Contextual Image Classification

Ian Poole

May 1991

(corrected version, May 1992)

Submitted for the degree of PhD at
University College London

This work was supported by an SERC CASE award, in conjunction with the National Physical Laboratories, Teddington.
This thesis studies some of the practical and theoretical issues arising in the supervised contextual classification of image pixels.

The main practical contribution is a learning/classification system, named 'Lapwing'. The system aims to:

- learn to exploit spatial characteristics such as texture, edge and line;
- learn to exploit contextual dependencies between classes;
- be efficient at classification time;
- return 'honest' probabilistic assessments of classification confidence at each pixel;
- exploit a parallel SIMD processor for classification.

The method involves the use of a genetic algorithm to search for 'good' partitions in a high-dimensioned pattern space, the partitions being built up hierarchically as a probability tree. This tree may then be used to produce a class probability image from similar test data.

Probabilistic relaxation labelling (PRL) is a popular method of iteratively refining probabilistic assessments in the light of contextual constraints. A study of the theoretical limits of excellence for any PRL scheme is presented. The main result is that no scheme can produce assessments equivalent to those conditioned on all the data in the image. It is also shown that for such a scheme to be optimal the updating function must be tailored for each iteration and that after the first iteration the function will depend on the actual distributions of the raw data, not simply on the spatial correlation of the classes.

A new form of PRL — trained probabilistic relaxation (TPR) is presented that uses Lapwing to estimate directly the updating function for each iteration in a particular domain of application.

Lapwing is demonstrated on noisy synthetic texture discrimination, edge detection and on problems encountered in remote sensing and medical imaging, with encouraging results.
Contents

1 Introduction ......................................................... 13
  1.1 Assumed knowledge ........................................... 16
  1.2 Notational conventions ......................................... 17
  1.3 Outline of thesis ................................................ 17
  1.4 A Survey of image classification in remote sensing ....... 18
    1.4.1 Per-pixel methods ....................................... 18
    1.4.2 Object classifiers ......................................... 20
    1.4.3 Discrimination of texture ................................ 21
    1.4.4 Knowledge based classification ......................... 23

2 Lapwing - A new approach ........................................ 25
  2.1 Pixel versus object classifiers ................................. 26
  2.2 Specifying the pattern recognition problem ................. 27
    2.2.1 Pattern vector extraction ............................... 27
    2.2.2 Formulation ............................................... 29
  2.3 Honesty and refinement ......................................... 31
    2.3.1 Overall honesty ......................................... 32
    2.3.2 Complete honesty (calibration) ......................... 32
    2.3.3 Refinement ................................................. 33
    2.3.4 Remarks ...................................................... 34
  2.4 Conventional pattern recognition methods .................... 34
    2.4.1 Non-statistical methods .................................. 35
    2.4.2 Probability density function (p.d.f) estimation techniques ....... 37
    2.4.3 Summary ...................................................... 40
  2.5 Lapwing’s probability tree approach ......................... 40
    2.5.1 Constructing a probability tree ......................... 41
    2.5.2 The splitting rules ........................................ 43
    2.5.3 An additional criterion for resistance to simple bias - $Em''(t)$ .... 44
5 Results and Evaluation

5.1 Introduction .............................................................. 99
  5.1.1 Artificial vs real test images ........................................ 99
  5.1.2 General notes on the conduct of tests .................... 100
  5.1.3 Performance statistics .............................................. 100

5.2 Regular noisy texture discrimination ..................... 103
  5.2.1 Discussion of noisy texture tests .......................... 104

5.3 Markov random field textures ................................. 106
  5.3.1 Generation of MRF textures .................................... 107
  5.3.2 Discussion of Markov texture tests ....................... 109

5.4 Edge detection .......................................................... 115
  5.4.1 Discussion of edge detection test ......................... 115

5.5 Remotely sensed imagery .......................................... 118
  5.5.1 Discussion of ATM tests ........................................ 119

5.6 Biological Images ...................................................... 122
  5.6.1 Discussion of biological image test ....................... 122

5.7 Some experiments ...................................................... 125
  5.7.1 The effect of training on an image which is more or less noisy ...... 125
  5.7.2 Comparing TPR with a computational PRL scheme ........ 130

5.8 Entropy vs Classification error ............................... 132
  5.8.1 Relationship between P(error|A) and $H(Y|A)$ ........... 133
  5.8.2 Discussion of entropy metric for preceding tests ......... 134

5.9 Summary ................................................................. 135

6 Conclusions ................................................................. 139

6.1 Lapwing ................................................................. 139
  6.1.1 Putting Lapwing into context ............................... 139
  6.1.2 Further work (1) ................................................ 140

6.2 PRL, IOU and TPR .................................................... 141
  6.2.1 Comparing PRL and TPR .................................... 141
  6.2.2 Further work (2) ................................................ 144

6.3 Concluding remarks ................................................. 146

A Glossary ................................................................. 149

A.1 Abbreviations .......................................................... 149
A.2 Notation introduced in chapter 2 ............................. 150
A.3 Notation introduced in chapter 4 ............................. 151
A.3.1 Abbreviated notation used in proofs .............................................................. 152

B Statistical independence diagrams (‘I-graphs’) ...................................................... 153

C Proof of theorems ........................................................................................................ 155
   C.1 Some theorems in second level probabilities .................................................. 155
   C.2 Proofs for chapter 4 ......................................................................................... 157

D REDUCE program and output .................................................................................. 163
# List of Figures

1.1 Blewbury. ................................................................. 14
1.2 Thematic overlay. ................................................... 15

2.1 'Inside box' classification. ......................................... 27
2.2 An example neighbourhood. ....................................... 28
2.3 A variety of neighbourhood models ......................... 29
2.4 Example density functions on $Q_f$. .......................... 32
2.5 Horizontal/vertical texture discrimination. .................. 35
2.6 The Parzen estimator with Gaussian kernel. .......... 38
2.7 kth Nearest-Neighbour density estimation. ................ 39
2.8 A probability tree and equivalent partition. ............... 42
2.9 Which are the better splits? ................................. 44
2.10 Illustrating a problem with $E m'(t)$. ....................... 50
2.11 Sketch of the function $\hat{S}_{0,\tau}(b)$. ............... 50
2.12 A genetic algorithm ........................................... 53

3.1 Overview of the Lapwing system targeted to the LAP. .. 60
3.2 Illustrating the computation of $E m'$. ...................... 63
3.3 A small PPL program ........................................... 66
3.4 A small probability tree as PPL code ....................... 68

4.1 The I-graph for the tooth-comb model. ....................... 75
4.2 I-graph for the Hair-brush model. .............................. 77
4.3 Incrementally optimal updating ................................. 81
4.4 Simplified notation. ............................................. 85
4.5 How to show that $h$ is not a function. ..................... 87
4.6 Plots of min and max stage 2 PPs against $\theta$. ........ 92
4.7 A data-flow diagram of TPR. ............................... 95
4.8 Functional pseudo-code describing TPR .................... 96
5.1 Inputs for test 1 — vertical vs horizontal lines with 50% noise. ............................. 104
5.2 Outputs for test 1 — vertical vs horizontal lines with 50% noise. ............................ 105
5.3 Outputs for test 2 — Vertical vs Horizontal lines with 100% noise. .......................... 106
5.4 Example MRF textures. ................................................................................................. 110
5.5 Inputs for test 3 — Correlated vs un-correlated MRF textures. ............................... 111
5.6 Outputs for test 3 — Correlated vs un-correlated MRF textures. ............................... 112
5.7 Results for test 4 — Directional small-scale MRF textures. ............................... 113
5.8 Outputs for test 5 — Directional large-scale MRF textures. ............................... 114
5.9 Inputs for test 6 — Edge detection. ................................................................................ 115
5.10 Outputs for test 6 — Edge detection. ............................................................................ 116
5.11 Inputs for test 7 — ATM. ............................................................................................... 119
5.12 Results for test 7 — ATM. ............................................................................................. 121
5.13 Inputs for test 8 — biological. ...................................................................................... 123
5.14 Outputs for test 8 — biological. ................................................................................... 124
5.15 Outputs for test 13 — 'A'. ............................................................................................. 127
5.16 Outputs for test 13 — 'B'. ............................................................................................. 128
5.17 Calibration curves after two iterations of TPR and H-PRL. ............................... 131
5.18 Scatter plots of $H(Y|\Delta)$ vs $P(\text{error}|\Delta)$ .................................................. 137

B.1 An example I-graph. ................................................................................................. 153

C.1 To show that $g^2$ depends on the data distributions. .................................................. 159
C.2 Data distributions yielding 3 non-zero points in $d(a_i)$. .............................................. 160
List of Tables

5.1 Results for test 1 — Directional texture, 50% noise (test 9 in paren.). .... 103
5.2 Results for test 2 — Directional texture, 100% noise (test 10 in paren.). .... 107
5.3 Results for test 3 — Correlated vs un-correlated MRF textures. .......... 108
5.4 Results for test 4 — Directional small-scale MRF textures. ............... 108
5.5 Results for test 5 — Directional large-scale MRF textures. ............... 109
5.6 Results for test 6 — Edge detection. .............................................. 117
5.7 Results for test 7 (test 11 in paren.). ............................................. 118
5.8 Results for test 8 (test 12 in paren.). ............................................. 122
5.9 Results for test 13 — ‘A’ (test 14 results in paren.). ....................... 129
5.10 Results for test 13 — ‘B’. .............................................................. 129
5.11 Results for test 15 — H-PRL applied to ATM data. ......................... 131
Acknowledgements

A great many people have contributed to this thesis, some of whom I shall try to mention.

Gillian Peacegood has been a colleague and friend throughout the project, helping with my remote sensing problems, and offering many helpful comments. I am also grateful to the following people: Hilary Adams for introducing me to genetic algorithms and making her GA code available; Phil Dawid for introducing me to belief networks and helping me with several statistical problems; John Crowcroft, Mark Handley, Valerie Isham, John Washbrook and Bill Witts for many and various stimulating conversations, and Melanie Palmer for bringing to bear her editing skills on my contorted prose. Thanks also to the hardware and systems group of UCL CS for their excellent technical support.

Thanks to Graeme Wilkinson for initiating the project and to Piers Plummer for his help with the LAP. Ed Brocklehurst provided supervision at NPL. Financial support was provided by the SERC and NPL.

I am grateful to David Mason, Jeff Settle and their team at NUTIS for making their ATM data available to me; thanks also Richard Baldock at the MRC for providing me with the electron-micrograph images; the image printing program was written by James Pearson.

I would like to express my particular gratitude to Derek Long for having taken an active interest in my work, and for all his help and guidance with my many mathematical difficulties.

Finally, a big thankyou to Paul Otto — the best supervisor any student could ask for. Paul has maintained a close involvement in the project, providing me with ideas, advice, criticism and encouraged — all in most generous measure.
Dedicated to my father,

David Poole
1924–1977,

in the hope that he would have been proud.
Chapter 1

Introduction

Consider figure 1.1. This image was acquired from an airborne sensor and shows the town of Blewbury, in the Berkshire Downs, UK.¹

The image is composed of 512 \times 512 picture elements — or pixels, each covering an area on the ground of approximately 5 \times 5 metres. The image shown is the 11th from a total of 11 registered images, or bands, each recording the intensity of reflectance in a particular range of the electromagnetic spectrum. The intensity at each pixel is encoded in 8 bits (ie 256 possible intensity levels), thus the entire data set comprises 512 \times 512 \times 11 \times 8 = 22Mbits of data.²

The potential uses for this data are many and varied — plotting of urban growth, mapping of wetland sites, prediction of national crop yields ... etc. While it may be possible to fulfill these requirements by working from the raw image data, it will often be expedient to prepare first a thematic map from the data, in a machine-readable form. The thematic map will label each pixel with a category, or class, selected from a set of classes of interest to the particular application. Figure 1.2 is such a map for the Blewbury image, distinguishing roads (black), cereals (light grey), urban area (dark grey) and 'other' (white). Questions such as “what proportion of cereal crop covers the area?” and “where are urban areas expanding (relative to an earlier image)?” can be answered swiftly and mechanically. If the thematic map is sufficiently detailed (ie there is a fine division of classes) and accurate, then it could replace the original data; the thematic map has captured the significant meaning — the true information contained in the data.

This thesis is concerned with the automatic preparation of such thematic maps, or over-

¹This image was acquired by the Natural Environment Research Council (NERC). The reference plane (fig. 1.2) was compiled by the NERC Unit for Thematic Information systems (NUTIS) (it has been simplified by the author).
²However the entropy of the data will be considerably less; this is what makes image compression (for storage or transmission) possible.
lays as they will be called. Remote sensing applications will not be discussed in detail, although the type of tasks which have been mentioned probably constitute the major application for these techniques; the reader is invited to keep these concrete applications in mind as the discussions become increasingly abstract. Other applications are medical imaging (eg, labeling of abnormal blood cells from a microscope image [Norgen81]) and industrial inspection (eg, identifying faults in rolled steel [Hill83], or perhaps gauging the proportion of fat in a cut of meat).

While the thematic overlay may be prepared by hand (indeed, this will often be the only alternative), there are obvious benefits of speed, objectivity and perhaps accuracy in making the transition from image data to thematic overlay a mechanical one. This immediately raises some general questions:

- What data will be considered relevant at each pixel — all the data from all bands or a selection of bands? Should the data around each pixel be taken into account, so that
image texture can be exploited?

- Are contextual constraints to be considered — for example, that forests occur as large areas, and roads are straight, thin, and more common near urban areas?

- Should the overlay be categoric or should it give some indication of the certainty of the classification?

- What time constraints impinge? Eg, must the overlay be generated in real-time as the data is captured?

- What special hardware architectures are available?

These questions will be addressed in the following chapter, however, it is worth mentioning straight away that part of the initial motivation for this project was to investigate techniques which would exploit a particular hardware architecture — that of the Linear array.
CHAPTER 1. INTRODUCTION

processor (LAP). The LAP has a 'single-instruction multiple-data' (SIMD) architecture and was developed at the National Physical Laboratories (NPL). An introduction to the LAP is provided in chapter 3.

1.1 Assumed knowledge

It is clear that we are primarily concerned with a problem of pattern recognition. That is, given some data $X$, associated with each pixel, assign to it a class $Y$.

This thesis cannot serve as an introduction to this large and rapidly developing discipline. There are many texts available to fulfill that role. Particularly useful for the beginner are [Duda72] and [Tou74]. For progressively more advanced texts see [Fukunaga72], [Patrick72] and [Devijver82].

It will be assumed that the reader is familiar with the basic terms and concepts of pattern recognition, such as a pattern vector, pattern space, decision function, supervised and unsupervised recognition, and basic probability theory such as the a priori probability, a posteriori probability, conditional density and Bayes theorem.

Knowledge of popular pattern recognition techniques such as the nearest neighbour (NN), minimum distance (MD) and Gaussian maximum likelihood (GML) classifiers will be useful.

