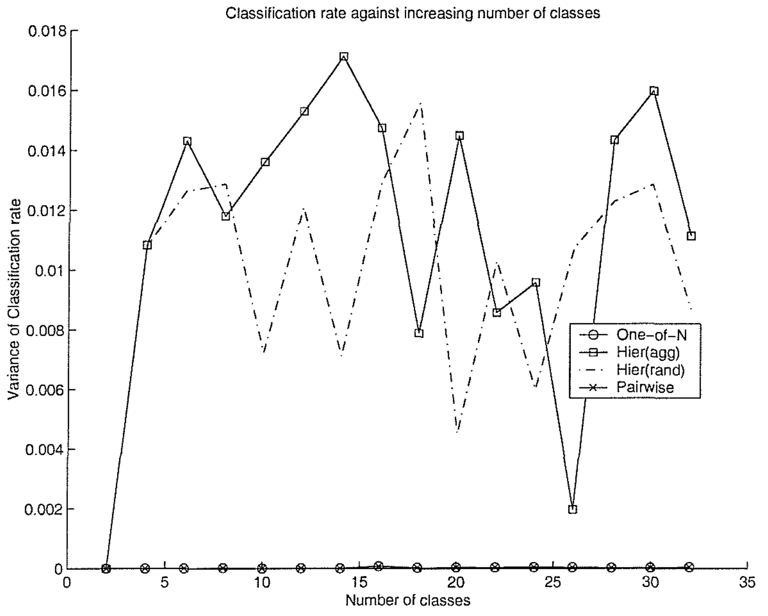


(b)

Figure 7.7: (a) Average classification rates over 10 runs for MLP submodels on 2-dimensional Gaussian data distributions. The classifiers all show good performance, however a clearly identified ordering is shown as the number of classes increase, showing the hierarchical and pairwise classifiers are consistently more accurate. (b) Variance of the classification rates.



(a)



(b)

Figure 7.8: (a) Average classification rates for Gaussian submodels on 2-dimensional Gaussian data distributions. The pairwise and one-of-n classifiers show equivalent results while the hierarchical classifier suffers considerably. This is explained by the fact that the one-of-n and pairwise classifiers are not breaking any underlying assumptions and use the correct density models. At each node the hierarchical classifier is approximating a mixture of Gaussians with a single Gaussian density model. (b) Variance of the classification rates.

7.3.2 Results

The graphs shown trends as expected. For two classes, all three models have equal performance, and as the number of classes increase the performance inevitably drops for all models as the classes overlap and become inseparable.

When using logistic linear discriminants as the form of the submodels for each class decomposition (see Figure 7.6a) the hierarchical and pairwise classifiers perform well. The classification rate is close to 1.0 and hardly decreases for large numbers of classes. This is in the order of the optimum performance for the particular simulated datasets. However the performance of the one-of-n model degrades drastically as the number of classes increases. This is as predicted since the one-of-n classifier is less likely to be able to separate many of the datasets separable by the hierarchical classifier. Even the hierarchical classifier using a random tree outperforms the one-of-n classifier. The variance of the classification rates over the 10 iterations (see Figure 7.6b) is small showing that the average classification rates shown are a good estimate of the true average classification rate. The statistical variance of the results for the one-of-n and random hierarchy are higher, showing that they are less apt to finding a consistent solution to the problem.

When using MLPs as the submodels for each class decomposition, all the classifiers perform well (over 0.9 classification rate for all but the greatest number of classes, see Figure 7.7a). However a distinct ordering in the performance is shown. The hierarchical classifier using agglomerative clustering consistently outperforms the pairwise classifier which outperforms the one-of-n classifier which again outperforms the random hierarchical classifier. This trend is also reflected in the statistical variance of the classification rate (see Figure 7.7b) where the variance is small for the hierarchical and pairwise classifiers but increases for the one-of-n and random hierarchical classifiers respectively.

The process behind this ordering may be explained both by the complexity of the subproblems in each class decomposition classifier and by the method used to train the MLPs. As argued in Chapter 4, the one-of-n classifier introduces complexity into a multicategory classification problem that increases with the number of classes. Although the MLP can represent problems of varying degress of complexity, it does have a limit to the complexity it can model which is defined by the number of hidden nodes. Thus the one-of-n classifier is expected to suffer using fixed complexity models for an increasing number of classes. The pairwise classifier has constant complexity for an increasing number of classes, and the hierarchical classifier controls the complexity by using the agglomerative clustering procedure. This explains the poor performance by the one-of-n classifier and the random hierarchical classifier. The pairwise classifier might be suffering from over-fitting due to the overcomplexity offered by the 5 hidden nodes. This might explain why the hierarchical classifier performs the best of all.

The training method may have an effect since the initialisation is done using the means of the left and right class sets. Then scaled conjugate gradients was used for a fixed number of iterations to adapt the initial parameters to the data. This method could be said to favour the hierarchical (using agglomerative clustering) and pairwise classifiers since the initialisation of the weights is more appropriate for these models. The initialisation might not favour the one-of-n or random hierarchical classifier, but the alternative is a random initialisation which aids none of the models.

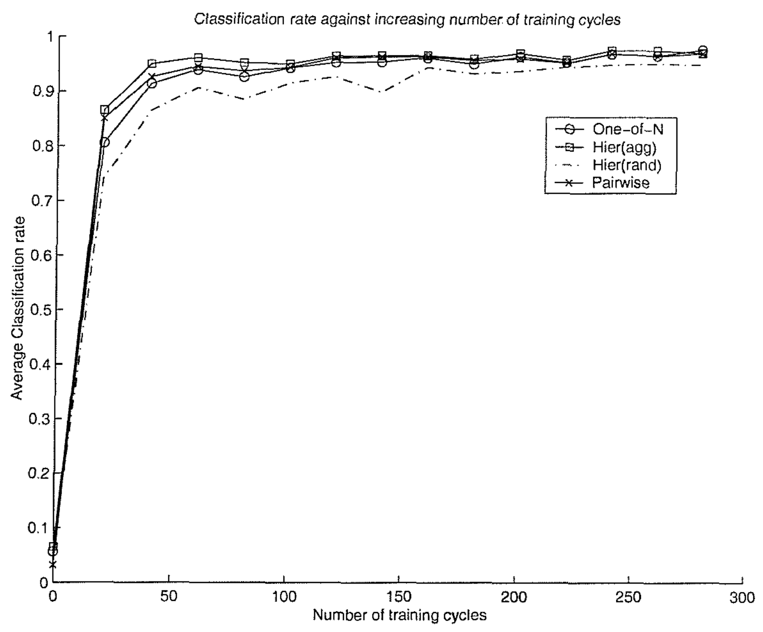
In the Gaussian case, the graph again matches the theory perfectly as shown in Figure 7.8a. The pairwise and one-of-n classifiers have the same performance, shown to be equivalent in theory, and the hierarchical classifier performs worse as predicted. Again, the hierarchical classifier would also be equivalent if a Gaussian mixture model was used. It should be noted that the performance achieved by the one-of-n and pairwise classifiers is optimal for the problem where the class points are normally distributed. The statistical variance of the the classification rate is shown in Figure 7.8b and this shows that the pairwise and one-of-n classifiers show much more consistent results than the hierarchical models, as expected.

7.4 Training complexity for each decomposition

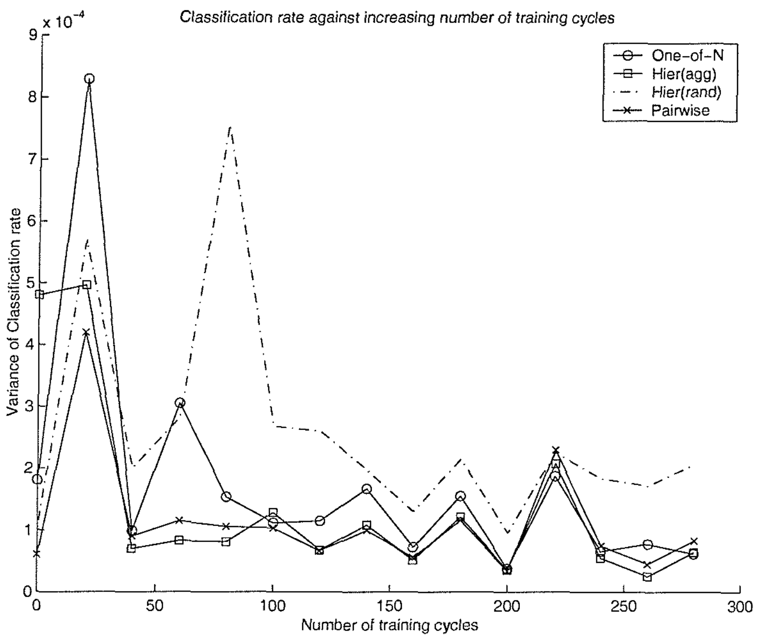
Although the results in the previous section show that the pairwise and hierarchical classifiers tend to outperform the one-of-n classifier for an increasing number of classes, in practise it is usually the case that the problem size is fixed and the most important issue is the complexity of training each classifier. This section describes an experiment to evaluate the training complexity of each decomposition classifier in terms of the number of training cycles needed when using MLP submodels. The MLPs are trained as in the previous section, but the number of classes is fixed at 20 and the number of training cycles is varied from 0 to 280.