---

3 Pattern vector — the (usually multivariate) data upon which classification is to be based ($X$).
4 Pattern space — the space of all possible values the pattern vector may take — visualise as a space with dimensionality equal to the number of components in the pattern space ($\Omega$).
5 Decision function — assigns a given pattern vector to a class — essentially this is the classifier ($D(X)$).
6 Supervised recognition — classified samples are available to train the classifier in advance.
7 Unsupervised recognition — no classified training samples are available; we wish to assign similar patterns to the same group or cluster.
8 A priori probability — the probability of finding a particular class prior to seeing any data (eg $P(Y = \text{forest})$).
9 A posteriori probability — our modified probabilistic assessment after (or given that) we have seen some data (eg, $P(Y=\text{forest}|X=x)$).
10 Conditional density — the probability density within a given class (eg $p(X=x|Y=\text{forest})$).
11 Bayes theorem — $P(Y=\text{forest}|X=x) = \frac{p(X=x|Y=\text{forest}) \cdot p(Y=\text{forest})}{p(X=x)}$.
12 NN classifier — assign a new pattern vector to the class of the sample to which it is closest in the pattern space.
13 MD classifier — assign a new pattern to the class whose sample mean vector is closest in pattern space.
14 GML classifier — estimate mean and co-variance statistics so that a conditional density may be computed for each class from the Gaussian distribution; assign a new pattern to the class whose conditional density at the given point in pattern space is greatest.
1.2 Notational conventions

Random variables are written in capitals (eg \(Y\)) and their realizations in lower case — eg, \(y_i\). Underscore is used for a joint list of variables — eg, \(y = (y_1 \ldots y_n)\). Upper case 'P' indicates the probability of a discrete event — eg \(P(Y = y_i)\) reads "the probability that \(Y\) takes the value \(y_i\", and \(D(Y)\) is the probability vector (or distribution) over all realizations of the discrete random variable \(Y\). A variable which represents a probability vector is written bold — eg, \(p_i = D(Y|X=x_i)\). A probability density on a continuous variable is indicated by \(p\) — as in \(p(X=x_i)\), with \(d(X)\) standing for the density function. In the few instances where a transition matrix is required, this is indicated by 'M' — eg, \(M(Y_i|Y_{i-1})\).

A glossary of notation may be found in appendix A.

1.3 Outline of thesis

In the following chapter the design goals for a supervised image classification system will be set out and a system, named Lapwing, developed to achieve them. The system delivers a probabilistic classification; a recurring theme will be the degree to which these probabilities are 'honest', and so a section is included that defines classifier honesty and related terms. Lapwing uses an optimization technique (a genetic algorithm (GA)) to construct a probability tree from training data. Implementation issues are addressed in chapter 3; in particular it is shown how classification may be performed on a linear array processor.

Chapter 4 contains the main theoretical results of the thesis. It considers ways of exploiting the contextual dependencies between classes to improve overall classifier confidence, and focuses on a particular and well studied technique — probabilistic relaxation labeling (PRL). Theorems are developed which establish a limit of excellence for this class of methods and highlight the short-comings of existing implementations. Some of the longer theorem proofs are removed and included in appendix C. Finally, a radically new scheme, trained probabilistic relaxation (TPR) is proposed which depends on Lapwing or a system like it for its implementation; this is the unifying link between chapters 2 and 4.

Experimental results from a variety of real and artificial problems are presented in chapter 5. TPR is compared with a particular version of PRL from the literature. The chapter also includes some discussion of the use of conditional entropy as a classifier performance metric.

A short summary concludes each chapter, but more detailed discussion is presented in chapter 6.

A glossary of abbreviations and notation may be found in appendix A.

This chapter concludes with a brief survey of the techniques which have been used or
proposed for remote sensing application.

1.4 A Survey of image classification in remote sensing

In this section techniques are reviewed that are concerned with the ultimate classification of image pixels. Work on contextual classification will be reviewed in chapter 4. This thesis is not concerned with image segmentation as such, though methods which involve segmentation as a pre-processing step to ultimate classification will be looked at briefly.

See [Swain78b], [Curren85] or [Lillesand87] for introductory texts to remote sensing.

1.4.1 Per-pixel methods

Probably the most common classification methods used in remote sensing are those based on some form of statistical analysis of the data for each pixel in isolation. For these methods, all spatial aspects are entirely ignored; the image is treated as an unstructured 'bag' of pixels and so can usually be used only when some form of multivariate data is available for each pixel (e.g., they would be inappropriate for panchromatic data). In remote sensing, this will usually be multispectral or possibly multitemporal data. Thus, the components of the pattern vectors will be composed of the intensities in different spectral bands or at different times. Principal component analysis will often be used first to select band components containing the greatest information.

A review of early work on remote sensing applications is provided in [Fu76], in which per-field classifiers, sequential pattern space partitioning (classification trees), cluster analysis and syntactic methods are discussed.

Supervised methods

The most commonly used classifiers in remote sensing are the parallelepiped, MD and GML classifiers [Lillesand87]. Given the approximate normal distribution in spectral intensity space for carefully specified classes [Crane72] the GML decision surfaces would be expected to fit the class distributions more accurately than the cuboid regions implied by the parallelepiped classifier. The MD classifier suffers from the assumptions of equal co-variances among the classes.

One of the principal concerns in per-pixel image classification is the sheer quantity of observations involved. While the GML classifier may be preferable in terms of accuracy it involves considerable computational effort — in its crude form the multivariate density function must be evaluated \( m \) times for each pixel where \( m \) is the number of classes involved. Thus, various modifications to the GML classifier have been proposed to reduce this
computational load.

A hybrid of two classifiers can be used — use the parallelepiped classifier by default, resorting to the more costly GML classifier only where the cuboids of two classifiers overlap [Haberacker79]. A related technique involves the definition of two cuboid regions for each class, termed the necessary and sufficient regions for that class [Koo86]. Only when the pattern vector for a pixel falls in the cavity between the two regions is it necessary to evaluate the GML rule.

The efficiency of the GML classifier is tackled in a more direct manner by [Feiveson83]. He describes a method by which fixed probability thresholds can be pre-calculated for each class-conditional density function; if this probability is exceeded, then it is unnecessary to calculate the density functions of the remaining classes.

It is unlikely that every pixel in an image will be unique; it is, therefore, possible to avoid repeated computation of a decision rule by function-memorization — storing the results in a lookup table as classification proceeds [Mather85]. The table can be implemented with a hashing algorithm to avoid unreasonable storage demands.

There have been several comparative studies to assess the efficacy of the various pattern recognition techniques to image classification in respect of both classification accuracy and computational efficiency.

The GML classifier (with priors) is compared with linear discriminant analysis for land use mapping in [Tom84], which concludes that the latter achieves greater classification accuracy and reduced computational load. This presumably implies that the assumption of Gaussian statistics was inappropriate in their application.

The performance of the GML and NN classifiers are compared by Ince [Ince87], using Landsat TM data, with the conclusion that the NN classifier is significantly the more accurate due to the often inappropriateness of the Gaussian assumptions, and to the problem of estimating the parameters. However, he also comments, as do many other authors, on the practical difficulties of implementing the NN classifier in high-dimensional data (his implementation involves a literal multi-dimensional array).

**Unsupervised methods**

Although we are not primarily concerned with unsupervised classification, this can be a useful technique to assist in the preparation of training data [Lillesand87]. The use of unsupervised learning is recommended even when training data is available, as a means of determining multiple modes of a perceived single category [Patrick72]. Furthermore, Coleman and Andrews [Coleman79] achieve image segmentation via unsupervised pixel classification using a clustering algorithm.
A clustering scheme for four-band multispectral data is described in [Goldberg78]. A four-dimensional histogram is constructed using a hashing method in order to avoid unrealistic storage demands. Histogram cells containing more than a certain threshold of observations are considered as peaks; peaks connected in pattern space are merged into one. The remaining cells are assigned to the nearest connected peak.

The 'watershed' algorithm (see [Watson87]) is reminiscent of region-growing techniques used in image segmentation, but applied to the pattern space rather than the image space. Multi-band imagery must first be reduced to two dimensions by some suitable transformation so that a bivariate histogram can be constructed on a hexadecimal grid. This histogram can then be treated as an 'image'. 'Bright' points in this 'image' are located and connected with any adjacent points that are just one observation count lower. This is repeated for points that are two lower and so on. The result is a set of connected regions that can be arbitrarily labeled. The 'image' can then be used as a lookup table to classify the (bivariate) observations.

1.4.2 Object classifiers

The accuracy of classification can often be improved if homogeneous regions of the image are first identified by segmentation, and the combined statistics of all the pixels in the 'object' are used for its classification. Clearly, this approach depends crucially on the quality of the segmentation.

The primary example of this approach is the ECHO system — 'Extraction and Classification of Homogeneous Objects', described in [Kettig76] and [Landgrebe80]. The segmentation is achieved by first dividing the image into a regular grid of cells and then merging adjacent cells if they pass a statistical test of similarity. The key to the classification stage lies in two assumptions: a) that the pixels of a region belong to the same class and b) that the observational data is \textit{class-conditionally independent}. These assumptions permit the factorizing of the joint conditional density function for all the pixels into a product of their individual densities. A maximum-likelihood rule is then applied to the joint conditional densities. The appropriateness of the class-conditional independence assumption is considered in [Poole88e].

In [Kalayeh87], class-conditional correlation is tackled by modeling the interactions with a Markov mesh model. They develop modifications to MD and GML object classifiers, which take account of this model.

\footnote{In [Kalayeh87], class-conditional correlation is tackled by modeling the interactions with a Markov mesh model. They develop modifications to MD and GML object classifiers, which take account of this model.}
1.4. A SURVEY OF IMAGE CLASSIFICATION IN REMOTE SENSING

1.4.3 Discrimination of texture

Many authors (eg [Palgen70], [Mason87], [Alm85]) have remarked that classification methods that ignore the spatial features of the image — as do the methods described so far — are severely limited in their discriminatory power. Clearly they cannot discriminate texture. Texture classification may become increasingly significant in remote sensing with the advent of 10-meter resolution, panchromatic data provided by the SPOT satellite.

Enrich and Foith [Enrich78] list three issues with which researchers have been interested concerning texture: texture classification, texture modelling (synthesis) and texture segmentation. It is the latter of these that has been the dominant subject of research in recent years. Here we must confine ourselves to only the former. However, both classification and segmentation require the extraction of features that will permit discrimination between two differently textured regions. Haralick [Haralick79] surveys many approaches to deriving texture features and categorises them into ‘optical transforms, digital transforms, textural edginess, structural element, grey tone co-occurrence, run lengths and autoregressive models’. The review has been expanded and updated in [Haralick86a].

Various statistical features for describing texture are discussed by Chen and Pavlidis [Chen79]. First-order statistics are concerned with only the frequency distribution of the grey levels in the image. These include the mean, variance, skewness and kurtosis. Second-order statistics are concerned with the joint intensity distributions of pairs of pixels separated by a given displacement $(dx, dy)$. For each displacement these statistics can be captured by a co-occurrence matrix. These are similar to the ‘grey-tone, spatial-dependence matrices’ first proposed by Haralick et al [Haralick73]. The $(i,j)$th entry of this matrix gives the probability of finding a pixel of intensity $i$, with a pixel of intensity $j$ at a displacement of $(dx, dy)$ from it (assuming a stationary image field). The matrix is symmetric. Typically, the number of significant intensity levels is reduced (eg to eight) for obvious storage and sampling reasons. Chen and Pavlidis simplify the situation further by considering values for $(dx, dy)$ of only $(0,1), (1,1), (1,0)$ and $(1,-1)$, and forming a sum of these four matrices. (Note that the summing discards any directional information, which may or may not be desirable.) As they are concerned with segmentation, they define a similarity measure between two matrices to establish whether two matrices derive from the same texture. In later work [Chen83] they use the correlation between displaced pixels in place of co-occurrence matrices. They assume that the joint intensities of the pixels at a given displacement follow a Gaussian distribution and thus can be described by a two entry mean vector and a $2 \times 2$ co-variance matrix. For any given region, these can be estimated directly.

An obvious candidate for texture discrimination are statistics derived from frequency transforms of the image. Indeed, in what seems to be the earliest published work on auto-
matic image analysis, Rosenfeld [Rosenfeld62] uses analog techniques to analyze the spatial frequency of the intensities across scanned lines of aerial photographs. This technique has been further developed in [Idelsohn70] to produce a trainable classifier. The approach generalizes to the digitally computed two dimensional Fourier transform of a (sub) image and is described in many places — see [Bajesy73a], [Harris80] [Ballard82]. Fourier features based on phase are less useful for texture classification than those based on amplitude [Eklundh79]. It also appears that Fourier features in general are inferior to the first and second order statistics described above, for terrain classification [Weszka76].

A very different approach to obtaining texture feature is proposed by Laws [Laws80] and further developed by Pietikainen et al [Pietikainen83]. This involves a set of zero summing $5 \times 5$ convolution masks which are derived systematically from simple one dimensional masks. Laws selected the eight most useful from the set. These are convolved in turn with a sub image (which is assumed to be texturally homogeneous) and the sum of the resulting absolute intensities is calculated for each. These eight values form a feature vector for treatment by an appropriate classifier.

The method of 'adaptive windows' [Aleksander82], [Aleksander83] has been applied to texture discrimination by Kani and Wilson [Kani87]. Following is a brief description of the adaptive window (or 'WIZARD' nets as they are commonly known).

Sets of random access memories (RAMs) are addressed by randomly selected pixels from a region of a binary image. There is one set of RAMs (each identically connected) for each pattern class of interest. During training several patterns are presented to the system and the RAM set of the associated pattern class is write-enabled. At classification time, with the RAMs in read-mode, outputs of each RAM are summed within each class set and the most voted-for class is chosen. It is the speed with which the discriminators can be trained and applied that constitutes the main advantage of the method. The 'confidence' of a particular decision is related to the relative number of votes of the 'winner' and 'runner-up'. In [Aleksander68], [Aleksander83], [Kani87] and [Masih88], a feedback mechanism is described which demonstrably improves the 'confidence' of the decision. However, it is by no means obvious that the reported increase in confidence need have any relationship with actual improvement in classification accuracy, since the feedback incorporates no new information.

Kani and Wilson train an adaptive window on textures that have equal first order but different second order statistics. They present results in terms of discriminator confidence as described above, but do not report the rates of misclassification.
1.4. Knowledge based classification

The methods described so far have been, loosely, statistical. In addition, there have been attempts to apply artificial intelligence techniques (see eg [Charniak85]) to the task of image interpretation. As this can ultimately produce pixel classification, some of this work will be mentioned here. A common feature of these systems is that they work at the object level, these objects having been first identified by some initial segmentation process. They are largely model and/or rule based. Reviews can be found in [Binford82].

Nagao et al [Nagao79a] describe a system for the interpretation of aerial photographs. An initial segmentation first roughly categorises the scene into a) large homogeneous regions b) elongated regions c) large vegetation regions d) high contrast regions. Each of these groups is then analyzed in greater detail by knowledge base modules tailored for each type. A further stage then improves on the initial region classification in the light of the classification of neighbouring regions. While good results were demonstrated, the techniques used are largely ad-hoc.

The system described by Nazif and Levine [Nazif84] is independent of domain, and is intended for image segmentation. Rules operate on image primitives such as line segments and regions to establish a consistent interpretation. A further set of 'meta-rules' control, and focus the attention of, the system.

Peacegood [Peacegood89] applies knowledge-base techniques to the task of detecting linear features (road, river, railway, field boundary etc) in aerial imagery. During a pre-processing stage, edge and line segments are detected and placed into a relational data-base. A collection of domain specific rules then act on this data-base, using relationships such as 'is parallel to ...', 'is straight' etc, to generate a probabilistic classification of each line/edge segment. A final stage involves the construction of a Bayesian belief network [Pearl88] to update the initial classification in view of contextual dependencies between spatially related segments.

Clearly much improved results can be achieved when some form of map data is available to assist in the segmentation and classification. The rule-based interpretation described by [Mckeown85] works in conjunction with an image/map data-base of the area being analyzed. The system is tailored to the specific task of airport interpretation. A similar system has been developed by [Mason87], but is intended for more general, lower resolution scenes. It is capable of working flexibly with multispectral, multitemporal and digitised map data as available.
Chapter 2

Lapwing - A new approach

In this chapter a novel means for performing supervised image classification is developed, intended to satisfy the following design goals.

1. The system should be trainable by example to a wide range of applications.

2. It should take account of local spatial features such as texture, edge and line, as well as information from multi-band imagery (multi-sensor, multi-spectral etc).

3. Potentially subtle contextual dependencies between classes should be learnt and exploited where they exist.

4. The classifier should yield a probabilistic assessment of its confidence; that is, we require the a posteriori probability (PP) vector for each pixel.

5. The methods used should be efficient at classification time and suitable for implementation on parallel SIMD architectures to improve speed further.

The case for goal 4 may not be obvious. It shall in fact become clear in chapter 4, that the existence of a classifier that generates (class conditional) a posteriori probabilities (PPs), on the basis of spatial relationships, is a vital requirement for the implementation of the context exploiting scheme which will be presented in that chapter; ie, it is a pre-requisite to goal 3. In any case, the PPs, when displayed as grey tones or pseudo-colour, can be useful for the human interpretation of the classifier’s results. Furthermore, the a posteriori probability is the measure which retains all information relevant to the specified classes. (see eg [Dawid85]; see also lemma 11 in appendix C).

The layout of this chapter is as follows. Section 2.1 discusses the dichotomy of pixel and object-based approaches. Section 2.2 specifies the pattern vector and formulates the problem in terms of classical pattern recognition. Section 2.3 pauses to consider the issue of classifier honesty which will be a recurring theme throughout this thesis. Conventional solutions to
the problem formulated in §2.2 are considered in §2.4. The solution finally adopted for Lapwing is then described in detail in §2.5, and the method is summarized in §2.6.

2.1 Pixel versus object classifiers

It was observed in the introductory chapter that image analysis — leading to ultimate pixel classification — can be approached in two very different ways:

- by classifying on a per-pixel basis, possibly taking into account neighbouring pixels (see §1.4.1);
- by segmenting out ‘objects’ (regions) and then assigning all the pixels in each object to the same class based on information derived from the whole object (see §1.4.2 and §1.4.4).

The primary advantage of the latter approach is that higher level features can be used when making the classification — for example, shape features such as area/perimeter factors, [Mason87] [Nagao80] or the presence of right-angles [Mckeown85]. Alternative representations of shape may be used, such as the Fourier description [Brill68], or the Hough transform [Hough62] [Duda72] [Leavers86b]. If the objects are line segments, then line length and sinuosity (the ratio of length to end separation) may be helpful (see eg [Peacegood89]). Features may also be derived from various aggregates of the pixel intensities within the object, such as simple first order statistics (eg mean and variance) [Mason87], second order-statistics from co-occurrence matrices [Chen79] or features from the fourier spectrum [Harris80].

It has become customary to deride per-pixel classifiers as being over simplistic — nevertheless, this is the strategy adopted in this thesis, for the following reasons:

- Object classifiers are crucially dependent on the quality of the initial segmentation. From the plethora of work that continues to be published on the topic (see [Haralick85] and [Rosenfeld88] for reviews), it may be concluded that the segmentation problem remains far from solved; no definitive segmentation method has yet emerged.

- The set of features extracted from the object must be decided in advance; this is a problem of feature selection, implying a loss of information. Of course, pattern recognition is all about discarding information not relevant to the task in hand, however, if we wish our system to be truly domain independent then as much information as possible must be retained until we know what the task is; the feature selection process would inevitably pre-judge the issue. A related point is that the form of the segmentation may itself need to vary with the application, raising the need for a ‘trainable
2.2. SPECIFYING THE PATTERN RECOGNITION PROBLEM

specifier'. (See [Reynolds83] for an approach to this involving the interaction of a human operator).

- Per-pixel schemes are more appropriately implemented on SIMD architectures as there is an obvious mapping — a pixel (or column of pixels) per processor. (However, see [Otto86] for the implementation of various object based algorithms on such machines).

Raster scanned images in their raw form are arrays of picture elements, thus any scheme must ultimately be expressible as a function of the pixel intensities. Object classifiers permit the classification of a pixel to be influenced by an intensity which is remote from it. Consider for example the somewhat perverse task of classifying pixels as being inside a rectangle, as shown in fig. 2.1. Clearly this problem is most sensibly tackled by an object based scheme,

however, provided a per-pixel classifier is endowed with a window twice the size of the largest box, then, when this window is centered on any point inside a box, the whole box will be 'visible', and the potential for correct classification exists, at least in principle.

The remainder of this thesis is concerned with per-pixel classification techniques.

2.2 Specifying the pattern recognition problem

2.2.1 Pattern vector extraction

Having decided that the classification will be based upon raw pixel intensities, rather than upon features extracted from segmented objects, pattern vector extraction becomes straightforward. We do however need to decide on precisely which pixels will be involved, ie, to define

Figure 2.1: 'Inside box' classification.
the *neighbourhood model*. Possible choices include the whole image (!), a random (fixed) selection around the pixel (as with Wizard [Aleksander83]), or a pre-determined window. Lapwing uses the latter, though a small degree of sophistication is introduced in order to extract information at multiple resolutions, and as a minor sop to efficiency.

Lapwing's neighbourhood size can be varied, but is essentially of the form shown in fig. 2.2. In this case, 18 values are extracted, 9 directly from the central region, plus 9 averages (arithmetic mean) from the outer $3 \times 3$ squares (the central $3 \times 3$ average is designated $x_{10}$, but is not shown in the diagram). This arrangement allows the perception of fine detail at the centre, as well as coarse structure at a greater distance, whilst keeping the number of features, $d$, within reasonable bounds ($d = 18$ rather than 81).

The neighbourhood just described is based on a $3 \times 3$ grid, extracted at two resolutions. We can generalize within this basic structure, denoting by $N_w$ the width of the basic neighbourhood, and by $N_I$ the number of resolution levels used. Thus, the neighbourhood model of figure 2.2 can be specified by $N_w = 3, N_I = 2$. Figure 2.3 shows some other possibilities. When multi-band imagery is used, the described neighbourhood is extracted from each band, multiplying the vector size accordingly. For notational completeness then, we may write the number of bands to be used as $N_b$. Thus, conventional per-pixel schemes used in remote sensing on, say, five band imagery, use a neighbourhood model $N_w = 1, N_I = 1, N_b = 5$. The ability to accept more than one input image also permits pre-processed versions of the image — a variance transform for example — to be presented to the system when it is suspected that this may be useful. Clearly these neighbourhoods generate a vector of size $d = N_b \cdot N_I \cdot N_w^2$ and cover $N_b \cdot (N_w \cdot N_I)^2$ raw pixel values. It must be stressed that the specific neighbourhood model must be decided upon in advance — $N_w, N_b$ and $N_I$ are system parameters, not run-time variables.

![Figure 2.2: An example neighbourhood.](image-url)
2.2. SPECIFYING THE PATTERN RECOGNITION PROBLEM

2.2.2 Formulation

Pattern vector — \( x \in \Omega \)

An image is a collection of \( N_T \) pixels \( I = \{i| i = 1..N_T \} \). With each pixel \( i \) is associated its pattern vector \( x_i \) extracted from the image in the manner described in the previous section (so for the neighbourhood system shown in fig. 2.2, each \( x_i \) has 18 components). The associated (vector valued) random variable is written \( X \). The order of the components in the pattern vector can be inferred from the diagrams; it is clearly arbitrary. The space of all possible vectors is a continuous \( d \) dimensioned hyperspace, termed the pattern space, and denoted \( \Omega; x_i \in \Omega \). Each component of \( x_i \) takes values from the range \([0, l] \).\(^1\)

Pattern class — \( y \in \Phi \)

Also associated with each pixel \( i \) is its class \( y_i \). The set of all possible classes is \( \Phi \), \( \Phi = \{1..m \} \) — meaning eg \{urban, forest, water \}.. The associated random variable is written \( Y \). As a notational convenience, we will often write \( y_i \) as shorthand for \( Y = y_i \). The subscript \( i \) will often also be omitted and the class realizations \( \alpha, \beta \in \Phi \) used when the identity of the pixel is not of interest.

Training sample — \( \mathcal{L} \)

The training (or ‘Learning’) sample is presented to the system in the form of an example image (or image stack) in conjunction with a registered training overlay which indicates the true class for each pixel. We have already met an example of an image/overlay pair in figures

\(^1\)In practice, \( \Omega \) will be a discrete space, with the vector components typically taking values from the set \( \{0..255 \} \) (ie 8-bit pixel data); it is convenient however to treat the space as being continuous for the theoretical discussions in this chapter.
1.1.2. From these it is possible to obtain for each pixel \(i\), realizations of its class \(y_i\) and pattern vector \(x_i\) (except that pixels at a distance of \(N_w \cdot N_l\) or less from the edge may not be used, thus \(L\) is taken to exclude these marginal pixels). Formally then, the training sample \(2\) is a set of \(N_L\) pairs:

\[
L = \{(x_i, y_i) | i = 1..N_L\}
\]

Note that unless \(N_w = N_l = 1\) the sample-points will not be independent since their pattern vectors will overlap.

The problem

We can now pose our task succinctly:

*Find the best refined and honest (see below) estimate of the a posteriori probability vector \(D(Y|X=x)\), for any \(x \in \Omega\), based on the training sample \(L\).*

Providing the prior probabilities \(D(Y)\) are available, it would be sufficient to find the conditional density functions \(d(X|Y=\alpha)\) \(\forall \alpha \in \Phi\) since by Bayes' theorem:

\[
P(Y=\alpha|x) = \frac{p(x|Y=\alpha).P(Y=\alpha)}{\sum_{\beta \in \Phi} p(x|Y=\beta).P(Y=\beta)}.
\]

(2.1)

The decision function, \(D(x) \in \Phi\) then follows immediately either by —

\[
D(x) = \alpha \text{ if } P(Y=\alpha|x) = \max_{\beta} P(Y=\beta|x)
\]

(2.2)

or by —

\[
D(x) = \alpha \text{ if } p(x|Y=\alpha).P(Y=\alpha) = \max_{\beta} p(x|Y=\beta).P(Y=\beta)
\]

(2.3)

The first form is commonly termed the maximum a posteriori probability (MAP) rule, and the second the Bayes rule, though they are equivalent by 2.1 — the normalizing denominator does not depend on \(\alpha\) and so can be dropped.\(^3\)

The problem has been stated in statistical pattern recognition terms. A similar formulation, for template matching, is alluded to by [Duda73, §7.5.2], however they comment that 'the application of formal statistical methods to problems of detecting objects in pictures

---

\(^2\)By sample we intend the usual statistical meaning — i.e. a collection of 'units' drawn from a population (see eg [Cochran63]). Our population is the set of all 'units' (i.e. pattern vector / class pairs) which the system may be called upon to deal with. Our sampling technique consists in manually selecting a single image which we judge to contain a representative set of these 'units'. We cannot therefore claim to have a strictly random sample of our population, but will treat it as such. In this thesis we will use the term point or sample-point to refer to a single 'unit'.

\(^3\)In the event of a tie between two or more classes, \(D(.)\) must make an arbitrary (though repeatable) decision.
2.3. HONESTY AND REFINEMENT

has proved difficult in practice'. They consider the problem to be tractable only if the distribution of intensities can be considered as statistically independent, thus permitting the factorizing of the joint distributions.

Before going on to consider conventional solutions to this problem, a digression ....

2.3 Honesty and refinement

Questions concerning the honesty of a classifier a will occur frequently; this section establishes the statistical grounding of the concept and relates it to similar terms use by other authors. A full discussion of these issues may be found in [Dawid85] and [Dawid86] — the latter being a particularly readable account. Dawid’s discussion relates to probabilistic ‘forecasting’ — taking the particular example of weather forecasting. In the Unites States it is common practice to issue a ‘probability of precipitation’ (‘PoP’), and the techniques to be described in this section are used routinely to ensure the honesty of these predictions.

The honesty of a classifier becomes an issue only when it claims to deliver a probabilistic assessment of its decision — ie when it admits to, and attempts to quantify, its own uncertainty. The question then arises: “how honest is the classifier being in its admissions?”.

Consider the three-class classifier — \( \Phi = \{ \text{forest, urban, other} \} \) — that returns a quantity \( Q \) at each pixel which it claims represents the a posteriori probability vector on \( Y \), given some data \( \Delta \) (realization written \( \delta \)). We consider \( Q \) to be a random variable (functionally dependent on \( \Delta \)), realizations of which will be written \( q \). So the classifier claims:

\[
Q = D(Y|\Delta).
\]

Let the component of \( Q \) relating to forest be \( Q_f \). Suppose that on examining a random sample of \( Q_f \) realizations from the classifier we observe numbers such as .04, .07, .92, .02, .91, .97 etc. The classifier invariably returns probabilities which are very close to zero or one, indicating a high degree of confidence that the MAP decision based on these values will be correct. The distribution \( d(Q_f) \) for this classifier might look like that of figure fig. 2.4 (a). Another classifier (using different data perhaps) might generate \( Q_f \) values such as .33, .47, .53, .21, .37 etc. Here the \( Q_f \) values appear to be hovering, non-committal, around the value of 0.4 — presumably the prior probability of forest. The distribution \( d(Q_f) \) would now look something like that of fig. 2.4 (b). This classifier appears to be very cautious in its predictions.

Suppose that the MAP decision rule based on the results of the first classifier in fact turned out to be incorrect 25% of the time. Intuitively, \( Q \) seems to be claiming far greater accuracy, and so we would conclude that this is dishonest since it is being over confident, or optimistic in its predictions. Conversely, if the MAP decisions based on the second classifier
in fact turned out to be correct 95% of the time then we would conclude that it is also being dishonest in delivering overly cautious, or pessimistic predictions.

The following two sub-sections define two criteria coined overall honesty and complete honesty; the latter is the more demanding and encompasses the former.

2.3.1 Overall honesty

A truly honest classifier is capable of predicting its own error-rate — the percentage of its incorrect MAP decisions — over a given sample. It is easy to show (see §5.1.3) that for a classifier which delivers genuine a posteriori probability vectors $q_i$ at each pixel $i$ —

$$\text{predicted error-rate} = (1 - \frac{\sum_i q_i \cdot D(q_i)}{N})$$

where $q_i \cdot D(q_i)$ is the component of $q_i$ selected by the MAP decision rule — ie the highest valued component. It should be emphasised that predicted error-rate (as with actual error-rate) is a quantity attached to a particular test set — it will not in general be the same as the error rate expected over the entire population.

The actual accuracy is simply the percentage of pixels incorrectly classified (by $D(Q)$) as compared with ground truth. Overall honesty is the requirement that:

$$\text{predicted error-rate} \approx \text{actual error-rate}.$$ 

How different these figures must be before we conclude that the classifier is significantly dishonest is not addressed — the two quantities will simply be compared subjectively.

2.3.2 Complete honesty (calibration)

We might demand more than the above condition and insist that the error predictions are correct within small bands of $Q_y$ values, so that of those pixels assigned $Q_y \approx 0.2$ (between
0.15 and 0.25 say), 20% are in fact forest. Thus, complete honesty requires:

\[ P(Y = y | Q_y = q) \approx q, \forall q \in [0, 1], \forall y \in \Phi. \] (2.4)

Again we do not address questions concerning the required strength of the equalities.

Dawid [Dawid86] lists the terms ‘unbiased in the small’, ‘reliable’, ‘valid’ and ‘well calibrated’ as having been used in the statistical literature to describe this strict requirement for honesty. The calibration of a classifier may be displayed as a ‘calibration curve’ (see [Dawid86], also suggested in [Poole88e]) which plots \( P(Y = y | Q_y = q) \) against \( q \). Separate curves are plotted for each class in multiclass problems. Examples of a calibration curve for an honest and a dishonest classifier may be seen in figures 5.17 (a) and (b) respectively.