The MLP submodels were initialised at first with random parameters, and secondly using the parameters from a linear discriminant which has been initialised using the means of the left and right class subsets for each submodel (see Section 4.11 for details). The results are shown in Figures 7.9 and 7.10.

It can be seen that for MLPs with randomly initialised weights that the classification rates are very low for 0 training cycles as expected, and increase dramatically with

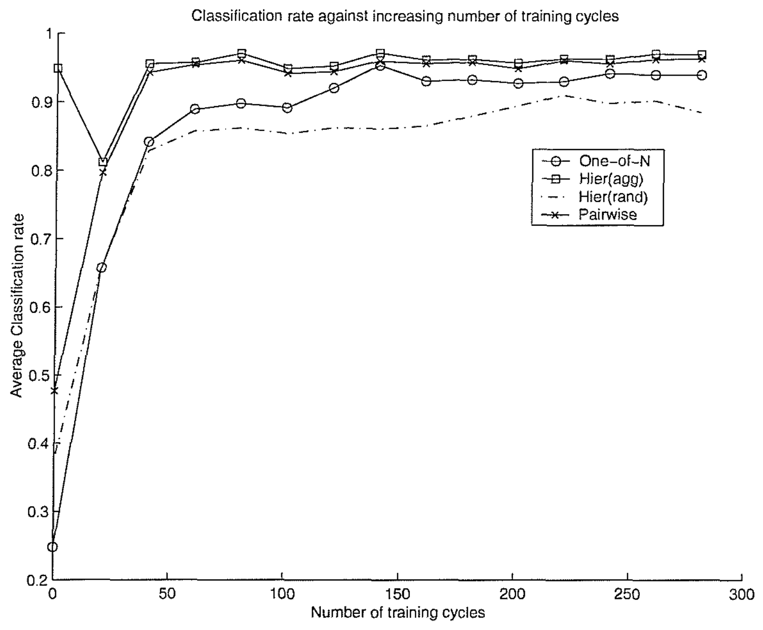


(a)

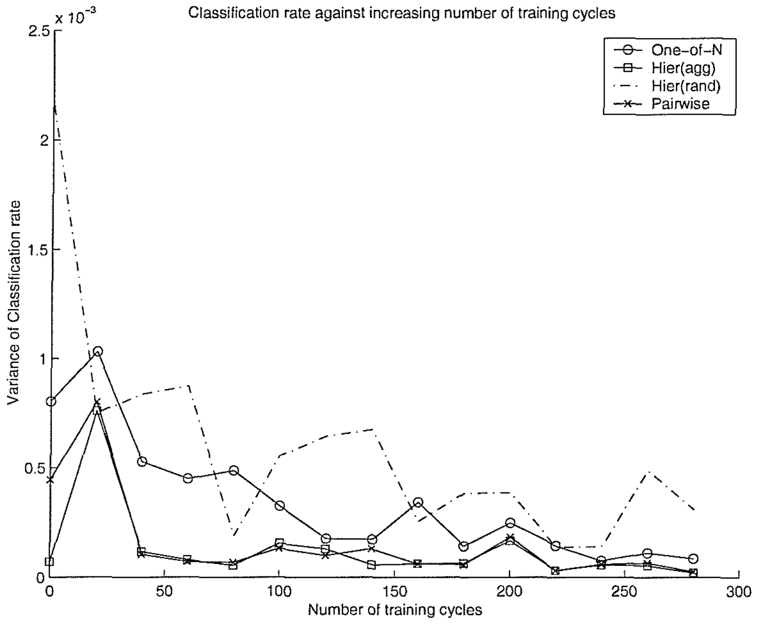


(b)

Figure 7.9: (a) Average classification rates for MLP submodels on 2-dimensional Gaussian data distributions on 20 classes. The number of training cycles was varied. The MLP parameters were initialised randomly. (b) Variance of the classification rates.



(a)



(b)

Figure 7.10: (a) Average classification rates for MLP submodels on 2-dimensional Gaussian data distributions on 20 classes. The number of training cycles was varied. The MLP parameters were initialised using linear discriminant models. (b) Variance of the classification rates.

more training cycles. The classification rates level off at their peak values for large numbers of training cycles. In the case for MLPs initialised from linear discriminants, a different picture can be seen, where the hierarchical and pairwise classifiers have a better classification rate with 0 cycles due to the initialisation, however, after 50 or so training cycles there is little advatange gained from using this type of initialisation. It is important to note however that the hierarchical classifier consistently gives rise to the best classification rate, followed by the pairwise classifier and then the one-of-*n* classifier. The next simulated experiment shows that the hierarchical classifier has a good potential for better classification rate using feature selection.

7.5 Feature selection on *k*-class problems

It has been shown in Section 4.7.1 that when using linear discriminants the hierarchical decomposition can represent a greater number of problems and therefore perform better on datasets with increasing numbers of classes. However it may be the case that as well as many classes, there exist many possible inputs to a problem, and using all the features is not always desirable, as shown by Hughes' phenomenon (Hughes, 1968). It will be shown that using a hierarchical classifier in such a case allows models of lower dimension to be used where each model uses a small, but different feature set, as opposed to a one-of-*n* decomposition that usually results in each model using the same feature set.

Data was generated as described in Section 7.2.2 above. Each dataset was classified using a Bayes plug-in classifier with Gaussian distributions, and linear classifiers using both the one-of-*n* and hierarchical decompositions. Feature selection was used on each of the linear models in the hierarchical decomposition to select one good discriminant feature if possible, and if not to select all the features.

In this case the average number of features for each model of the hierarchy is considerably less than both the one-of-*n* classifier and the Bayes classifier, which have to use all the features all the time. The hierarchical classifier can take advantage of the fact that there is one discriminative feature that will separate two class subsets, if chosen properly. This effect is shown clearly in Figure 7.11.

It should be noted that the class hierarchy, found using an agglomerative design procedure, does not always produce the correct divisions to allow the single discriminative feature to be used. In this case all the features are used for the errant node in the hierarchy, thus not degrading classification performance though not benefitting from the effects of feature selection. Nevertheless, the average number of features per node

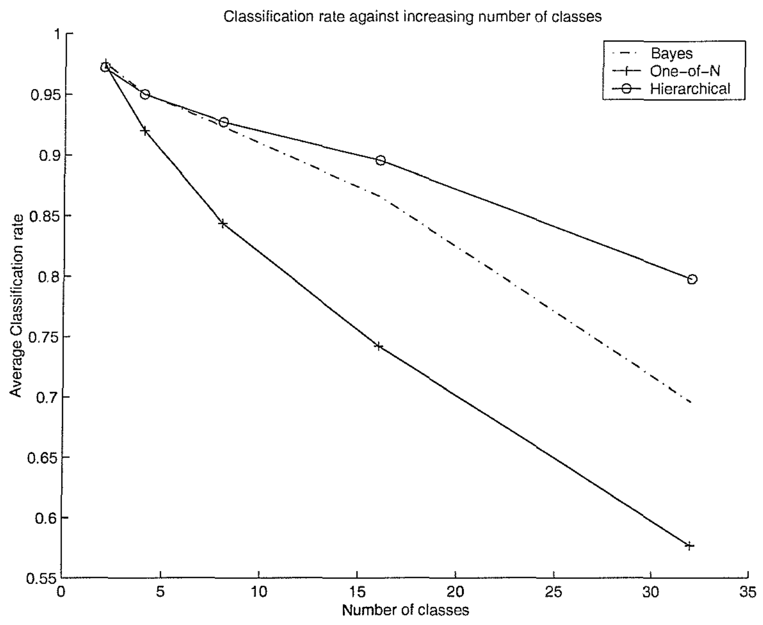


Figure 7.11: Classification rate using linear submodels and feature selection. The one-of-n and hierarchical classifiers select either one or all features for each submodel. The Bayes classifier is used as a benchmark using all the features.

is still much lower than the other algorithms, resulting in a considerably better average classification performance.