A classifier returning any genuine probabilistic quantity will be a priori well calibrated regardless of its conditioning. For example, the quantity \( Q_r = P(\text{rain} | \text{DAY-OF-WEEK}) \) will be perfectly honest — but of little use! Dishonesty may arise either when probabilities are obtained subjectively, when estimations are made from a sample of insufficient size (for the fitted model), or when they are computed under statistical assumptions that are not entirely valid.

2.3.3 Refinement

Suppose a (multi-class) classifier claims to deliver the vector quantity

\[ Q = D(Y | \Delta) \]

and we wish to verify the truth of this claim. Firstly, \( Q \) claims to be a probabilistic quantity and so we require it to be well calibrated and satisfy condition 2.4. But this is not sufficient to verify the classifier’s claim since as has already been remarked, this will be satisfied by any probability (on \( Y \)) — the prior probability \( D(Y) \) for example! If the classifier claims that \( Q \) is already fully conditioned on \( \Delta \) then we should also find that \( \Delta \) gives us no more information about \( Y \) than does \( Q \), so that:

\[ D(Y | Q = q, \Delta = \delta) \approx D(Y | Q = q), \forall q, \forall \delta. \]

When a classifier is completely honest, but does not satisfy the above criterion we say that it is not fully refined on the data \( \Delta \). This will arise, for example, when the width of the kernel used by the Parzen estimator (discussed in §2.4.2) is too great, or, as shall be seen in §2.5.6, when a probability tree is pruned back too severely.

Refinement is difficult to verify in practice, however, since \( \Delta \) will almost certainly have a high dimensionality — if we could perform the check then we could presumably construct the perfectly refined classifier!
It is possible for a classifier to be perfectly refined on the data $\Delta$ and yet not be honest. In this case the classifier is capturing all the information contained in $\Delta$ and relevant to $Y$, but is not delivering a posteriori probabilities. In such cases the classifier must be delivering some reversible function of the a posteriori probability and thus it will usually be a simple matter to re-calibrate the classifier after observing its outputs on test data for which the true classification is available. The re-calibration process is equivalent to training a second classifier on the output of the refined but dishonest classifier.

2.3.4 Remarks

Several advantages accrue to a classifier which returns an honest probabilistic assessment:

- As has been seen, it can predict its own error rate over a particular test set (not just in general).

- A decision might have 'economic' consequences, presenting a so-called 'cost-loss' problem. It is shown in [Dawid86], that only by delivering the completely honest a posteriori probability vector, can these decisions be taken in a statistically justifiable way.

- When the results of a classifier are to be post-processed, to take account of contextual constraints, for example, probabilistic reasoning can only be applied if the classifier delivers genuine (completely honest) probabilistic quantities.

- For an important application one might consider using several classifiers — each based on different features perhaps, or employing different techniques. It is then quite possible that each classifier will be well refined in some regions of the pattern space but poorly refined in others, but that these regions are different for each classifier. Providing the classifiers deliver completely honest PPs it would be reasonable to select the MAP decision of the most confident classifier, in each case. Clearly this would be meaningless if one or all of the classifiers were being wildly under- or over-confident.

In the Results and evaluation chapter, both the actual and predicted error-rates will be given, permitting the overall honesty of the PPs generated by Lapwing to be assessed.

2.4 Conventional pattern recognition methods

What are the salient characteristics of the problem set out in §2.2? Firstly, the pattern space $\Omega$ has a high dimensionality — eg 18, 27, 50 .... Secondly, the class distributions in this space will in general be multi-modal. The latter point can be appreciated by considering figure 2.5, illustrating a problem of directional texture discrimination. Assuming the lines
are exactly one pixel wide and one apart, then both classes will have two modes, eg, for the vertical class, (with $N_w = 3, N_i = 1$) and assuming binary data:

\[
\begin{array}{ccc}
0 & 1 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
\end{array}
\quad \text{or} \quad
\begin{array}{ccc}
1 & 0 & 1 \\
1 & 0 & 1 \\
1 & 0 & 1 \\
\end{array}
\]

In fact the problem can be reduced to the 'exclusive or' type which is easily shown to be linearly inseparable (see [Minsky88, §2.1]).

We will now consider the appropriateness of a number of 'standard' pattern recognition techniques to the formulated problem, paying particular attention to their efficiency at classification time (goal 5).

Some techniques are 'non-statistical' in that they deliver the decision function $D(.)$ directly, ie without generating the $a$ posteriori (or conditional density) distributions. Although these methods immediately fail to satisfy us (goal 4), it will be instructive to consider them briefly.

### 2.4.1 Non-statistical methods

In [Minski88, ch.12] the following non-statistical pattern recognition methods are discussed:

- Linear discriminants via:
  - the 'Best plane'
  - the 'Perceptron' procedure
- the 'Bayes statistical' procedure

- 'nearest-neighbour' procedure

Linear discriminants (hyperplanes)

The first group can effectively be dismissed on sight. They all involve partitioning the pattern space with a single hyperplane (in the two class case), and so by definition cannot deal with linearly inseparable problems. All ultimately generate a linear discriminant function of the form:

$$D(x) = \begin{cases} 1 & \text{if } v \cdot x > 0 \\ 2 & \text{otherwise.} \end{cases}$$

Here, $x$ is the augmented pattern vector; it simply has an extra component, set to one, and avoids the need for a threshold constant.\(^6\) $v$ is the $d + 1$ coordinate (but having only $d$ degrees of freedom) weight vector, which specifies the hyperplane and must be determined from the training procedure. It is in the means of determination of $v$ that the three methods differ.

'Best plane' is not a method at all — it is simply defined to be the plane which achieves the lowest error-rate in any circumstances, and no practically efficient method exists for finding it [Minsky88, §12.2.3]. The perceptron algorithm will converge on the perfect plane (ie no errors) if such a plane exists, otherwise it is unpredictable. The Bayes method (see [Minsky88, §12.3]), is guaranteed to find the Best Plane, even in the linearly inseparable case, providing the distributions of the individual components of $x$ are class-conditionally independent — ie the within-class co-variance matrix is diagonal. This is a very strong assumption indeed.

The 1st Nearest Neighbour method — 1-NN

The nearest neighbour is a powerful and much used classifier. See [Devijver82] for a full description of the method. It can cope with multi-modal pattern spaces and linear separability does not concern it. It can be shown that for a sufficiently large sample size, the 1-NN rule achieves an error-rate that is no more than twice the ideal Bayes rate (see [Devijver82, §3.8]). The principal draw-back to the method (apart from the fact that it does not yield PPs) is its efficiency at classification time. In its crude form, the classification of each new pattern (pixel in our case) requires a search through the whole of the training sample, $L$, calculating a $d$-dimensional distance measure to find the closest pattern to the new point.

---

\(^6\)Strictly a notational convenience; it is not adopted in the remainder of this thesis.
2.4. CONVENTIONAL PATTERN RECOGNITION METHODS

The basic form of the algorithm can be improved however. For example the exhaustive search can be mitigated by the use of a ‘branch and bound’ algorithm [Fukunaga75]. Alternatively, the problem may be viewed as one of constructing/accessing a Voronoi diagram. See eg [Edelsbrunner87, §13.2.1], where a data structure is described which permits access in \( O(\log N_L) \) time, where \( N_L \) is the size of the training sample. Unfortunately, this structure requires \( O(N^{d/2}) \) storage and \( O(N^{(d+1)/2}) \) time to construct. In our case, typically \( N \approx 65000 \) and \( d \approx 18 \); this approach is impractical to say the least.

More realistically, a condensing algorithm can be used to substantially reduce the size of \( L \) (see [Devijver82, §3.14]). The algorithm is founded on the observation that patterns at the center of class homogeneous clusters are unnecessary for correct classification and can therefore be removed without loss of accuracy. See also [Bryant89] for further efficiency refinements to the 1-NN classifier.

2.4.2 Probability density function (p.d.f) estimation techniques

Recall that to obtain the PP distributions \( D(Y|X=x) \) it is sufficient to estimate the prior probability vector \( D(Y) \) and the conditional density distributions \( d(X|\alpha) \) for each class in \( \Phi \), from \( L \). Of these, the former presents no computational difficulty (though ensuring a training sample accurately reflects the priors can be a problem), so this section considers a number of possible means for estimating the latter. We dismiss immediately the use of parametric methods (eg the multivariate Gaussian) as being over simplistic for such a complex pattern space (eg it assumes the within-class pattern space is unimodal; also, the method is not particularly cheap at classification time, when the pattern space dimensionality is high).

In [Devijver82, §A.3] the following non-parametric methods for p.d.f estimation are given:

- Histogramming
- Parzen estimator
- K-th nearest-neighbour method

Histogramming

This involves the partitioning of \( \Omega \) into a pre-determined number of ‘bins’, which in the pattern space represent segments, squares, cubes or hypercubes depending on the dimensionality \( d \). Each training sample pattern then increments the count of the bin associated with the area of the pattern space in which it falls. The density is assumed to be constant within each bin and is straightforwardly estimated from the proportion of the total sample falling into a bin, divided by the bin’s volume.
The method works well for low-dimension spaces, but the storage problems encountered in higher dimensions are obvious. There are also problems in choosing the granularity of the histogram as this will be influenced by the size of the training data available and the dimensionality of the problem.

**Parzen Estimator**

The essence of this method is that each pattern in the training sample is assumed to make a contribution to the density at \( x \) and also in the region around it. See fig. 2.6. The regional influence is modelled by a *kernel* function, the common choice being a Gaussian distribution on the Euclidean distance from the pattern. The spread of the kernel must be decided in advance, and determines the 'spikeiness' of the final p.d.f. Selecting a smaller spread than is justified by the sample size results in a spiky distribution and will generate optimistic (dishonest) PPs (i.e., \( P(Y=\alpha|x) \) will be over-confidently close to zero or one). Conversely, too large a spread will smooth out significant detail producing PPs which are honest, but not fully refined on the available data.

At classification time the density of the \( \alpha \)th class at a new point is determined by summing the contributions from, in theory, *all* class \( \alpha \) patterns in \( \mathcal{L} \). In practice, however one can assume a pattern's contribution to be zero beyond a certain distance (e.g., two standard deviations for the Gaussian kernel). In principle the Euclidean distance to all patterns must be calculated to determine whether it is in the sphere of influence, though as for the NN methods this can doubtless be improved upon.

Sample condensing *cannot* be applied as this would destroy the density information. One could, however, envisage an editing scheme which replaced those patterns with equal \( x \) vectors by a single, weighted point. However, in real, noisy imagery, it is unlikely that
many, if any such perfect hits would be found. One may consider extending the scheme, looking for very similar points. This rapidly turns into unsupervised clustering which is suggested by many authors (eg [Patrick72],[Minsky88]) as a means of dealing with multi-modal pattern spaces, prior to applying piece-wise, parametric assumptions. It is feared however that there will be considerable practical and theoretical difficulties in applying any form of un-supervised learning in such a voluminous and ‘ill-behaved’ pattern space as our \( \Omega \); consider, how many ‘things’ (significantly different 9 \( \times \) 9 pixel patterns) are there likely to be in a complex image? In effect, the problem is grossly under-constrained.

The k-th Nearest Neighbour method

The training sample is planted, conceptually at least, in \( \Omega \) as for the 1-NN method. The probability density at a new point, \( x \) is then estimated by finding the smallest hypersphere around \( x \) which just contains k sample patterns. In crude implementation the patterns must be ranked by distance from \( x \) and the kth closest selected. See fig. 2.7. The distance to

![Figure 2.7: kth Nearest-Neighbour density estimation.](image)

this kth point is the radius \( r \) of the hypersphere, from which the hyper-volume may be calculated. Since this sphere contains a known \( k \) of the total from each class, the density can be calculated directly. Similar remarks as were made for the choice of the kernel width in the previous method, apply to the choice of \( k \) here. However, here the problem is less severe as the method automatically adapts to some extent; in sparse regions of \( \Omega \) the density is ‘averaged’ over a large volume and in dense regions a much smaller volume is involved, permitting the resolution of the fine detail in the distribution; this is not the case with the Parzen estimator. A value of \( \sqrt{N_\alpha} \) is suggested ([Devijver82]) for \( k \), where \( N_\alpha \) is the number of the class \( \alpha \) patterns in the training sample.

The computational burden of this method in its raw form is rather serious since, in addition to calculation of distance to each sample point, these distances must also be ranked
in order to find the $k$th closest. It is possible to construct a $k$th order Voronoi diagram which permits access to the $k$th nearest pattern in $O(\log N_C)$ time, but as with the 1-NN method above, the preprocessing and storage complexities are prohibitive - both are $O(N_C^{(d+1)})$ [Edelsbrunner87, §13.3]. See [Chazelle85] for further recent work on the $k$-th order Voronoi diagram.

As with the previous method, condensing of the sample is not permitted.

### 2.4.3 Summary

A number of the 'standard' methods have been considered as candidates for tackling the problem which was defined in §2.2.2. These are summarized in the following table, which also includes other 'elementary' methods. The final entry gives the 'required' characteristics of the solution. For the purpose of this table a non multi-modal pattern is assumed to have a well behaved epi-spherical distribution — fitting the Gaussian model. Where a 'no' appears in a 'can deal with' column then the other entries assume that difficulty does not exist.

<table>
<thead>
<tr>
<th>method</th>
<th>Can deal with—</th>
<th>Time complexity</th>
<th>Yields PPs?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Muli-modal</td>
<td>Linearly insep.</td>
<td>Training</td>
</tr>
<tr>
<td>GML</td>
<td>No</td>
<td>Yes</td>
<td>Mod.</td>
</tr>
<tr>
<td>Perceptron</td>
<td>Yes</td>
<td>No</td>
<td>Mod.</td>
</tr>
<tr>
<td>Box classifier</td>
<td>No</td>
<td>Yes</td>
<td>Low</td>
</tr>
<tr>
<td>1-NN + Editing</td>
<td>Yes</td>
<td>Yes</td>
<td>Mod.</td>
</tr>
<tr>
<td>k-NN Density</td>
<td>Yes</td>
<td>Yes</td>
<td>Low</td>
</tr>
<tr>
<td>Parzen Estimator</td>
<td>Yes</td>
<td>Yes</td>
<td>Low</td>
</tr>
<tr>
<td>REQUIRED</td>
<td>Yes</td>
<td>Yes</td>
<td>Not crucial</td>
</tr>
</tbody>
</table>

It is interesting to note that as one considers ways of improving the efficiency of methods such as histogramming, k-NN and the Parzen estimator, the distinctions between them become blurred, and proposed solutions begin increasingly to look like the probability tree approach to be developed in the following section.

### 2.5 Lapwing's probability tree approach

In this section the solution adopted for Lapwing is described.

Probability trees were not discussed in the previous section; this was simply to avoid repetition here. Classification trees (where a categoric classification rather than class PPs are
required) are being used increasingly on pattern recognition problems. "The tree classifier is probably the best engineering solution to classification, especially when the feature dimension, or the number of pattern classes is large, as experienced in remote sensing" [Chen87].

It is possible to regard classification trees as a means of inducing rules for use in an expert system, from a collection of examples. The ID3 algorithm [Quinlan86] achieves precisely this and the method to be described here owes much to this algorithm.

Classification trees have been used in machine vision, for identifying types of chocolate on a conveyor belt from a number of shape primitives (ie an object based approach) [Shepherd83]. Applications of classification trees in remote sensing have already been mentioned in the introductory chapter. These involved classification on the basis of single multispectral pixel data only. They appear not to have been used for the purpose of deriving probabilistic assessments in remote sensing.

For elementary introductions to classification trees see [Quinlan79], [Hart86] or [Thompson86]. Additional theoretical material may be found in [Kurzynski83], [Yung83] [Li86] and [Wang87].

It is worth remarking on the spurious similarity between the classification tree approach and that of sequential pattern recognition [Fu69]. The methods appear similar in that both involve decomposing the classification task into a sequence of decisions. However, the relationship is spurious since the latter technique is concerned with ordering the acquisition of features, which are assumed to have some 'cost' associated with them (eg a measurement may have to be taken manually). The choice of which feature to acquire next is made 'on the fly' at classification time, and may involve substantial computational effort; once a new feature has been obtained it will be exploited to the full (considered jointly with previously acquired features) and further features requested only if a sufficiently confident decision is not yet possible. In the classification tree method however, all features are assumed to be present from the outset and the aim is to obtain the best possible decision with minimal computational effort. Thus the tree is constructed in advance and only simple rules need be evaluated at classification time in order to decide which branch to follow.

2.5.1 Constructing a probability tree

A probability tree is similar to a classification tree, except that it is used to return a probabilistic rather than categoric assessment of class membership. Whereas the methods mentioned in §2.4.2 were aimed at delivering the class-conditional density, probability trees deliver PPs directly, and in fact are ill-suited to density estimation due to the difficulty in calculating the 'subset volumes'.

Possibly the most comprehensive work on the theoretical aspects of classification/probability
The view of probability trees taken in this thesis is that they represent a *hierarchical partition* of a pattern space. Each node, \( t \), represents a subset of \( \Omega \), \( \Omega_t \) and includes an estimate of \( D(Y|X \in \Omega_t) \). A predicate function, \( D_t(x) \in \{ \text{TRUE, FALSE} \} \) partitions the subset \( \Omega_t \) into \( \Omega_{t,l} \) and \( \Omega_{t,r} \) (ie 'left' and 'right' subsets) as follows (assuming \( x \in \Omega_t \)):

\[
\text{if } D_t(x) = \text{ TRUE then } x \in \Omega_{t,l} \text{ else } x \in \Omega_{t,r}.
\]

(2.5)

We are thus considering *binary* trees. Figure 2.8 shows an example tree and the form of the partition it may represent.

![Figure 2.8: A probability tree and equivalent partition.](image)

A tree is constructed with reference to the training sample of classified patterns \( \mathcal{L} \). Although we shall refine the following criterion in §2.5.3, for now we consider our informal aim is for each subset \( \Omega_t \) to contain cases predominately from only one class, ie for \( P(Y=\alpha|X \in \mathcal{L}_t) \approx 1 \) or 0, for each \( \alpha \in \Phi \), where \( \mathcal{L}_t = \Omega_t \cap \mathcal{L} \), that is, \( \mathcal{L}_t \) is the subset of the training data which falls into \( \Omega_t \).

The tree is constructed from the top down. Associated with each node is the set \( \mathcal{L}_t \). The problem is to find, at each node, a predicate function \( D_t(x) \) to divide \( \mathcal{L}_t \) into \( \mathcal{L}_{t,l} \) and \( \mathcal{L}_{t,r} \), so as to reduce the *average split impurity* w.r.t. class membership. The average split impurity, \( I(t) \), is the weighted average of the *node impurities* of the two descendants, ie

\[
I(t) = P(\mathcal{L}_{t,l}|\mathcal{L}_t).i(t,l) + P(\mathcal{L}_{t,r}|\mathcal{L}_t).i(t,r)
\]

(2.6)

where here, \( P(\mathcal{L}_{t,l}|\mathcal{L}_t) \) is used non-strictly to represent the proportion of cases falling into \( \mathcal{L}_{t,l} \) out of the total in \( \mathcal{L}_t \), and \( i(t) \) is the *node impurity* function:

\[
i(t) = \prod_{\alpha \in \Phi} P(Y=\alpha|X \in \mathcal{L}_t)
\]

(2.7)

---

The book is primarily concerned with classification trees; probability trees are discussed specifically on pages 121–126.
2.5. LAPWING'S PROBABILITY TREE APPROACH

again using "P" to represent proportion.

The impurity function 2.6 is known as the Gini criterion function [Breiman84, §4.3.1]. An alternative function is the node entropy:

\[ \sum_{\alpha \in \Phi} P(Y = \alpha | X \in \mathcal{L}_t) \ln P(Y = \alpha | X \in \mathcal{L}_t). \] (2.8)

Limited experimentation has given no reason for choosing one or the other criteria, and so we follow Breiman et al.s in adopting the former due to its simplicity. It is important to appreciate that \( \mathcal{L}_t \) will not generally cover \( \Omega_t \), and thus there will be many possible binary partitions of \( \Omega_t \), i.e. different predicates, which will produce identical \( \mathcal{L}_{t,l} \) and \( \mathcal{L}_{t,r} \), and thus identical \( I(t) \). But are they all equally good splits? This will be considered shortly.

The tree growing procedure is recursive. Having split a node \( t \) into \( (t, l) \) and \( (t, r) \) the procedure is repeated with each of \( (t, l) \) and \( (t, r) \) to produce \( (t, l, l) \) etc. Ideally, the process should continue until each \( \mathcal{L}_t \) becomes completely pure, even if this means they contain only one member! Statistically meaningless splits are identified in a subsequent pruning phase to be described in section 2.5.6. The tree growing process is expensive, however, and so is terminated when a node contains less than \( N_{\text{min}} \) cases from any class. \( N_{\text{min}} \) has been set to 30 in this application.

2.5.2 The splitting rules

Lapwing is able to search for a partitioning hyperplane in any orientation. We first express \( D_t(x) \) in the form

\[ D_t(x) = [S_t(x) < c_t] \] (2.9)

where \( S_t \) is a scalar function of \( x \) and \( c_t \) is a threshold value (square brackets will be used to indicate a predicate). To permit the specification of general hyperplanes in \( \Omega \) we can define \( S_t(x) \) as

\[ S_t(x) = \frac{v_t \cdot x}{|v_t|}. \] (2.10)

From vector algebra, \( S_t(x) \) is the signed perpendicular displacement of \( x \) from the hyperplane which passes through the origin and has normal vector \( v_t \). The sign of \( S_t \) indicates which side of the hyperplane the pattern falls.

Recalling that \( x \) represents a square neighbourhood of pixels, it is clear that \( v_t \) can be regarded as a square convolution operator, familiar in image processing. It is known that many image processing operations (for example edge/line/point filters and smoothing, (see eg [Gonzalez77]) can be performed with convolution operators and this gives intuitive support for the chosen form of \( S_t \). Note also that [Laws80] (see §1.4.3) successfully used
combinations of simple convolutions for texture classification. Conveniently, it is precisely this type of operation at which the LAP excels — convolutions are supported directly by PPL.\footnote{PPL is the high-level language used to program the LAP (see §3.2).}

### 2.5.3 An additional criterion for resistance to simple bias - $Em''(t)$

It was remarked in §2.5.1 that there are likely to be many different partition surfaces which produce identical split impurities. The relevance of this can be readily appreciated if one considers fig. 2.9 (a), which shows a two class pattern space. A criterion function based on Gini alone would make no distinction between splits $D_1$ and $D_2$, yet clearly $D_2$ is to be preferred. Although both give perfect classification of the the training sample, $D_1$ could be described as 'fragile', since only a small shift of the population, caused by some bias of the training sample perhaps, could produce large classification errors. On the other hand $D_2$ would tolerate a relatively large shift of the pattern space.

In fig. 2.9 (b) the Gini criterion would select $D_3$ since this gives perfect class separation; intuition and caution however favours $D_4$ followed by $D_5$. Note that the 'least mean-square-error' procedure (see [Devijver82]), or any criterion based on maximising the distance between class means would also select $D_3$.

#### Outline of derivation

The above discussion seems to suggest that when a partition is to be built up hierarchically, as it is in a probability tree, then each partitioning surface should be selected so as to pass through \textit{unpopulated} regions of the pattern space, avoiding clusters, even if this gives poorer class separation at that stage. The following sections derive a criterion function which is used by Lapwing to assess the 'fragility' of a partitioning surface; it is intended to be used in conjunction with the impurity criterion $I(t)$ already described.

![Figure 2.9: Which are the better splits?](image-url)
the aforementioned criterion, that to be developed is unsupervised — it does not involve the pattern classifications. The approach is first to define a plausible model of the way that patterns in this application are likely to shift en mass, and then to derive an expression for the proportion of the sample patterns within a node $t$ expected to cross a proposed partition surface. This expectation will be called $Em(t)$ and forms the basis of the new criterion. To this will be added two pragmatically motivated embellishments to deliver $Em'(t)$ and finally $Em''(t)$.

**Model**

Recall that we are concerned with image classification and that the components of $x$ represent pixel brightness. In this domain the simplest and most common form of bias encountered is a variation of light intensity across the subject, or a variability in the gain of the image capturing system between images. We will assume the effect is additive rather than multiplicative, and that it is constant for the pixels of one neighbourhood. The model of bias proposed then, is simply that a pattern $x$ may become transposed to the pattern $x'$ by

$$
x' = x + bU,
$$

$$
U = \frac{1}{\sqrt{d}} \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}
$$

(2.11)

The scaler $b$ is the magnitude of the bias in a particular image and will vary from image to image (or even within an image), but assume that within a particular application domain it is normally distributed, mean zero, standard deviation $\sigma$, ie,

$$
p(b) = \phi_0,\sigma(b)
$$

(2.12)

$$
P(b \leq b') = \Phi_0,\sigma(b')
$$

(2.13)

$$
P(b > b') = 1 - \Phi_0,\sigma(b')
$$

(2.14)

where $\phi$ is the Normal probability density function, and $\Phi$ is the cumulative Normal probability function.

**Derivation**

The following derives an expression for the expected proportion of patterns that will cross a given partition due to the bias model specified above.

When projected by the function $S_t$, the effect of bias is to transform $S_t(x')$ as follows

---

9We are saying that the recognition system should embody a degree of invariance to changes in lighting; this relates to the group theoretic view of pattern recognition which is proposed by [Wilson88].
\[ S_t(x') = S_t(x + bU) = S_t(x) + bS_t(U) \]  
(2.15)

(since \( S_t \) is linear), and so the partitioning predicate in terms of \( x \) and \( x' \) will be

\[ D_t(x) = [S_t(x) < c_t] \]  
(2.16)

\[ D_t(x') = [(S_t(x) + bS_t(U)) < c_t]. \]  
(2.17)

We are interested to know for what values of \( b \), \( x' \) crosses the decision boundary, i.e. when is \([D_t(x) \neq D_t(x')]\)? By inspection

\[ [D_t(x) \neq D_t(x')] = \begin{cases} 
[c_t - S_t(x) < bS_t(U)], & c_t - S_t(x) \geq 0 \\
[c_t - S_t(x) \geq bS_t(U)], & c_t - S_t(x) < 0 
\end{cases} \]  
(2.18)

and re-arranging gives

\[ [D_t(x) \neq D_t(x')] = \begin{cases} 
[b > \frac{c_t - S_t(x)}{S_t(U)}], & c_t - S_t(x) \geq 0 \\
[b \leq \frac{c_t - S_t(x)}{S_t(U)}], & c_t - S_t(x) < 0. 
\end{cases} \]  
(2.19)

Combining 2.13 and 2.14 with 2.19 gives the probability that \( x' \) crosses the decision boundary:

\[
P([D_t(X) \neq D_t(X')] | X=x, t) = \begin{cases} 
1 - \Phi_{0,\sigma} \left[ \frac{c_t - S_t(x)}{S_t(U)} \right], & c_t - S_t(x) \geq 0 \\
\Phi_{0,\sigma} \left[ \frac{c_t - S_t(x)}{S_t(U)} \right], & c_t - S_t(x) < 0. 
\end{cases} \]  
(2.20)

The additional conditioning on \( t \) is intended as an informal reminder that the expression is specific to a given node, i.e. particular \( v_t \) and \( c_t \).

It is becoming tedious to deal with two separate cases, so let us define a symmetrical version of \( \Phi \), call it \( \tilde{\Phi} \), as follows

\[
\tilde{\Phi}_{0,\sigma}(b) = \begin{cases} 
1 - \Phi_{0,\sigma}(b), & b \geq 0 \\
\Phi_{0,\sigma}(b), & b < 0. 
\end{cases} \]  
(2.21)

Using \( \tilde{\Phi} \) and writing \([D_t(X) \neq D_t(X')]\) as simply ‘miss-split \( X \)’, we can re-write 2.20 as

\[
P(\text{miss-split} \ X | X=x, t) = \tilde{\Phi}_{0,\sigma} \left( \frac{c_t - S_t(x)}{S_t(U)} \right) = \tilde{\Phi}_{0,\sigma}S_t(U)(c_t - S_t(x)). \]  
(2.22)
Equation 2.22 gives the probability of a single particular \( x \) crossing the decision boundary due to simple bias as defined by 2.11 to 2.12. What we now require is the expected proportion of the pattern population (within \( \Omega_t \)) being miss-split by a given \( D_t \) (ie a given \( S_t \) and \( c_t \)). We will call this proportion \( Em(t) \). This will of course depend on the total distribution of \( x \) within \( \Omega_t \). Let \( p(x|t) \) be this within node density\(^{10}\). Then:

\[
Em(t) = \int_{\forall x \in \Omega_t} P(\text{miss-split } X|X=x,t).p(x|t)dx
\]

and with 2.22 gives:

\[
= \int_{\forall x \in \Omega_t} \Phi_{0,\sigma S_t(U)}(c_t - S_t(x)).p(x|t)dx.
\]

We now substitute \( s = S_t(x) \) and separate the integration into two stages:

\[
= \int_s \int_{\forall x:S_t(x)=s} \Phi_{0,\sigma S_t(U)}(c_t - s).p(x|t) dx ds
\]

\[
= \int_s \Phi_{0,\sigma S_t(U)}(c_t - s) \int_{\forall x:S_t(x)=s} p(x|t) dx ds.
\]

Now the inner integral is clearly \( p(s|t) \), thus we finally obtain:

\[
Em(t) = \int_s \Phi_{0,\sigma S_t(U)}(c_t - s)p(s|t)ds.
\]

The \( Em \) criterion depends on \( v_t \), \( c_t \) and \( \Omega_t \), each of which are specific to a node \( t \), and the parameterization is intended to convey this.

Note that we have carefully avoided referring to \( Em(t) \) as the likelihood of misclassification; it is the estimated proportion of miss-split sample patterns at a single node alone.

**Histogramming for \( p(s|t) \)**

The following sub-sections will develop further refinements to the \( Em \) criteria, however it will be helpful to first explain how the quantity \( p(s|t) \) is obtained.

Recall that (scalar) \( s \) is a function both of a given pattern vector \( x \) and of the convolution vector \( v_t \) (see 2.10). The quantity \( p(s|t) \) could be more fully written as \( p(S_t(x) = s|X \in \Omega_t) \). Note that this quantity does not involve \( c_t \). In practice the required probability density is obtained by forming a histogram of \( S_t(x_i) \) values at a suitable quantization, out of a random selection of the \( x_i \) vectors in the current node \( t \). The histogram can then be normalized by dividing each bin by the number of sample-points in the node to yield the required probability distribution.

\(^{10}\)Formally \( p(x|t) = p(X=x|X \in \Omega_t) \)
The $Em(t)$ criteria along with the modified versions which follow, as as well as the node impurity $I(t)$ \(^{11}\) can all be obtained from the distribution $p(s|t)$ alone; thus for a given $v_t$, the best $c_t$ may be selected from the histogram of $S_t(x)$ values alone. There is further explanation in §3.1.1.

**A refinement**

The development of $Em$ has thus far been statistically precise (given the model); unfortunately, the following modifications are heuristic.