This is an important result which lies at the heart of the motivation for the hierarchical classifier. The aim is to reduce the average number of features per node in problems of many classes in high dimensions. The analysis on simulated datasets in this chapter shows that the necessary trends in performance can be shown for a hierarchical classifier to perform well under these conditions. In the next section similar results are shown for all three decompositions using real-world data.

7.6 Real-world data

Although the simulated datasets in the previous section are useful in demonstrating properties of the different decomposition methods, results on real-world data are needed to give a more reasonable empirical evaluation. The datasets used to investigate the related performance of each algorithm are the MNIST and MFEAT handwriting recognition datasets described below. To illustrate the added understanding provided by the hierarchical classifier the FLIERS dataset is used.

Automatic handwriting recognition has been a challenging topic since the introduction of digital computers in the 50's. Today it is a well established field with many successful applications (Plamondon and Srihari, 2000). The interest in handwriting recognition in this thesis is for the purpose of demonstrating the techniques presented in the previous chapters.

Handwriting recognition is categorised by the nature of the input and the form of the handwriting to be recognised. The input method may be *on-line*, or *off-line*. In the on-line case the handwriting is input using a electronic pointing device on an active surface, such as liquid crystal display which electronically mimics the ink from a pen. A series of 2-dimensional data points are collected as a time-series as the electronic pen moves. In the off-line case, no time-ordering information is available since the handwritten script is written to paper using traditional pen or pencil and then digitally scanned to produce a raster-image.

The nature of the script may be free-flowing cursive script, or separately printed characters. The nature of the recognition may be at the level of individual characters, or words, or even recognition of the type of script (e.g. latin or arabic), or the identity of the writer.

In this thesis off-line handprinted character recognition has been chosen for the following reasons:

- It is a sufficiently hard, yet well researched area, suitable for the demonstration of techniques in this thesis.

- The availability of standard datasets for comparison with other work.

There are also two general approaches to handwriting recognition, namely *statistical* and *structural* methods. The statistical methods use the low-level pixel data and statistics derived from these as features for classification (Plamondon and Srihari, 2000). Little prior knowledge is incorporated, but feature extractors have known invariance properties that may be useful to know in advance. The structural methods on the other hand use prior knowledge to extract information on high-level features such as lines and loops in the image (Marcelli, Likhareva and Pavlidis, 1997). Much effort goes into the design of the feature extractor, but with the advantage that the features are related to meaningful concepts in the problem domain.

The statistical approach is by far the more popular approach used in handwriting recognition at present. Structural approaches tend to suffer from the additional effort to design, implement and fine-tune many application-specific feature extractors, and from the problem of robustness from noise. For these reasons, and due to their open availability on the internet, either the raw data or non-structural feature sets are used in this thesis.

7.6.1 MNist handprinted digit data

The first dataset is the modified NIST dataset, constructed by Yann Le-Cun at AT&T labs. This was created from a larger dataset compiled by NIST. The dataset is available at the following internet web address: `http://www.research.att.com/~yann/ocr/mnist/`. Example digits from the dataset are shown in Figure 7.12.

The modifications that were made to the original NIST database are described below, these details are taken from the above web-address.

"The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. The images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.

"The MNIST database was constructed from NIST's Special Database 3 and Special Database 1 which contain binary images of handwritten digits. NIST originally designated SD-3 as their training set and SD-1 as their test set. However, SD-3 is much cleaner and easier to recognize than SD-1. The reason for this can be found on the fact that SD-3 was collected among Census Bureau employees, while SD-1 was collected among high-school students. Drawing sensible conclusions from learning experiments requires that the result be independent of the choice of training set and test among the complete set of samples. Therefore it was necessary to build a new database by mixing NIST's datasets.

"The MNIST training set is composed of 30,000 patterns from SD-3 and 30,000 patterns from SD-1. Our test set was composed of 5,000 patterns from SD-3 and 5,000 patterns from SD-1. The 60,000 pattern training set contained examples from approximately 250 writers. The sets of writers of the training set and test set are disjoint."

Using a one-of-n, hierarchical and pairwise classifier with linear nodes the classification results were achieved by training on the first 20000 training points and testing on all 10000 test points.

These results, given in table 7.2, using combinations of linear classifiers on the raw data, and without the use of prior knowledge specific to handwriting recognition,

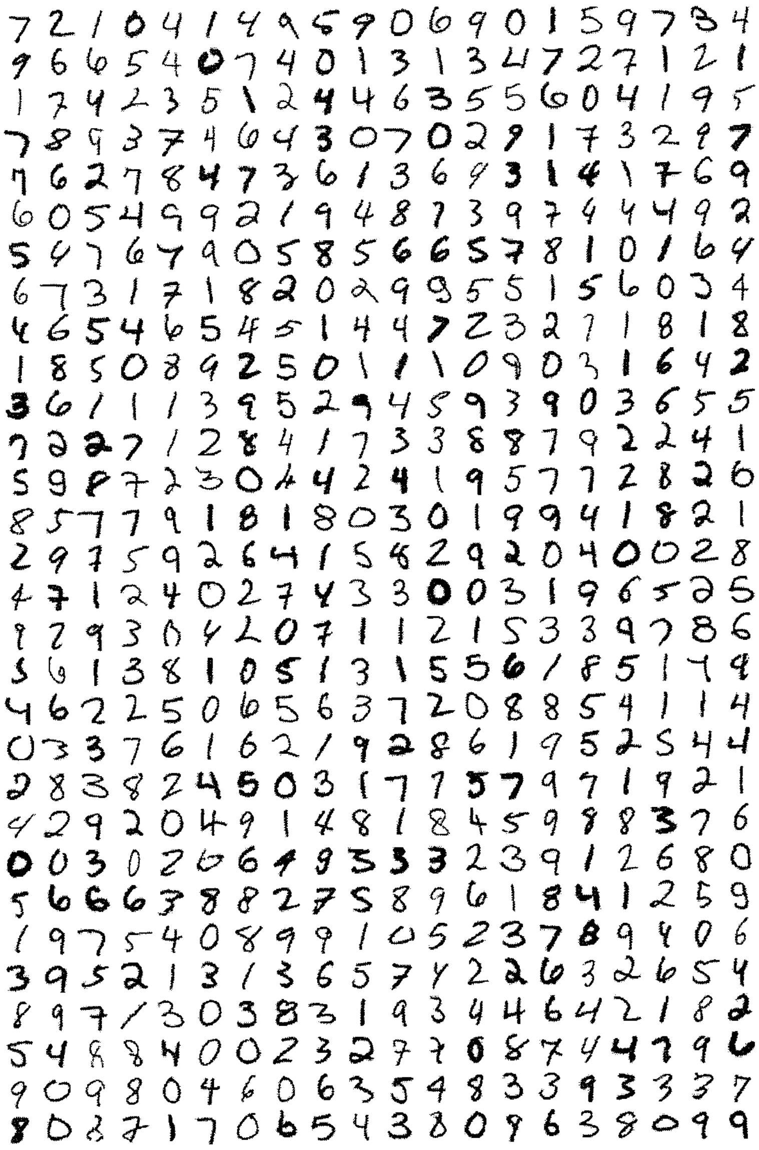


Figure 7.12: Examples of handwritten digits (MNIST dataset)

	Training data	Test data	% Test Error
One-of-n	0.90815	0.9031	9.69
Pairwise	0.9409	0.9245	7.55
Hierarchical	0.9164	0.9051	9.49

Table 7.2: Classification rates for classifier on raw MNIST data

correlate with the results published on the Image Processing Research Department website at AT&T which are repeated here for convenience in Table 7.3.

It must be stressed that the algorithms in this thesis are not competing with the error rates given in this table. The table is merely intended as a guideline for results given, where they are comparable (i.e. linear models compared with linear models). No deskewing techniques (where slanted digits are set upright), or distortion techniques (the distortions are random combinations of shifts, scaling, skewing, and compression) are used to improve the classification performance. It is fair to compare the error rates for the 2-layer Neural Networks (NNs), since the linear models are not much worse than many of these. In addition, the complexity of a NN with 1000 hidden units is a very unwieldy model to have to train and sheds little interpretation on the problem. The author feels that the techniques presented in this thesis are much more interpretable for a multi-class problem than using such a large single model. The LeNet quoted in the table is a model specifically designed for handwritten character recognition that uses edge filters and smoothing operators developed by the group that created the dataset, and it is likely that their algorithm has been fine-tuned and optimised to this particular dataset.

7.6.2 Comparison of local and global feature selection

To assess the impact of local and global feature selection the following experiment was conducted using the raw MNIST data.

Again 20000 training points and 10000 test points were taken, and for each decomposition, the classifier was trained using a selection of p features, both globally and locally. The fraction p/n was varied where n is the total number of features (in this case n is 400 since we are working with a 20 by 20 pixel image). Features were selected by choosing the best p features according to the class-dependent overlap metric given in Section 6.3. The results are presented in Figures 7.13, 7.14, 7.15, and 7.16.

It can be clearly seen that, for each decomposition, using local feature selection results in higher classification rates for a smaller number of features selected per sub-model. When using all the features the local and global paradigms become equivalent, which is shown in these results. Since linear models tend to suffer very little from overfitting the classification rate increases with the number of features used. However, there are advantages of using models with fewer features asides from the reduction of overfitting since models of fewer features are simpler, quicker to train and the cost associated with evaluating the features is lower.

When comparing the three decompositions using local feature selection in Figure

METHOD	TEST ERROR RATE (%)
linear classifier (1-layer NN)	12.0
linear classifier (1-layer NN) [deskewing]	8.4
pairwise linear classifier	7.6
K-nearest-neighbors, Euclidean	5.0
K-nearest-neighbors, Euclidean, deskewed	2.4
40 PCA + quadratic classifier	3.3
1000 RBF + linear classifier	3.6
K-NN, Tangent Distance, 16x16	1.1
SVM deg 4 polynomial	1.1
Reduced Set SVM deg 5 polynomial	1.0
Virtual SVM deg 9 poly [distortions]	0.8
2-layer NN, 300 hidden units	4.7
2-layer NN, 300 HU, [distortions]	3.6
2-layer NN, 300 HU, [deskewing]	1.6
2-layer NN, 1000 hidden units	4.5
2-layer NN, 1000 HU, [distortions]	3.8
3-layer NN, 300+100 hidden units	3.05
3-layer NN, 300+100 HU [distortions]	2.5
3-layer NN, 500+150 hidden units	2.95
3-layer NN, 500+150 HU [distortions]	2.45
LeNet-1 [with 16x16 input]	1.7
LeNet-4	1.1
LeNet-4 with K-NN instead of last layer	1.1
LeNet-4 with local learning instead of ll	1.1
LeNet-5, [no distortions]	0.95
LeNet-5, [huge distortions]	0.85
LeNet-5, [distortions]	0.8
Boosted LeNet-4, [distortions]	0.7

Table 7.3: Reported performance using MNIST dataset

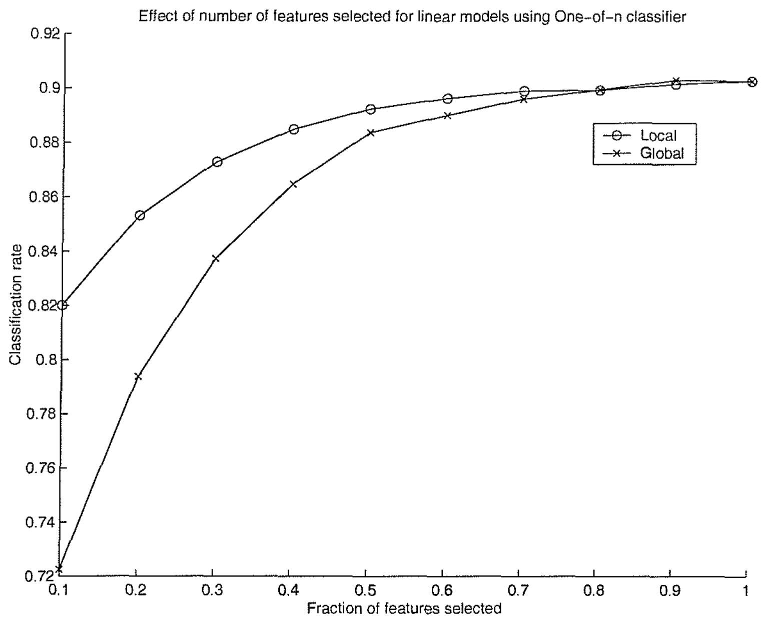


Figure 7.13: Classification rate using global and local feature selection using linear submodels with a one-of-n classifier on the MNIST data

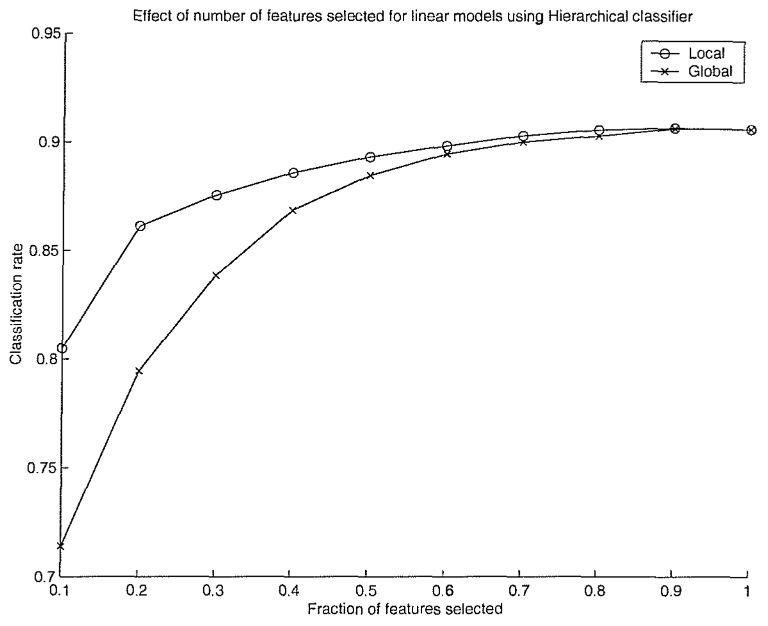


Figure 7.14: Classification rate using global and local feature selection using linear submodels with a hierarchical classifier on the MNIST data

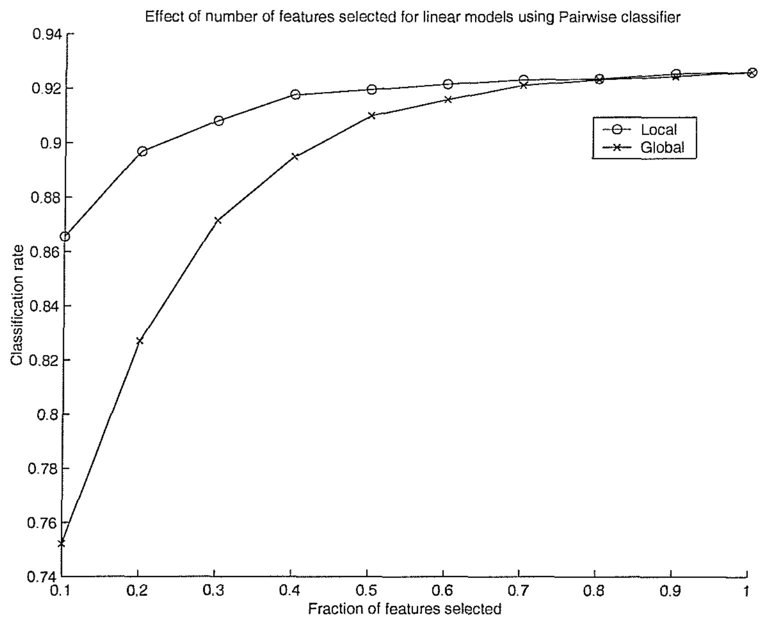


Figure 7.15: Classification rate using global and local feature selection using linear submodels with a pairwise classifier on the MNIST data

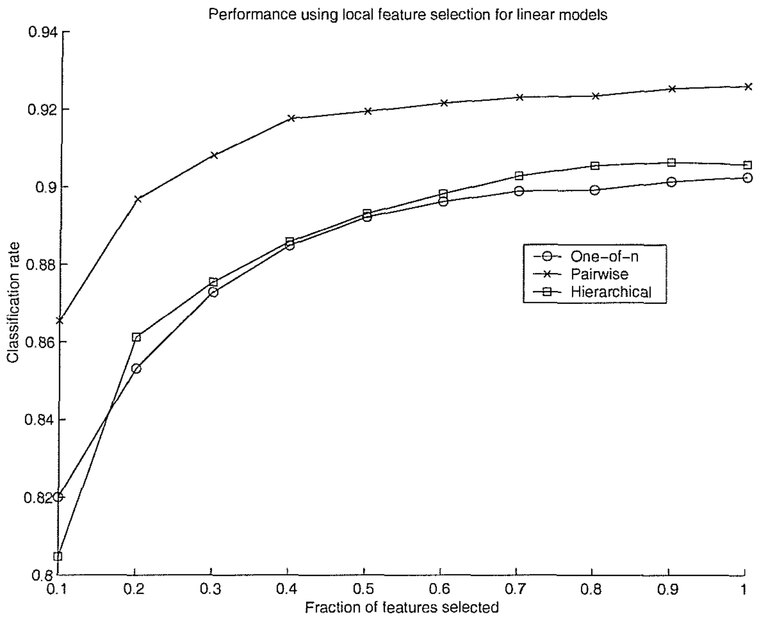


Figure 7.16: Classification rate using local feature selection for all decompositions using linear submodels on the MNIST data

7.16 it is shown that the one-of-n classifier consistently has the lowest classification rate when using linear models.