As defined by equation 2.27, $Em(t)$ will be zero if $v_t$ is perpendicular to $U$, no matter how close the plane is to sample clusters. This is because the model of bias assumes that the patterns will shift only in the direction of $U$. In practice there may well be other variability (changes in contrast, shape distortions etc), and so we would like $Em(t)$ to reflect some small degree of ‘sample phobia’ even when the hyperplane is parallel to $U$. This may be achieved by adding a constant component $\sigma'$ to the spread of $\Phi$; we thus define a modified version of $Em(t)$ called $Em'(t)$ as follows:

$$Em'(t) = \int_{s=-\infty}^{s=\infty} \Phi_{0,\tau}(c_t - s)p(s|t) \, ds$$  \hspace{1cm} (2.28)

where

$$\tau = \sqrt{(S_t(U)\sigma)^2 + \sigma'^2}.$$

The setting of $\sigma$ should in principle be obtained from the standard deviation over a number of similar images, of the mean intensity values. In practice however the value may have to be ‘guessed’. No theoretical basis can be suggested for the setting of $\sigma'$. For the tests presented in chapter 5, $\sigma$ and $\sigma'$ were set to 16 and 8 respectively.

**Another refinement**

A more serious problem remains, which would limit the direct use of this criterion function for partition selection; it is that $Em'(t)$ can always be minimized by placing the partitioning surface way outside the space occupied by the sample, so that all patterns fall to one side. See fig. 2.10. This would be a highly robust though hardly useful partition! We need in some way to penalize a split which is highly asymmetrical — ie which places most of the training patterns to one side of it.

We construct a modification by suggesting that when $p(s|t)$ is Gaussian distributed, then we require $Em'(t)$ to express no preference about the placement of the partition (ie setting

\(^{11}\)In fact, to calculate $I(t)$ we require the class conditional densities $p(S_t(x)=s|x \in \Omega_t, Y=\alpha)$ for each $\alpha$, from which the total density is easily obtained.
of $c_t$, that is, we wish the modified function to yield a constant value in this situation, regardless of the setting of $c_t$.

The following solution achieves the above only approximately; it assumes that the spread of the pattern (sub) space under consideration is large compared with $\tau$ (the spread of $\hat{\Phi}$). If this is the case then $Em'(t)$ yields a value which is proportional to the pattern density as projected by $S_t$, at the point $c_t$.

Thus, if we divide by the density of a Gaussian which has been fitted to the observed $p(s|t)$ at the proposed $c_t$ value, then it is clear that if is $p(s|t)$ is Gaussian, then this quotient will be one whatever the value of $c_t$. When $p(s|t)$ is platykurtic (as in fig.2.10), however, the quotient will have its minimum value where the distribution 'dips' — i.e., at the point where we would intuitively prefer the split to be made.

Thus, we fit a Gaussian so that the proportion, $r$, of patterns to the left of the candidate split, matches the proportion expected for the Gaussian. The Gaussian can be fitted so that the proportion, $r$, of patterns to the left of the candidate split, matches the proportion expected for the Gaussian. Thus consider:

$$Em''(t) = \frac{Em'(t)}{\phi_{0,1}(\Phi_{0,1}^{-1}(r))} \quad (2.29)$$

where

$$r = P([s < c_t]|t)$$

and $\Phi_{0,1}^{-1}$ is the inverse of $\Phi_{0,1}$. The expression $\Phi_{0,1}^{-1}(r)$ delivers the number of standard deviations from the mean of a Gaussian necessary to achieve the observed cumulative proportions, and this may be passed directly to $\phi_{0,1}$ to deliver the density at the point of splitting. The quantity $r$ is readily available from the histogram of $s$, and the composite function $\phi_{0,1}(\Phi_{0,1}^{-1}(\cdot))$ can be tabulated for efficiency.

When $r$ is not negligible then the modified criterion function will slightly favour a split at the mode of the Gaussian — which seems as good or bad as anywhere else, for such a structure-less projection.

Discussion

Equation 2.29 in conjunction with 2.28 is the proposed additional criterion of miss-split likelihood. The function $\hat{\Phi}_{0,\tau}(b)$ is sketched in fig. 2.11. This function shrinks about $b = 0$ depending on the value of $\tau$. Recall that $\tau$ depends heavily on $S_t(U)$, which has a familiar interpretation as the cosine of the angle $\theta$ between the convolution vector $v_t$ and the constant vector $U$. $\hat{\Phi}$ is at its broadest when $\theta = 0^\circ$ or $180^\circ$ and shrinks to its slimmest at $\theta = 90^\circ$ or $270^\circ$ when the spread is governed by $\sigma'$ only. $\hat{\Phi}_{0,\tau}(b)$ is approximately zero when $|b| > 2\tau$.

\[^{12}\text{An alternative, but not equivalent method would be to fit a Gaussian with equal mean and SD as } S_t \text{ within } t.]$
The derived criterion function is intuitively reasonable since it is sensitive to the density of patterns 'close' to the partitioning hyperplane, with very close points being more significant than those less close. Further, the rate of decrease in significance is related to the orientation of the hyperplane, with 'smooth and threshold' splits incurring a greater penalty than 'differencing' splits (e.g., edge operators) which are less sensitive to ambient lighting.\footnote{In image processing terms our $v_i$ is simply a convolution operator. There are two broad classes of such operators which may be called 'smoothing' and 'differencing' operators. The former class is typically used for removing noise and its components will have the same sign. The latter are used for detecting lines, edges or points and will be constructed so that their components sum to zero; they thus have the desirable property of giving a zero response when placed over a homogeneous region of an image, and that their response will be unaffected by a uniform additive change in the image intensity.} Smoothing operators do not share these properties. The fact that this intuitively reasonable result has been analytically derived (as far as $E_m$ at least) from a defined (if simplistic) model is satisfying.

Note that $E_m''(t)$ is an unsupervised criterion function since it does not require knowledge of the true class.
2.5.4 The total cost-function used by Lapwing

The complete cost-function used by the Lapwing learning program as the heuristic by which it will search for the best split at each node, is a weighted sum of three criteria as follows:

\[ C_t = k_I I(t) + E m''(t) + k_z Z(t). \] (2.30)

\( I(t) \) is the Gini criterion, \( E m''(t) \) is the derived criterion for robustness, and \( Z(t) \) is a measure of the complexity of the convolution \( v_t \) defined as the number of non zero elements it contains. The inclusion of this latter term produces a slight pressure towards simpler convolutions. This is relevant as the LAP does not have a large program store, and the PPL compiler is clever enough not to generate code for zero components of a convolution.

The setting of the weighing factors \( k_I \) and \( k_z \) is problematic. Since each of the three terms in the cost-function represent incompatible quantities, it has not been possible to make any theoretical justification for particular settings. Values of \( k_I = 5 \) and \( k_z = 0.005 \) have been found satisfactory. Because of the arbitrary construction of the final cost-function it is not possible to make meaningful claims concerning the optimality of the selected partitions.

2.5.5 Searching — genetic algorithms

The previous sections have developed the criterion function 2.30 which may be used to assess the quality (in fact the ‘poorness’) of a proposed split at a given node. Any proposed split will be parameterized by a convolution vector \( v_t \) and a threshold \( c_t \) (see 2.9 and 2.10). Clearly therefore we need to find those values of \( v_t \) and \( c_t \) which minimize the criterion function 2.30. Since it seems unlikely that the minimization of this function would succumb to analysis, a search procedure has been adopted with the given cost-function acting as the heuristic to guide the search. The search strategy used is taken from a class generally referred to in the literature as a Genetic Algorithm.\(^{14}\) The key papers on the subject are due to Holland. [Holland73], [Holland78]. He approaches the optimization problem as one of efficiently allocating trials of candidate solutions (assuming that the only significant resource usage is that of evaluating the cost-function). He shows that the processes of genetics as seen in sexual reproduction do in fact achieve this efficient allocation of ‘trials’.

Genetic algorithms are being applied to an increasing number of optimization problems which are characterized by a high dimension parameter space and a multi-modal, non-convex cost-function. Smith [Smith81], has used GAs for the calibration of an information driven model of US interstate migration patterns. In structural engineering, Goldberg [Goldberg86]

---

\(^{14}\) An alternative related method is simulated annealing (see [Kirkpatrick83], [Jeffrey86], [Siarry87] or [Davies87]) which takes its inspiration from statistical mechanics and the metallurgical technique of ‘annealing’ — hardening a metal by slow cooling.
CHAPTER 2. LAPWING - A NEW APPROACH

has optimized the members of a ten member plane truss. Several learning systems have been
developed including LS-1 [Smith83b] which has been applied to bet decision making in draw
poker, CS-1 [Holland83] which has been used in a maze walking problem and a medical
diagnosis system and RS-1 [Grefenstette86b] which has been used for robot navigation.
GAs have been applied directly to a pattern recognition problem by [Stadnyk87].

An excellent and up-to-date text is [Goldberg89], and a useful collection of articles may
be found in [Davies87].

A general purpose genetic algorithm

The algorithm to be described here is based on work by [Adams87] who in turn took in­
spiration from [Grefenstette86a] and [Holland73]. A number of modifications were made by
the author to produce a fully re-entrant Pascal procedure. The resulting package has been
named 'GAGA' (Genetic Algorithm for General Application) and is described in [Poole88c].

The application of genetic algorithms to the optimization of linear discriminant functions
was first reported in [Poole88a].

Genetic algorithms model the process of evolution. A candidate solution to the opti­
mization problem is regarded as an organism and represented as a chromosome built out
of a linear sequence of genes. A collection (typically ≈ 100) of these chromosomes form a
population. Development refers to the mapping of the chromosome representation onto the
organism. The fitness of an organism is assessed by a cost-function. Pairs of organisms are
permitted to re-combine to produce progeny via the genetic operation of cross-over. All new
chromosomes are subject to possible mutation and inversion. The algorithm is summarized
in fig. 2.12. The crossover operation involves a break occurring at exactly corresponding
points in the two chromosomes and their rejoining crosswise. The resulting progeny thus
inherit aspects of 'the solution' from both parents, with the potential for two successful but
partial solutions to be brought together in one organism.

Inversion involves two breaks occurring in a single chromosome and the chromosome
reconstituting itself with the central portion reversed. Since each gene is tagged with a
number which identifies the parameter for which it codes, inversion has no effect on the
organism actually produced by development; however, it permits genes that may operate in
a cooperative fashion to be brought closer together and so to be less likely to be separated
in future crossovers. In genetics this phenomenon is known as linkage.

Mutation is less important than crossover. The value of a randomly chosen gene is
randomly changed. Mutation serves to ensure that all points in the search space are reachable.

Development is the mapping from the GA search space of chromosomes onto the solution
Randomly initialize the population of chromosomes.

REPEAT
   FOR each chromosome in the population DO
      Develop into an organism (candidate solution).
      Evaluate cost-function for this organism.
   END.

Remove a given percentage of the poorest performers from the population.

Select pairs from survivors in proportion to their fitness to mate and produce new chromosomes by crossover.

Randomly mutate a few of the chromosomes in the population.

Randomly perform inversions on a few of the chromosomes in the population.

UNTIL best organism achieves desired cost OR a fixed number of generations have been simulated.

Figure 2.12: A genetic algorithm

space. Call these spaces $\mathcal{G}$ and $\mathcal{S}$ and the mapping $h$, i.e. $h : \mathcal{G} \rightarrow \mathcal{S}$. $h$ may be highly complex and may possibly incorporate a partial solution — by expressing some dimensions of $\mathcal{S}$ in terms of others, so reducing the dimensionality of $\mathcal{G}$ required to express all possible solutions, or by concentrating the range of $h$ onto known 'likely' regions of $\mathcal{S}$. Thus, it is in $h$ that prior intelligence on a problem may be encapsulated. A genetic algorithm moves through $\mathcal{G}$ in a particular way (discussion of which is, sadly, outside the scope of this thesis); by good or bad choice of $h$, these movements will be mapped onto intelligent or dumb movements in $\mathcal{S}$. GAs appear able to 'cope' with even poor choices of $h$, and this is perhaps their greatest strength — 'they reliably perform reasonably'. However, this does not alter the fact that a careful choice of representation ($h$) will speed the search. Just how a 'good' representation may be selected is a fertile ground for research.

The GAGA package has been re-coded into the C language ([Kernighan78]) and applied to the task of learning efficient client/host assignments in a distributed computing system ([Crowcroft89]) and to learning control parameters for a respiratory trigger ([Hall88]).

---

$^{15}$ $\mathcal{G}$ and $\mathcal{S}$ may be considered as analogies to the biological spaces of 'genotypes' and 'phenotypes' respectively.
CHAPTER 2. LAPWING - A NEW APPROACH

The genetic algorithm in Lapwing

The algorithm described in the previous section is completely general except for the cost-function and the 'develop' routine. The former has already been described — it is exactly that given in §2.5.4.

Recall that the key component of $D_t$ is the convolution vector $v_t$; for reasons that are discussed in the next chapter, the genetic algorithm is configured to search through the space of possible $v_t$ values only and does not concern itself with $c_t$. The value of $c_t$ is optimized within the cost function by an exhaustive (quantized) search for each value of $v_t$ that the GA proposes. This is explained further in §3.1.1.

Develop must generate a candidate solution (an organism) from a given chromosome. In our case the organism is the convolution vector $v_t$.

Various ways of mapping a chromosome onto $v_t$ can be envisaged; presently the simplest method is used — ie mapping individual gene values onto individual components of $v_t$. A more subtle method than this one-to-one mapping may give better results.

2.5.6 Tree validation and pruning

Recall that the purpose of growing a probability tree is to obtain estimates of the PP vector $D(\gamma|X \in \Omega_t)$ for each terminal node $t$. These vectors are readily available from the proportions, by class, of the training sample falling into $\Omega_t$. But to what extent are they to be believed? For example, if splitting is continued until each node contains only one pattern then all the components of the PPs will be 1 or 0!

The issue of when to stop splitting a node (or whether to remove a split from an existing tree) parallels the choice of histogram bin size, kernel width for the Parzen estimator, and the choice of $k$ in the kth-nearest neighbour method — too small a tree will not extract all the available information from the data, and too large a tree will result in optimistic (dishonest) PPs. The problem here is in some sense more severe since the partitions of the pattern space will have been tailored specifically for the particular training sample, which is not the case for the previously mentioned methods. The ever present danger of over-fitting the data is thus particularly acute.

\textsuperscript{16}And of course the underlying assumption is that when $t$ is a terminal node and $x \in \Omega_t$, then $D(\gamma|X \in \Omega_t) = D(\gamma|X=x)$ — ie the density is constant within a terminal node.

\textsuperscript{17}However, the $E_m(t)$ criteria mitigates somewhat against this since it discourages 'finely tuned' splits which dissect tight clusters.
2.5. LAPWING'S PROBABILITY TREE APPROACH

Use of a validation sample

Pruning is carried out on the basis of a separate (notionally independent) sample image (or stack of images) and training overlay \(^{18}\). This sample is fed into the tree and the estimates of \(D(Y|x \in \Omega_t)\) recalculated at each node. Note that, whether or not the tree is to be pruned, the use of PP estimates from the validation sample is to be preferred precisely because the partitions were not tailored for this particular sample.

Why prune?

We may wish to remove branches of a probability tree for any of the following reasons:

- Some nodes will have captured too few validation sample patterns to provide statistically significant PP estimates; some may have captured no patterns at all and so are clearly unusable.

- The improvement in impurity after several further splits may be so slight that the additional splits are not worth the increased tree complexity and increased computation at classification time.

- If the class distributions of the validation sample within a node differ greatly from the training samples, then we might wish to conclude that that portion of the pattern space is somewhat 'chaotic' and that neither sample can be considered as representative of the true population, and so retreat back to a larger subset.