7.7 The MFEAT dataset

The MFEAT dataset is again a handwritten digit recognition dataset. It is used in (Jain, Duin and Mao, 2000) as a comparison of classification algorithms. The dataset consists of 2000 handprinted digits (200 per class) and the correct class labels. The raw image data is not available, but six feature sets are available which have been extracted from the raw data. These are as follows:

- Feature set (1): 76 Fourier coefficients of the character shapes,
- Feature set (2): 216 profile correlations,
- Feature set (3): 64 Karhunen-Loève coefficients,
- Feature set (4): 240 pixel averages in 2×3 windows,
- Feature set (5): 47 Zernike moments, and
- Feature set (6): 6 morphological features.

The feature sets 2, 3, and 4 were used in the experiments in this thesis since they represent both pixel-level information (pixel averages), image-level information (Karhunen-Loève coefficients) and shape information (profile correlations). The Karhunen-Loève transformation is also known as *principle component analysis* (Jollife, 1986) where the component vectors in input space (the eigenvectors of the covariance matrix) that represent the most variation in the data are found, and the data is transformed into a lower dimensional space by projecting onto the component vectors, removing the components with low variance that are usually attributed to noise. Of interest, Zernike moments are described in (Mukandan and Ramakrishnan, 1998), they represent moments of 2-dimensional images using a polar co-ordinate transform. Details of this dataset is given in (van Breukelen, Duin, Tax and den Hartog, 1998).

7.8 Comparison of local and global feature selection using MLPs

An experiment similar to that in Section 7.6.2 was carried out. The first 100 points per class were used for training and the remaining 100 points per class were used as test points, and for each decomposition, the classifier was trained using a selection of p features, both globally and locally. The fraction p/n was varied where n is the total number of features. In this case there were 520 features. Features were selected by choosing the best p features according to the class-dependent overlap metric given in Section 6.3.

The MLP models had one layer of input nodes, one layer of 15 hidden nodes and one output node using a sigmoid activation function. Weight decay regularisation and scaled conjugate gradients were used to train the models. Unfortunately due to the slow training time for MLPs, each point on the graphs represents a single classification value, and not an average of many values as before. This results in some variance between the points, although the desired trends are more significant than this variation and the graphs still serve as a reliable demonstration.

The results are presented in Figures 7.17, 7.18, 7.19, and 7.20.

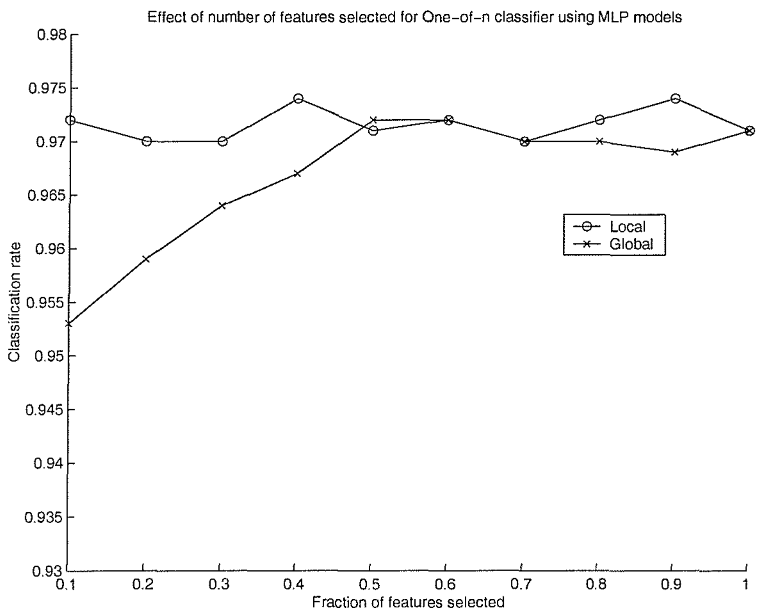


Figure 7.17: Classification rate using global and local feature selection using MLP submodels with a one-of-n classifier on the MFEAT data

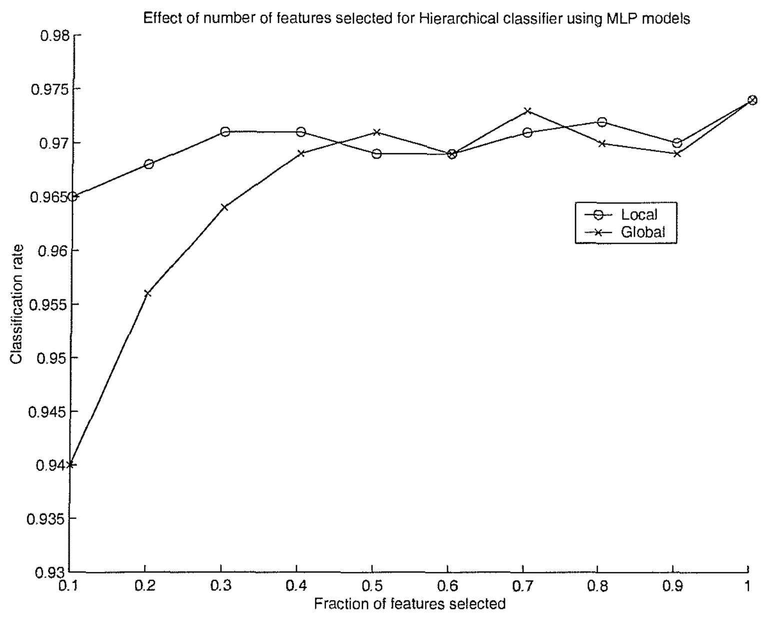


Figure 7.18: Classification rate using global and local feature selection using MLP submodels with a hierarchical classifier on the MFEAT data

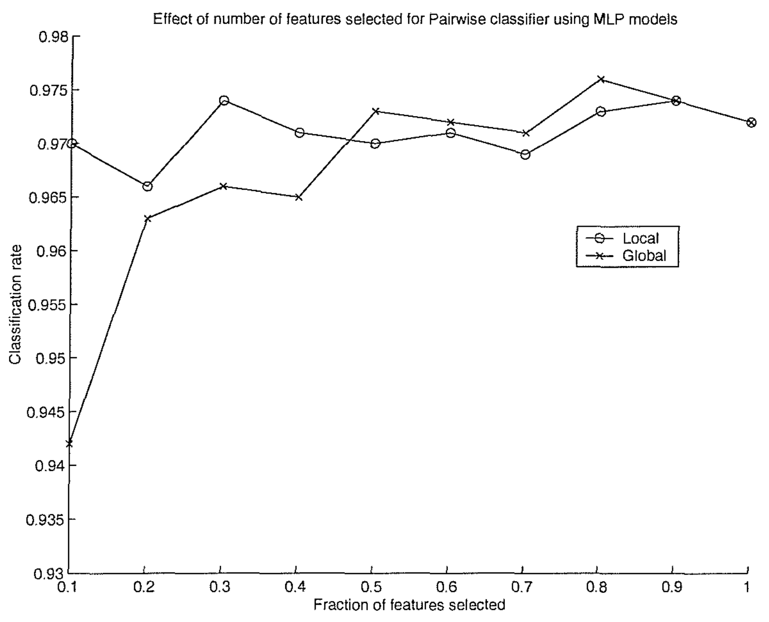


Figure 7.19: Classification rate using global and local feature selection using MLP submodels with a pairwise classifier on the MFEAT data

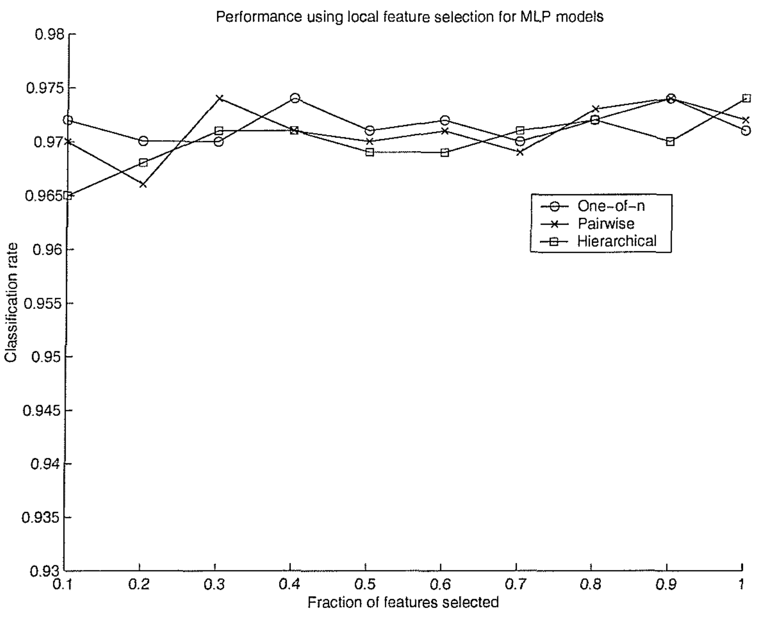


Figure 7.20: Classification rate using local feature selection using MLP submodels on the MFEAT data

Again the classification rate is shown to be consistently better for a low fraction of features selected. Due to the regularisation used on the MLP models, there is little likelihood of overfitting and using all the features did not result in poor classification results. In fact the graph in Figure 7.20 shows that using local feature selection and regularised MLP models gives a consistent classification rate across large and small fractions of features selected.

7.9 FLIERS data

The FLIERS (Fuzzy Land Information in Environmental Remote Sensing) large-scale agricultural dataset (Hughes, Bastin and Fisher, 1998) is data collected from a remote sensing satellite for use in land cover classification. The digital image has a resolution of 0.5m, each pixel represented by six different spectral measurements (6 real-valued inputs). There are 27 classes which have been selected by field work on the actual land photographed. Class mixture coefficients are given in the original dataset. Only the most dominant class for each pixel was used here.

Several subgroups of classes have been suggested through expert knowledge of the land cover. The methods used in this thesis were used to find a class hierarchy.

Similar class subsets
Wheat, Oats, Barley
Grass, Cut silage
Concrete, Buildings
Lone deciduous tree, Woodland
Maize, Potatoes, Beet, Re-seeded grass
Shrub, Young Hedge

Table 7.4: Expert subgroupings

This is then compared with the expert groupings.

The fourth and final class of experiment is simply to illustrate that the class hierarchy in a hierarchical classifier can aid problem understanding. A class hierarchy was found automatically using the agglomerative clustering procedure and then this is compared with the class groupings proposed by a team of experts. This experiment is qualitative and is intended as a simple illustration that the class hierarchy can represent the underlying problem.

The classes present in the FLIERS dataset are detailed in Table 7.6. The FLIERS team have previously grouped several classes together by similarity according to observations made from field work and expert knowledge of land cover types. A class hierarchy was generated directly from the FLIERS data using agglomertive hierarchical clustering without any prior knowledge. The results shown by the hierarchy in Figure 7.21 demonstrate a good correlation between the expert knowledge and the data driven approach.

The class subsets suggested by the experts is shown in table 7.4. The class subsets suggested by the algorithm (amongst others) is shown in table 7.5. This shows that the subsets derived from finding the class hierarchy correlate well with the subsets defined by the field experts. In this case, in the absence of field work it might be reasonable to use the groupings found through the automatic method as a basis for understanding the nature of the data.

7.10 Conclusions

It has been shown empirically that the hierarchical and pairwise models have a strong advantage over the one-of-n decomposition when using linear models or MLPs. This is in accordance with the theory in Chapter 4.

Similar class subsets
Wheat, Oats, Barley
Grass, Cut silage
Maize, Potatoes, Beet, Re-seeded grass
Shrub, Lone deciduous tree, Linseed, Tall Herb, Young Hedge

Table 7.5: Algorithm subgroupings

Class label	Land cover type	Class label	Land cover type
1	Wheat	14	Cut silage
2	Maize	15	Water
3	Oats	16	Buildings
4	Barley	17	Asphalt
5	Rape	18	Gravel
6	Set-aside	19	Concrete
7	Linseed	20	Deciduous woodland
8	Broad beans	21	Shrub
9	Potatoes	22	Tall herb
10	Beet	23	Coniferous woodland
11	Grass	24	Lone deciduous tree
12	Re-seeded grass	25	Mature hedge
13	Uncut silage	26	Young hedge

Table 7.6: Class labels for FLIERS data

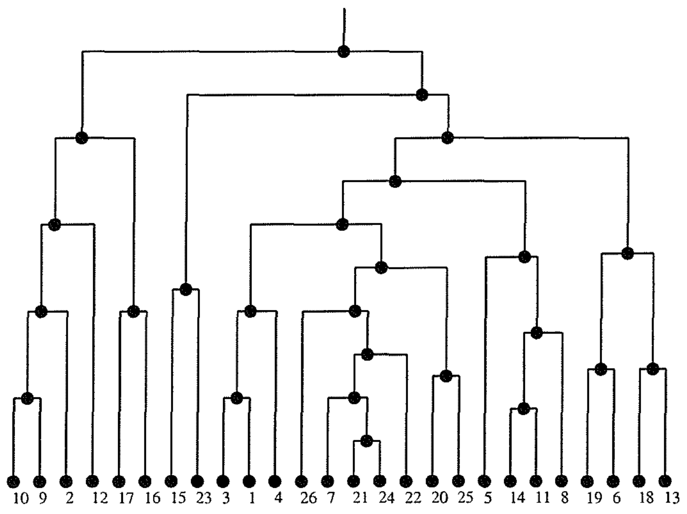


Figure 7.21: Class hierarchy for FLIERS data

It has also been shown that for both linear and non-linear models (MLPs) that local feature selection results in a higher classification rate than global feature selection. This demonstrates strong evidence that features are indeed class-dependent and that an advantage can be gained through exploiting the class-dependent nature of these features. The class decomposition paradigm is shown to be a suitable method for using class-dependent features. It should be noted that using local feature selection will mean the total number of features evaluated by the classifier will tend to be higher than using global feature selection, but this can be counteracted by using computational cut-off for the hierarchical decomposition.

The results presented in this chapter reflect the predictions made via the theory presented in Chapter 4 on multicategory classification and Chapter 6 on class-dependent features. This is a strong confirmation of the underlying theory and that the fields of multicategory classification and class-dependent features are worthy of investigation and further research. Conclusions and key ideas for future work in this area are presented in the final chapter.

Chapter 8

Conclusions

In this thesis the two main themes of multicategory classification and class-dependent features have been investigated within the field of statistical pattern recognition. To address the diversity of previous research in multicategory classification, a new taxonomy of multicategory classification was presented that brings together many popular multicategory classification algorithms using a probabilistic framework. Equivalences were shown between different algorithms under certain assumptions and then the differences discussed for real applications. The algorithms were investigated in terms of their scalability in the number of classes.

Using the framework presented, a subset of algorithms was derived that decompose a many-class problem into submodels by partitioning the set of class concepts rather than partitioning the input space directly. This then allowed class-dependent features to be analysed in terms of subsets of classes and the correlation between features and classes within defined subsets. This leads to the idea of global and local feature selection which has been shown to be beneficial when using real-world data. These algorithms were also investigated in terms of their ability to aid the understanding of the underlying multiclass problem. These issues are discussed further in this concluding section and avenues for future work are presented.

8.1 Multicategory classification

The problems identified with previous research in multicategory classification were that there was a diversity of techniques. Little research exists in the inherent multicategory nature of classification algorithms such as k-nearest neighbours, decision tree classifiers, pairwise discriminants, and multicategory formulations of generalised linear discriminants. Much research has concentrated on the 2-class problem, but the issues behind generalising this research to multicategory problems is often overlooked.

Of the existing inherent multicategory classification algorithms, decision tree classifiers and pairwise discriminants were shown to vary considerably in the details of their design, and in part heuristics were used to formulate the algorithms. Classification algorithms that use a one-of-n output encoding were shown to add complexity to the multicategory problem unnecessarily as the number of classes increased.

In response to these issues a new probabilistic framework was presented in Chapter 4 through which many popular multicategory classification algorithms can be derived. This was achieved by making the conditioning on the input space, R, and set of classes, Ω, explicit in the formulation of the posterior probabilites and then choosing to either partition R or Ω. This allowed a distinction to be made between many-class algorithms that can be derived from one or the other viewpoint.

A taxonomy of classification algorithms that clarified the difference between a 2-class classifier and a multicategory classifier was defined, and using the probabilistic framework, many popular multicategory classification algorithms were placed in this taxonomy. Popular classifiers such as the k-nearest neighbour classifier, and decision tree classifiers were derived from the input space decomposition. The class decomposition was shown to derive hierarchical, pairwise and one-of-n algorithms. This clarifies the distinction between different types of multicategory classifiers, especially the two distinct, but similar, hierarchical paradigms which result from either method of decomposition.