This thesis does not tackle the latter situation, though the author suspects that standard statistical approaches exist.

A simplistic approach is adopted for case 1; all nodes which contain fewer than \(N_{vm}\) validation patterns are removed. \(N_{vm}\) is typically set to 50. Note that attempts to statistically justify a particular minimum node size are complicated by the already mentioned (§2.2.2) non-independence of the training sample patterns. The cross-validation approach to tree pruning ([Mabbet80]) is primarily intended for classification trees, but may also be relevant to probability trees; it is computationally costly however, involving the growth of multiple trees. Certainly more investigation is needed in this area.

\(^{18}\)Breiman et al recommend that all available sample patterns be pooled and subsets selected at random for training and validation. However, in our application it is recognized that the available images are unlikely to be truly representative of all images the system is likely to meet — there will be significant variation image to image. Thus, validating on a separate image is a 'tougher' test and will result in a more honest tree.
For case 2, Lapwing adopts the solution of Brieman et al — 'minimum cost-complexity pruning'. The technique is described fully in their book, and has been implemented without modification.

2.5.7 Classification

Once a tree has been grown, it may be applied to other images from the same domain, to generate a probability image for each class in \( \Phi \). A probability image is an image in which pixel intensities are proportional to the class PPs. Of course in an \( m \)-class problem it is only necessary to generate \( m - 1 \) probability images (except perhaps for ease of interpretation). In particular, it is only necessary to generate one image for a two class problem.\(^{19}\)

The procedure for classification is straightforward. At each pixel \( i \), the pattern vector \( x_i \) is extracted from the input image (or 'stack' of images if, for example, multispectral data is being used) and ‘dropped’ into the root of the tree. Decision functions \( D_i(x_i) \) are evaluated and the left or right branch followed accordingly until a terminal node is reached. This node will hold the vector \( D(Y|x_i) \), the components of which may be assigned to the pixel of the output probability images.

A categoric classification may be produced by applying the MAP decision rule. However, as shall be seen in chapter 4, the PPs are best preserved so that they may be refined by trained probabilistic relaxation.

If we assume that a balanced tree is produced (best case) and that all leaf nodes contain the minimum allowed number of sample patterns (worst case) then the complexity at the classification stage is clearly \( O(\log(N_C)) \) per pixel, where \( N_C \) is the size of training sample.

2.6 Summary

In this chapter the problem of image classification at the pixel level has been formulated in statistical pattern recognition terms. The solution to this problem requires a posteriori probability estimation in a high-dimensional pattern space. Conventional (non-parametric) approaches such as the kth nearest neighbour and the Parzen estimator have been rejected principally for their computational cost at classification time. A novel solution has been developed and is summarized in the following sub-section.

\(^{19}\)In fact the implementation to be described in the next chapter is restricted to two class problems — multiple trees are grown for multiple class problems. See §3.3.
2.6. SUMMARY

2.6.1 Summary of the Lapwing technique

An example image and pre-classified overlay provide a training sample; at each pixel in the training sample, a pattern vector is extracted that is made up of the pixel’s local neighbourhood — eg a 5 x 5 window. A probability tree is grown on the basis of this training sample as follows.

Tree growing

The root of the probability tree notionally holds the entire training sample; a search is made (see below) to find the best splitting rule according to a particular cost function which will divide this sample into two subsets, to form the left and the right sub-trees. This procedure is then recursively applied to each of the two sub-trees until a sub-tree is obtained which contains fewer than a pre-set number of sample-points from any class.

Searching

At each node, the search for the ‘best’ splitting rule is accomplished by a general purpose genetic algorithm in conjunction with a cost function which assesses the ‘quality’ of any proposed splitting rule (see below). The splitting rules are modelled as hyperplanes which may be at any orientation and displacement. The GA searches through the space of possible orientations, (ie values of $v_t$) only; the displacement (ie $c_t$) is optimized by exhaustive search within the cost function itself. The optimization process is thus split into two levels.

Cost function

The cost function by which a proposed split is assessed is composed from a weighted combination of three criteria functions:

- purity w.r.t sample-point class in the two sub-samples the split would produce;
- robustness of the split — this encourages splits which pass through sparse regions of the pattern space;
- simplicity (ie number of components that are zero) of the convolution vector which specifies the hyperplane orientation.

The first two are the most significant and can be calculated from the set of class-conditional histograms having as its horizontal axis the perpendicular distance from the hyperplane of given orientation and passing through the origin. Once these histograms have been constructed it is computationally cheap to evaluate the purity and robustness functions for a
split in the given orientation at any displacement. Thus the weighted sum of the two functions is computed at each quantised displacement along the histogram to find that which gives the lowest cost. This lowest cost is then added to the ‘simplicity’ value and returned to the GA as the cost for the proposed orientation.

**Pruning**

The probability tree that is produced will generally be very large and contain many worthless, insignificant or inefficient splits; these are identified and removed in a subsequent pruning phase which requires an independent validation sample (ie another example image and pre-classified overlay).

**Classification**

Subsequent, unclassified images may then be classified by ‘dropping’ the pattern vector extracted at each pixel into the root of the tree and applying the stored splitting rule to decide to which sub-tree it should belong. This is applied recursively until a leaf node is reached, which will hold an a posteriori probability vector to assign to the pixel.

The above package has been called Lapwing and further details of its implementation are described in the next chapter.
Chapter 3

Implementation details

The method of *a posteriori* probability estimation developed in chapter 2 has been implemented as a suite of Pascal [Wirth76] programs on an IBM PC-AT under the MS-DOS operating system.

In this chapter the implementation is described, including an option for performing the classification stage on a parallel array processor. It is not intended to provide user instructions or detailed internal documentation; the present system is a test-bed for research, and does not claim to offer a user-friendly interface. Lapwing involves three principle stages of processing — learning, pruning and classification. These stages are implemented in five separate programs: LEARNER, HONEST + PRUNE and CLASSY or TOPPL respectively. The final (classification) stage can be performed optionally on the PC using CLASSY or on the Linear Array Processor (LAP) after translation by TOPPL. The function of each program is summarized below.

- **LEARNER** — generates a probability tree from a training sample (image/overlay).
- **HONEST** — runs a validation sample through a given tree, modifying the tree statistics accordingly.
- **PRUNE** — removes uneconomic or unsound branches of the tree.
- **CLASSY** — produces a probability image from a test image and tree.
- **TOPPL** — generates a PPL classification program for the LAP from a given tree.

The inter-relationship of these components for the LAP option (ie using TOPPL and the LAP in place of CLASSY) is shown in fig. 3.1.

In the interests of clarity, the discussion in chapter 2 concerned multiclass problems; simply for practical reasons, Lapwing has been restricted in implementation to deal directly
with only two-class problems, though multiple classes can be tackled by growing multiple trees. This is discussed further in §3.3.
3.1 Implementation on a serial machine

3.1.1 Tree generation — LEARNER

Learner accepts as input a training image (or image stack) and a classification overlay and from these generates a probability tree. Non-terminal nodes of the tree represent linear discriminant functions which hierarchically partition the pattern space, with the terminals holding the \textit{a posteriori} probability assessment for the class in question.

Recall that the partitioning hyperplane is selected at each node by a genetic search algorithm (GA). The GA used on this project was developed out of [Adams87] and encapsulated into a general purpose package christened "GAGA" — Genetic Algorithm for General Application. This package is described in [Poole88c].

Any search (optimization) based algorithm requires repeated evaluation of the cost function. In our case, this is the function \( C_t \) defined in §2.5.4. The GA will typically require \( \sim 2000 \) evaluations of this cost function to optimize each partition (tree node), and a typical tree may contain \( \sim 200 \) nodes (before pruning). Thus the cost function will be evaluated \( \sim 400 000 \) times on each run of the program; it is essential therefore that \( C_t \) be implemented efficiently. The following sub-sections list some of the measures which were taken to achieve this.

Sub-sampling

In principle the evaluation of \( C_t \) requires the evaluation of \( (\nu_t \cdot x_i)/|\nu_t| \) for \textit{all} samples in \( L_t \). When the training sample is extracted from a 256 \( \times \) 256 image/overlay, this would involve \( \sim 65 000 \) cases at the root node and several hundred up until the penultimate leaves. In order to reduce this computational burden a randomly selected sub-sample of \( \sim 200 \) sample points is taken from each \( L_t \). As a subsidiary advantage, it is now possible to store the extracted pattern vectors \( x \) for this sub-sample in main memory, obviating the need to repeatedly extract them from the image at each evaluation of the cost function. Note that once the splitting function has been selected the entire sample \( L_t \) is split and passed on to the child nodes where a (different) sub-sample is again taken. Following the advice of Breiman \textit{et al}, the sub-sample is selected with equal numbers from each class \( (L_t \) permitting). Since this will in general distort the class prior probabilities of the node, it is necessary to weight each sample according to its class to compensate. The weighting is applied as the histogram of \( S_t(x) \) values is constructed (see below).
Efficient selection of $c_t$

For each value of $v_t$ that the GA proposes, a set of within-class histograms is constructed (and note that in implementation we have limited the number of classes to two), having $s$ (where $s = S_t(x), S_t(x) = \frac{v_t \cdot x}{|v_t|}$) on their horizontal axis and weighted within class sample counts on their vertical-axis. The weighting is necessary to compensate for bias introduced by the sub-sampling procedure. These histograms (when normalized) thus represents graphs of $p(s|t, Y = a)$. From its definition it can be seen that $s$ must lie between plus or minus $l\sqrt{d}$. So for example, if $N_w = 3, N_f = 2$ and $N_b = 1$ (see 2.2.1) this will give a pattern vector of 18 dimensions. With $l = 255$ (as is usual), $s$ will lie between plus and minus $255\sqrt{18} \approx 1082$. A quantization of 4 units has been found to be adequate, requiring a 543 bin histogram for each class.

The set of histograms is constructed to enable the efficient calculation of the two main components of cost, $I(t)$ (see 2.6) and $Em''$ (see 2.29, 2.28). The histograms depend on the hyperlane orientation ($v_t$), but not on its displacement ($c_t$).

Taking $I(t)$ first, this is a function of the proportions of sample points for each class falling on either side of the hyperplane, and these can be easily obtained, for any displacement $c_t$, by accumulating the points to the left and right of the given horizontal position in each of the class-conditional histograms.

The crucial equation involved in the calculation of $Em''$ is 2.28, which delivers the intermediate result $Em'$. The total density $p(s|t)$ is pre-computed from the class-conditional histograms and held in a tabulated form. The function $\hat{F}_{0,\tau}$ can also be pre-computed for the given $v_t$ and tabulated for values between plus and minus $2\tau$ — outside of this range we assume the function to be zero. The required integration can then be approximated by summing the tabulated densities around the $c_t$ value under consideration, each weighted by $\hat{F}_{0,\tau}$. That is, equation 2.28 becomes:

$$Em'(t) = h \sum_{j=2\tau/h}^{j=2\tau/h} \hat{F}_{0,\tau}(hj).p(S = c_t + hj|t)$$

where $h$ is the histogram bin size (eg 4). Figure 3.2 will aid visualization.

The refinement described by 2.29 simply involves dividing $Em'$ by a value which is a function only of the proportion of sample points falling to one side of the hyperplane, and this is also available from the tabulated total density.

Now, at each node, the optimization problem is to determine:

$$\min_{v_t,c_t} C(v_t, c_t)$$

where here we write $C_t$ (see §2.5.4) as $C(v_t, c_t)$ to emphasise the dependence of the function on $v_t$ and $c_t$. However, this can be re-written as two nested optimization problems, ie to
3.1. IMPLEMENTATION ON A SERIAL MACHINE

The outer minimization (w.r.t \( v_t \)) is handled by the GA, and the inner minimization is performed by a one dimensional exhaustive search within the domain of \( s \) — i.e., by stepping along the histogram(s) and calculating \( I(t) \) and \( Em'' \) at each position, as described above.

This division turns out to be efficient since it is necessary in any case to construct the class-conditional histograms in order to evaluate the cost function, and these histograms do not depend on \( c_t \).

Thus, the cost function seen by the GA is a function of \( v_t \) only.

Note that whilst the two expressions 3.1 and 3.2 are mathematically equivalent in the ideal, they may well in general yield different results when real (imperfect) optimization strategies are used. Limited experimentation suggests that the latter form gives better results, probably due to the high sensitivity of \( C \) to changes in \( c_t \).

Cost function caching

There is nothing preventing a GA from evaluating the cost of a point in its search space of chromosomes more than once. This will happen only rarely in the first dozen or so generations when there is still a large degree of variation in the population. However, once the population begins to converge on a solution it may occur frequently. In fact it has been
observed that by the 30th generation of some runs, ~ 80% of the evaluations were for points that had already been requested. Once this behavior begins to set in it is unlikely that any useful searching is being performed, however it is difficult to predict after just how many generations it will occur.

A substantial improvement in running time was achieved by maintaining a queue of (typically 40) of the previously costed \( v \)'s. For each call of the cost function a linear search is made (beginning with the most recent) of the queue. If a matching entry is found then the stored result is returned immediately; otherwise the cost function is evaluated and a new record created on the queue. If a record is 'hit’ when it is \( > \frac{3}{4} \) of the way down the queue, then it is re-inserted at the front to speed subsequent access and ensure it is not lost off the end of the queue.

Clearly the merits of using such a scheme depend on the relative computation times for searching an element of the queue and evaluating the actual cost function. Doubtless there are standard results available for choosing the optimal queue length given this ratio and statistics describing the likelihood of repeat requests; also tree or hashing data structures might be beneficial. None of this has been investigated.

### Execution time

LEARNER is not fast! Execution times depend heavily on the complexity of the training image and the settings of various run-time parameters (in particular the number of cost-executions permitted to the GA). Times of between 1 and 4 hours are typical. There is considerable scope for parallelism, however, either within the GA (evaluate a whole population in parallel) or at the probability tree level (grow sub-branches of the tree in parallel).

#### 3.1.2 Tree validation and pruning — HONEST and PRUNE

The probability tree generated by LEARNER is pruned on the basis of an independent validation sample. HONEST is a program which drops this sample into the given tree and accumulates the class proportions for every node — overwriting the originals. Note that some leaves or indeed whole branches of the tree may capture no validation samples at all. PRUNE performs tree pruning and can be run in an automatic mode where it removes only nodes with fewer than a given number of validation samples. Alternatively the user may specify a maximum tree complexity, and the system then uses minimum cost-complexity pruning (see §2.5.6 and [Breiman84]) to remove less efficient branches until the requested tree size is achieved.
Execution times

HONEST takes ~ 3 minutes and PRUNE ~ 1 minute.

3.1.3 Classification on the PC — CLASSY

Classy uses a (pruned) probability tree to classify other (similar) images. It produces a probability image — ie an image in which the pixel intensities represent the a posteriori probability of class-1 given the window of data; clearly this is all that is needed in the two class case. The probability is coded in 255ths, so that 128 represents (approximately) 0.5.

CLASSY cannot process pixels which lie too close to the edge of the image for the full pattern vector to be extracted. These marginal pixels are given probabilities equal to the nearest one clear of the margin. This was a somewhat arbitrary choice — an alternative is to set them equal to the class prior probability.

Execution time

Classy takes 2 to 3 minutes for a 256 x 256 pixel image depending on tree complexity. A significant proportion of the effort is in extracting the pattern vector from the image, and this would probably benefit from hand coding.

3.2 Classification on the Linear Array Processor (LAP)

The original motivation for this research was to develop image classification techniques which would exploit the LAP to classify satellite data rapidly.