By assuming the decomposition of the set of classes, a principled approach to the combination of 2-class classifiers was given and the hierarchical, pairwise and one-of-n decompositions were shown to be equivalent when the true class-conditional densities are known. However in reality, models need to be trained on finite data and the issues that arise for the class-decomposition classifiers were also addressed. This is summarised below.

8.1.1 Scalability and efficiency of class-decomposition classifiers

The three types of class set decompositions were compared in terms of their scalability in the number of classes for training and classification. The number of models in each class-decomposition is shown to be $k-1$ for the hierarchical classifier, k for the one-of-n classifier, and $k(k-1)/2$ for the pairwise classifier. The number of points used in training was shown to be in the order of k^2 for the one-of-n and pairwise models, but significantly less at $k\log_2 k$ on average for the hierarchical model. A novel and principled approach to improving the classification complexity of a hierarchical classifier was given.

The hierarchical model is concluded to scale the most efficiently as the number of classes increase, although it suffers from the need to find the class hierarchy. However, the effort spent on identifying the class hierarchy can be reduced if flexible enough models are used for each submodel. These issues were addressed empirically as described in the following section.

8.1.2 Support from experiments on simulated data

The average classification rate for each of the class-decomposition classifiers was evaluated over a range of simulated datasets with an increasing number of classes. The algorithms were shown to behave as expected on the simulated data which had a known statistical structure of Gaussian distributed class points in a 2-dimensional input space.

The one-of-n classifier was shown to degrade in classification performance for both linear and multi-layer perceptron submodels. The difference in classification performance between a hierarchy found using the agglomerative design procedure and a random hierarchy was also shown to be much less significant when using MLP submodels. The pairwise and hierarchical classifiers were shown to perform best using linear models, and the hierarchical classifier resulted in the highest classification rate using MLP submodels for problems of increasing numbers of classes.

Experiments were also carried out to show the effect of parameter initialisation and the number of training cycles needed for the class-decomposition classifiers using MLP submodels. The pairwise and hierarchical models were shown to train slightly faster than the one-of-n decomposition model. Unfortunately a more in-depth comparison with a single multi-output MLP classifier was not undertaken, as the class-decomposition classifiers are expected to train significantly faster.

8.1.3 Future work

After a detailed analysis of multicategory classification, and more specifically class-decomposition classifiers, unexplored avenues for future research are identified.

In a similar vein to the reduction of the computational complexity of a hierarchical classifier during classification, it is known that not all the models in a pairwise classifier need to be evaluated for each classification (Jia and Richards, 1998), and possibly the number of models in training can be minimised. No principled methods for doing so have been presented and this would be a suitable avenue for research given the good performance of a pairwise classifier, and the disadvantage that the number of models increases in the order of k^2.

Additionally, one significant multicategory classification algorithm that was not included in the taxonomy or the probabilistic formulation was the error-correcting output encoding classifier of Dietterich and Bakiri (1995). It is strongly suspected that this technique will fit into the class-decomposition classification paradigm due to the binary nature of the submodels learnt, and the discrimination of classes in their entirety, unlike input space partitioning techniques. Further analysis might shed light on the nature of this algorithm and its place in the taxonomy.

8.2 Class-dependent features

The problems identified in the field of class-dependent features were that although there is strong motivation for the use of class-dependent features in multicategory classification, most notably hierarchical classification, that it is a rarely cited field. Previously researched algorithms do have the ability to select different features for different classes, namely hierarchical classifiers, pairwise classifiers (Jia and Richards, 1998), and the modular neural network of Oh, Lee and Suen (1999), but they have been researched independently and need to be drawn together. A detailed definition of class-dependent feature is lacking; Oh, Lee and Suen (1999) are the only researchers to define and use the term.

In response to these issues a novel definition of class-dependent features, including the distinction between weak and strong class-dependent features is presented in Chapter 6, motivated from examples from real-world data. The class-decomposition paradigm for multicategory classification is shown to provide a strong framework from which class-dependent features can be exploited in a simple and elegant manner. Since the submodels of a class-decomposition classifier operate on specific sets of classes, this allows the class-dependent nature of features to be exploited by simply selecting features locally for each submodel. A metric for measuring the dependence between specific features and the class variable is given, which is defined on the overlap of the probability densities of that feature given two specific sets of classes. Again the distinction between a hierarchical classifier that partitions the input space and a hierarchical classifier that partitions the set of classes is important since the class-dependent paradigm applies more succinctly to the latter.

8.2.1 Support from experiments on real-world data

The class-dependent feature metric was used to select features for submodels in the pairwise, hierarchical, and one-of-n multicategory class-decomposition classifiers in Chapter 7.

Consistently across all types of class-decomposition classifier, and for both linear and non-linear models, the local feature selection method resulted in a higher classification rate when compared to a global feature selection method using the same number of features at each submodel. When using multi-layer perceptron submodels, classifiers using local feature selection showed little sign of the degradation of classification performance even for low numbers of features selected per submodel.

8.2.2 Future work

Although the distinction between weak and strong class-dependent features was made, no analysis of strong class-dependent features was attempted. It is expected that for real-world problems, where the number of classes is large, that the likelihood of needing strong class-dependent features will increase. This thesis has laid down the framework for further investigation and future work can build on this to develop algorithms that use the nature of strong class-dependent features to their advantage.

In particular, the number of unique features evaluated by the each complete classifier using local and global feature selection needs to be evaluated. Local feature selection may involve the evaluation of more features in total during classification than a global model. However for the hierarchical decomposition a fraction of the submodels need to be evaluated when using computational cut-off, reducing the number of local features to be evaluated. It is also the case that the number of submodels to be evaluated for the pairwise classifier may be reduced. The combined effects of local feature selection and computational cut-off on the average number of features evaluated requires further analysis. This is important for applications where the evaluation of a feature during classification carries an associated cost, and the total cost must be minimised.

8.3 Interpretability

An issue touched on in this thesis is the interpretability of trained classification models. Models that are said to be interpretable are decision tree classifiers, and fuzzy classifiers due to their ability to be output in the form of rulebases for human interpretation.

A significant problem with this paradigm is that for complex problems of many classes, the rulebase can contain many rules and becomes so unwieldy that the transparency of the model is clouded.

The hierarchical class-decomposition classifier addresses this problem and is described as an interpretable classifier due to:

- fewer models,
- meaningful class groupings,
- association between features and class groupings, and
- class hierarchy diagrams.

This model has an advantage over the other class-decomposition classifiers considered (one-of-n and pairwise) if the understanding of the underlying problem is important. This would in many cases justify any extra effort required to learn the class hierarchy. An important second advantage is that the size of the tree is fixed, and the number of submodels to be interpreted is always $k-1$. When linear models are used and the number of features is reduced at each submodel by class-dependent feature selection then this method is similar to multivariate decision tree classification, but with a fixed number of multivariate rules.

Regardless of the form of the submodels, the class hierarchy can provide an insight into the nature of the multicategory problem. This is demonstrated for a remote sensing application. Although not demonstrated, the lists of features at each node in the hierarchy can also provide a level of interpretability.

8.4 Final conclusions

The work in this thesis has contributed to the field of multicategory classification by bringing together techniques in a global probabilistic framework. This allows the equivalence of different paradigms to be shown. This then leads to a framework whereby class-dependent features may be used via a well understood motivation. The experimental results confirm the predictions made by the theory, showing that indeed this is a consistent and well-motivated paradigm. There is also much scope for future work in both multicategory classification and class-dependent features.

References

Ambros-Ingerson, J., R. Granger, and G. Lynch (1990). Simulation of the paleocortex performs hierarchical clustering. *Science 247*, 1344–1348.

Bailey, A. and C. J. Harris (1999). Using hierarchical classification to exploit context in pattern classification for information fusion. In *Proceedings of the Second International Conference on Information Fusion (FUSION99)*, Sunnyvale, CA, pp. 1196 – 1203. ISIF.

Basseville, M. (1989). Distance measures for signal processing and pattern recognition. *Signal Processing 18*, 349–369.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford: Clarendon Press.

Bishop, C. M. and M. E. Tipping (1998). A hierarchical latent variable model for data visualisation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 20*(3), 281–293.

Burges, C. J. C. (1998). A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery 2*(2), 121–167.

Chan and C. J. Harris (2001). The neurofuzzy SVM algorithm. *Engineering applications of artificial intelligence to appear*.

Cover, T. M. (1965). Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers 14*, 326–334.

Dash, M. and H. Liu (1997). Feature selection for classification. *Intelligent Data Analysis 1*(3).

DeGroot, M. H. (1989). *Probability and Statistics* (2nd ed.). Addison Wesley.

Dietterich, T. G. and G. Bakiri (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research 2*, 263–286.

Dodd, T. J., A. Bailey, and C. J. Harris (1998). A data driven approach to sensor modelling, estimation, tracking and data fusion. In M. Bedworth and J. O'Brien (Eds.), *EuroFusion98*, pp. 103–111.

Ealey, D. (1997). Expanding the vocabulary: Fractal feature space in neural networks. In *Colloquium on prospects for spoken language technology*. IEE.

Friedman, H. P. and J. Rubin (1967). On some invariant criteria for grouping data. *Journal of the American Statistical Association 62*, 1159–1178.

Friedman, J. H. (1977). A recursive partitioning decision rule for non-parametric classification. *IEEE Transactions on Computers 26*, 404–408.

Friedman, J. H. (1996). Another approach to polychotomous classification. Technical report, Department of Statistics and Stanford Linear Accelerator Center, Stanford University.

Friedman, J. H. (1997). On bias, variance, 0/1 loss, and the curse of dimensionality. *Data Mining and Knowledge Discovery 1*, 55–77.

Fu, K. S. (1968). *Sequential Methods in Pattern Recognition and Machine Learning*. New York: Academic.

Gama, J. (1997). Oblique linear tree. *Lecture Notes in Computer Science 1280*, 187–198.

Gordon, A. D. (1987). A review of hierarchical-classification. *Journal of the Royal Statistical Society Series A - Statistics in Society 150*(2), 119–137.

Grama, A. and V. Kumar (1999). State of the art in parallel search techniques for discrete optimization problems. *IEEE Transactions on Knowledge and Data Engineering 11*(1), 28–35.

Gunn, S. R. and M. Brown (1999). Supanova - a sparse, transparent modelling approach. In *Proc. IEEE International Workshop on Neural Networks for Signal Processing*, Madison, Wisconsin. IEEE.

Harris, C. J., A. Bailey, and T. J. Dodd (1998). Multi-sensor data fusion in defence and aerospace. *The Aeronautical Journal 102*(1015), 229–244.

Haykin, S. (1999). *Neural Networks - A comprehensive foundation* (2^{nd} Edition ed.). Prentice Hall.

Henrichon, E. G. and K. S. Fu (1969). A nonparametric partitioning procedure for pattern classification. *IEEE Transactions on Computers 18*, 604–624.

Hong, X. and C. J. Harris (2001). A mixture of experts network structure construction algorithm. *Applied Intelligence*. Accepted for publication.

Hughes, G. F. (1968). On the mean accuracy of statistical pattern recognisers. *IEEE Transactions on Information Theory 14*(1), 55–63.

Hughes, M., L. Bastin, and P. F. Fisher (1998). FLIERS: verification data report. Report to the EU, Department of Geography, University of Leicester.

Jain, A. and D. Zonker (1997). Feature selection: Evaluation, application, and small sample performance. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19*(2), 153–158.

Jain, A. K., R. P. W. Duin, and J. Mao (2000). Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22*(1), 4–37.

Jia, X. and J. A. Richards (1998). Progressive two-class decision classifier for optimization of class discriminants. *Remote Sensing of Environment 63*(3), 289 – 297.

Jollife, I. T. (1986). *Principle component analysis*. New York: Springer-Verlag.

Jordan, M. I. and R. A. Jacobs (1991). Hierarchies of adaptive experts. In *Advances in Neural Information Processing Systems*, pp. 985–992.

Jordan, M. I. and R. A. Jacobs (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation 6*, 181–214.

Kim, B. and D. A. Landgrebe (1991). Hierarchical classifier design in high-dimensional numerous class cases. *IEEE Transactions on Geoscience and Remote Sensing 29*(4), 518–528.

Koza, J. R. (1992). *Genetic Programming: On the programming of computers by means of natural selection.* MIT Press.

Kressel, U. H.-G. (1999). Pairwise classification and support vector machines. In B. Scholkopf, C. J. C. Burges, and A. J. Smola (Eds.), *Advances in Kernel Methods*, Cambridge, MA, pp. 255–268. MIT Press.

Kulkarni, A. V. and L. N. Kanal (1976). An optimization approach to hierarchical classifier design. In *Proceedings of the 3rd International Joint Conference on Pattern Recognition*, San Diego, CA.

Kushnir, M., K. Abe, and K. Matsumoto (1983). An application of the hough transform to the recognition of printed hebrew characters. *Pattern Recognition 16*(2), 183–191.

Landeweerd, G. H., T. Timmers, E. S. Gelsema, M. Bins, and M. R. Halie (1983). Binary tree versus single level tree classification of white blood cells. *Pattern Recognition 16*(6), 571–577.

Lin, Y. K. and K. S. Fu (1983). Automatic classification of cervical cells using a binary tree classifier. *Pattern Recognition 16*(1), 69–80.

Manslow, J. (2000). On the extraction and representation of land cover information derived from remotely sensed imagery. Phd thesis, University of Southampton.

Marcelli, A., N. Likhareva, and T. Pavlidis (1997). Structural indexing for character recognition. *Computer Vision and Image Understanding 66*(3), 330–346.

Mui, J. K. and K.-S. Fu (1980). Automated classification of nucleated blood cells using a binary tree classifier. *IEEE Transactions on Pattern Analysis and Machine Intelligence 2*(5), 429–443.

Mukandan, R. and K. R. Ramakrishnan (1998). *Moment functions in image analysis.* World Scientific.

Ng, L. S., M. S. Nixon, and J. N. Carter (1998). Combined feature selection for texture classification. In S. Singh (Ed.), *Proceedings of the International Conference on Advances in Pattern Recognition*, pp. 35 – 44. Springer.

Nilsson, N. J. (1965). *Learning Machines: Foundations of trainable pattern-classifying systems.* McGraw-Hill.

Oh, I.-S., J.-S. Lee, and C. Y. Suen (1998). Using class separation for feature analysis and combination of class-dependent features. In A. K. Jain, S. Venkatesh, and B. C. Lovell (Eds.), *Proceedings of the Fourteenth International Conference on Pattern Recognition*, Volume 1, Los Alamitos, CA, USA, pp. 453–5. IEEE Computer Society.

Oh, I.-S., J.-S. Lee, and C. Y. Suen (1999). Analysis of class separation and combination of class-dependent features for handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence 21*(10), 1089–1094.

Payne, H. J. and W. S. Meisel (1977). An algorithm for constructing optimal binary decision trees. *IEEE Transactions on Computers 26*(9), 905–916.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference.* Morgan Kaufmann.

Plamondon, R. and S. N. Srihari (2000). On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22*(1), 63–84.

Qing-Yun, S. and K. S. Fu (1983). A method for the design of binary classifiers. *Pattern Recognition 16*(6), 593–603.

Quinlan, J. R. (1987). Decision trees as probabilistic classifiers. In *Proceedings of the sixth international workshop on Machine Learning*, Irvine, CA, pp. 31–37.

Quinlan, J. R. (1993). *C4.5 : Programs for machine learning.* Morgan Kaufmann.

Ripley, B. D. (1996). *Pattern Classification and Neural Networks.* Cambridge University Press.

Rounds, E. M. (1980). A combined nonparametric approach to feature selection and binary decision tree design. *Pattern Recognition 12*, 313–317.

Safavian, S. R. and D. Landgrebe (1991). A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man and Cybernetics 21*(3), 660–674.

Schuermann, J. and W. Doster (1984). A decision theoretic approach to hierarchical classifier design. *Pattern Recognition 17*(3), 359–369.

Schurmann, J. (1996). *Pattern Classification: A unified view of statistical and neural networks.* Wiley-Interscience.

Sethi, I. K. (1995). Neural implementation of tree classifiers. *IEEE Transactions on Systems, Man and Cybernetics 25*(8), 1243–1249.

Srikantan, G., S. W. Lam, and S. N. Srihari (1996). Gradient-based contour encoding for character recognition. *Pattern Recognition 29*(7), 1147–1160.

Swain, P. H. and H. Hauska (1977). The decision tree classifier: Design and potential. *IEEE Transactions on Geoscience Electronics 15*(3), 142–147.

Trunk, G. V. (1979). A problem of dimensionality: a simple example. *IEEE Transactions on Pattern Analysis and Machine Intelligence 1*(3), 306–307.

van Breukelen, M., R. P. W. Duin, D. M. J. Tax, and T. E. den Hartog (1998). Handwritten digit recognition by combined classifiers. *Kybernetika 34*(4), 381–386.

Vapnik, V. N. (1998). *Statistical Learning Theory*. John Wiley and Sons.

Webb, A. (1999). *Statistical pattern recognition*. London: Arnold.

Zhou, J. Y. and T. Pavlidis (1994). Discrimination of characters by a multistage recognition process. *Pattern Recognition 27*(11), 1539–1549.