3.2.1 Introduction to the LAP

The LAP is a single instruction multiple data (SIMD) parallel processor designed at the National Physical Labs., specifically for image processing applications [Plummer85]. It consists of (typically) 256 bit-slice processors controlled by a common sequencer. The processors are connected together in a 1-D (linear) array, thus each processor may directly communicate with its two neighbours (except at the two ends). The mapping from processors to a 256 square image is a natural one; each processor deals with one column of pixels, the array scanning serially down the rows. Most image processing operations require the collection of data from a neighbourhood around each pixel. Data from horizontal neighbours (ie same row different column) can be accessed by shifting along the array. To permit similar facility in the vertical direction, each processor carries 256 bits of random access memory (RAM), sufficient to store data from several rows of the image. It is thus possible to hold a strip
of image data in memory to satisfy vertical access, provided these access are not too long range.

The linear mapping of processors onto images is particularly appropriate where the image is being captured by a line scan camera and is to be processed in real time. In such a situation a 2-D array of processors (eg as in the DAP [Flanders77] or CLIP-4 [Duff80]) would be of little use as the array would have to idle whilst waiting for a full frame, introducing a possibly unacceptable lag time. Interestingly, the latest CLIP machine — the CLIP-7A [Fountain88], also adopts this linear array of processors.

In its low level implementation, the LAP handles rows and columns in a completely asymmetrical fashion, and it would be tedious in the extreme if the applications programmer had to be aware of this. In fact the LAP is always programmed in a high level language — 'Picture Processing Language', or PPL [Plummer86]. PPL permits the programmer to specify output pixel values in a functional style, using symmetrical 2-D coordinates; the compiler takes care of the mapping to the asymmetrical LAP. Figure 3.3 shows an example PPL program which applies a (approximate) Gaussian smoothing function and then thresholds at the mid grey (127) level. Code between the apply and end is ‘applied’ at every pixel position in the image.

### 3.2.2 Generation of PPL code for the LAP

Since PPL directly supports image convolutions and conditional expressions. It is convenient to represent a probability tree as a nested conditional expression. The program TOPPL takes, a probability tree as input and generates the equivalent PPL program. Sample output from a small tree is shown in fig. 3.4. The numbers which form the terminal expressions are the probability estimates of class-1 in terminal node $t$ — ie $P(Y=1|X \in \Omega_t)$, expressed in 255ths. The output from the LAP is therefore a probability image, white representing certain class-1, black meaning certain class-2, and the shades of grey in between indicating varying shades of uncertainty.
3.2.3 LAP execution times

In its present form the LAP depends on a LSI-11 computer for its input and output; image rows are passed to the LAP and collected from it one row at a time. There is no direct memory access. Because of this bottleneck, all PPL programs seem to take \( \sim 1 \) second to process a \( 256 \times 256 \) image, regardless of the complexity of the program. Note that as no program loops are permitted, this complexity is limited by the size of the program store. There are in fact other implementation problems which are specific to the LAP (eg only a single image band may be used) but as these are not fundamental to SIMD architectures in general, they will not be discussed further. A small tree has been run successfully on the LAP though in fact the results presented in chapter 5 were generated on the PC using CLASSY.

3.3 Multiple class problems

The techniques developed in chapter 2 are perfectly well suited to multiclass problems. However, a decision was taken to impose a two-class restriction on the implementation, partly for reasons of efficiency and ease of coding, but also because the LAP is able to input and output only one image at a time. Multi-class problems must therefore be broken down — eg, if \( \Phi = \{ \text{water}, \text{urban}, \text{other} \} \), then this may be treated as two separate problems of \( \Phi = \{ \text{water}, \text{(urban} \cup \text{other}) \} \) and \( \Phi = \{ \text{urban}, \text{(water} \cup \text{other}) \} \), requiring the growth of two probability trees. The PP for 'other' \( (Q_{\text{other}}) \) may be inferred from \( 1 - Q_{\text{water}} - Q_{\text{urban}} \) although this quantity might turn out to be negative. Alternatively, the 'other' class may be explicitly estimated by growing a third tree with \( \Phi = \{ \text{other}, \text{(water} \cup \text{urban}) \} \). Again, the generated probability vector is unlikely to sum to exactly one. One advantage of this latter method is that classes may be assigned non-exclusively — for example it may be useful to classify road pixels in an urban area as both road and urban.

3.4 Summary

The techniques developed in the previous chapter have been implemented as a suite of five Pascal programs — LEARNER, HONEST, PRUNE, CLASSY and TOPPL. The implementation permits classification to be performed on a serial machine, using CLASSY, or on the Linear Array Processor (LAP) after translation of the probability tree by TOPPL into PPL — the LAP's standard language. Multiple trees must be grown for multi-class problems.
apply
  if convolve [ 0, 0, -2, -6, 4, 3, 1,
                 0, -4, 1, 1, 0, -4, 4,
                 3, -2, 3, -4, -6, 0, -2,
                 -6, 7, -2, 4, -7, -2, 0,
                 -6, -2, 2, -1, -3, 1, 1,
                 -6, 5, -2, 1, 0, 1, 5,
                 -4, 5, -5, 5, -4, 0, 0 ] < -16 then
    if convolve [ 0, 1, 5, -1, 0, 0, 7,
                   4, 7, 7, -2, 1, 0, 6,
                   0, 7, -1, -6, 0, 5, -1,
                   -3, -5, 7, -5, -7, -7, -4,
                   -3, -4, 3, 5, -7, -5, 5,
                   5, 2, 1, 6, 5, 0, 4,
                   4, 3, -2, -1, -5, 2, 0 ] < -28 then

  254
  else
  18
else
  0
end

Figure 3.4: A small probability tree as PPL code
Chapter 4

Exploiting context

There are many examples of how appreciating the context of a situation can enable us to make a better decision. Joseph was able to recognize his ten famine-weary brothers because they were together, in context, while he himself remained unrecognized because he was alone and distinctly out of context — "dressed in fine linen and a golden chain" (Genesis, 41-42). Hand-written letters that are singly indecipherable may be read easily when seen in the context of a word or whole sentence.

Similarly, an interpreter of satellite imagery may be hard pressed to distinguish rivers from roads or canals if forced to view the image through a small cut-out window. When allowed to see more of the image however, characteristics such as sinuosity, connections into lakes or other river or road networks will often be sufficient to enable identification. At a simpler level, where a small region of the image is degraded by some form of noise — cloud cover perhaps, and the region surrounding it is easily identifiable, eg sea, then the classification of the obscured region may be guessed with reasonable confidence.\(^1\)

That a better classification can be made from more (relevant) information is obvious. Unfortunately, many pattern recognition techniques become infeasible when presented with too many variables (see the discussion in chapter 2) and more samples are required for their training. Simply ‘throwing more data’ at a problem has little theoretical interest in any case. The problem becomes interesting only when the overall task is broken down into sub-tasks, each based on some small subset of the data, and the results combined subsequently in the light of known contextual constraints, either to improve the accuracy of each sub-task or to achieve a result that is more globally consistent.

Focusing the discussion on the particular subject of this thesis, we are interested in ways that estimates of per-pixel *a posteriori* probability based on only single pixel data (or

\(^1\)However, clouds at sea tend to gather over islands where the damp air is forced to rise and so cool and condense! There is a general lesson here: if the ‘noise’ is known to be correlated with the categories we wish to identify, contextual information must be used with care.
data in a small window) can be combined to achieve estimates that incorporate contextual constraints between neighbouring pixels. For a general review of approaches to context in pattern recognition see [Toussaint78].

The layout of this chapter is as follows. In section 4.1 the statistical framework is established within which context is to be considered, and a review of some important work that fits into this framework is presented. Section 4.2 introduces the technique that is of particular interest to this thesis — probabilistic relaxation labeling (PRL). In §4.3 are presented the main theoretical results of this thesis, concerning the upper performance limits of any PRL scheme. The approach is to examine the performance of a scheme that is defined to be optimal. However, it is argued that attempts to find computational implementations of the optimal scheme are unlikely to succeed and may be ill founded. Finally, in §4.4 a practical method of implementing the optimal scheme through direct estimation rather than by computation is described. The method has been called ‘trained probabilistic relaxation’ (TPR) and is the context exploiting method used by the Lapwing system.

4.1 Introduction to the Bayesian approach to context

For the purpose of this overview, we need only a general formulation of the contextual pattern recognition problem. Thus, assume that there are a number of objects (e.g., pixels) \( \mathcal{I} = \{1 \ldots N_I\} \). Associated with each object \( i \) are random variables for its true label (e.g., class) \( Y_i \) and some data (e.g., pixel intensities) \( X_i \). Particular realizations are written \( y_i \) (\( y_i \in \Phi \)) and \( x_i \) respectively. A random variable that joins the labels of all objects in \( \mathcal{I} \) is \( \mathbf{Y} = (Y_1 \ldots Y_{N_I}) \); similarly, \( \mathbf{X} = (X_1 \ldots X_{N_I}) \). Realizations of these joint random variables are written \( \mathbf{y} \) and \( \mathbf{x} \) and may be thought of as representing the combined true classification and combined pixel intensities respectively, for a whole image.

Statistically founded methods of exploiting context in pattern recognition can be usefully divided into two distinct approaches, termed ‘message’ and ‘object-centred’ schemes [Devijver88b].\(^2\) A useful summary of the various statistical approaches to context in images is provided in [Ripley86].

4.1.1 The ‘message-centred’ scheme

The message-centred approach can stated as:

\(^2\)There is a unfortunate clash in the usage here of the word ‘object’ and its use in §2.1. Here, ‘object’ refers to a single atomic item — in our case a pixel. A ‘message’ is a collection of interacting objects — in our case a whole image or perhaps a part of one. It appears that the nomenclature was conceived with speech or text recognition in mind.
Find the best simultaneous classification for all objects in the light of all available data and contextual constraints.

formally:

\[
\hat{y}^* : P(Y = \hat{y}^* | x) = \max_y P(Y = y | x)
\]

where the max operator runs over all combinations of label values. Even a small \((64 \times 64)\) image contains 4096 pixel objects, so a literal implementation of this scheme is clearly out of the question \((2^{4096}\) label combinations for a binary classification).

In spite of the computational difficulties, considerable success has been achieved with this scheme using heuristic optimization techniques, most notably simulated annealing ([Kirkpatrick83]) by Geman and Geman [Geman84] [Geman87b]. They assume a Markov Random Field (M-RF) model\(^3\) for the label generating process, representing the joint probability \(p(y, x)\) with a Gibbs distribution (see [Devijver88c] or [Besag86]). Simulated annealing involves considering possible changes to an initially random configuration of the labels, on the basis of their effect on the above joint probability. Improvements are always accepted, degradations might be. Initially, almost any change is accepted, but as the annealing process proceeds, the 'temperature' of the system is slowly reduced so that highly detrimental changes are less and less likely to be adopted. Eventually, when the temperature reaches zero, the system 'sets' at some local maximum of the probability distribution. The slower the temperature is reduced the more likely the system is to set at the global maximum. (cf MRF texture generation described in §5.3.1). This technique has been applied to texture image segmentation [Geman87a], [Geman87c]. Related work may be found in [Derin87].

A similar approach, but one that is computationally less demanding, is proposed by Besag [Besag86], which he calls iterated conditional modes (ICM). Again, a Markov model is assumed. The MAP classification of each pixel, based on its own data, is used as the starting point. Each pixel \(i\) is then visited in a random order and its label \(y_i\) changed (if necessary) to that which maximizes the joint distribution \(p(y, x)\), where \(y\) is all current label realizations, including \(y_i\). Each pixel will be visited several times before stability is reached. This method is computationally cheap, but the results are sensitive to the initial classification as the 'search' can only ascend to the local maximum of the distribution peak in which the initial classification has placed it. Apart from the use of a non-random initial classification, this method is equivalent to a simulated annealing approach, with a fixed temperature of zero [Besag86].

Recently, Greig et al [Grieg89] have discovered a method of performing exact message-centred MAP classification on the same type of image model assumed by Besag and Geman

---

\(^3\)for introductions to MRFs see [Woods72], [Besag74] or [Isham81]
& Geman, but applicable only to binary data (ie, two classes and two-state data). The method borrows results from the theory of 'capacitated networks' and is computationally costly. Nevertheless, this exact solution is valuable for comparing other methods. They have shown that simulated annealing in fact generates poor results, at least for any practicable annealing schedule.

### 4.1.2 The 'object-centred' scheme

This can be stated as follows:

*Given all the data in the image, find the most likely classification for each pixel.*

Formally:

\[
\text{for each } i \in I \text{ find } y_i^* : P(Y_i = y_i^* | \mathbf{x}) = \max_{\alpha} P(Y_i = \alpha | \mathbf{x}).
\]

Thus, each pixel is classified independently, taking into account the data of (ideally) all other pixels in the image. The range of the optimization is now trivial, but the computation of the probability vector \(D(Y_i | \mathbf{x})\) (from whence \(P(Y_i = \alpha | \mathbf{x})\)) is extremely problematic for 2-D image data under the MRF model. However, if the model is simplified to a Markov mesh [Devijver88a], then this scheme can be implemented efficiently via an adaptation of Baum's forward/backward chaining algorithm [Baum72] (see also [Devijver85]). The implementation is possible because the Markov mesh model allows for interactions in only two perpendicular directions, rather than the four of the full (first order) MRF. This removes the possibility of cycles in the dependency graph, and thus converts it from the intractable non-causal network to a tractable causal one. See [Besag86] for further discussion.

A quite different approach to object-centred classification uses Bayesian belief networks — directed graphs that explicitly represent the statistical relationships between variables ([Pearl86], [Pearl87], [Pearl88] also [Dawids88]). Each node of the graph acts as an autonomous processor that receives messages from its connecting arcs, updating its own probabilistic assessments and in turn re-transmitting these. This technique has been applied to linear feature classification in aerial imagery by [Peacegood89]. A major restriction of the belief network approach is that the network must be in the form of an acyclic graph — ie a causal network. The MRF image model is non-causal and so the updating scheme cannot be applied. However, as has been mentioned earlier, the simpler Markov mesh model is causal. In fact it can be shown [Devijver88d] that Baum's algorithm is a specific case of Pearl's parallel updating scheme.

Probably the most studied method of incorporating contextual information into image pixel classifiers is probabilistic relaxation labelling (PRL). This involves the iterative, parallel
4.2 Introduction to probabilistic relaxation labeling (PRL)

Probabilistic relaxation labelling grew out of discrete relaxation originally proposed by Waltz [Waltz75] and developed by Rosenfeld et al ([Rosenfeld76], [Zucker77]) and Haralick and Shapiro [Haralick78a]. In its original form only categoric classifications were involved. Rosenfeld et al assigned fuzzy quantities to the objects and called them probabilities. However the method of calculating the quantities was intuitive [Kittler85a] rather than statistically founded.

Various studies (eg [Eklundh80], [Swain81], [Kittler84], [Harris85] and [Settle87c]) have found some form of PRL to be useful in multispectral image classification. A review of early work may be found in [Davies81]. For some related work (mostly pertaining to image segmentation rather than classification) see [Derin78], [Hanson82], [Cooper83], [Yu83], [Cohen87] and [Zenzo87a].

This thesis follows the view expressed by Kittler and Foglein [Kittler86a], that PRL should be statistically founded, fitting into the object-centred scheme previously described. PRL is then seen as a means of iteratively calculating new PPs from the existing PPs in a small neighbourhood. The supposition is that the quantities so calculated are PPs conditioned on a window of data which increases in size with each iteration. See [Kittler85a] for a review.

The process is begun with the single pixel PPs which we will call \( p_i \) (ie, \( p_i = D(Y|X_i=x_i) \) — PPs conditioned on the data from a single pixel only). An updating function then aims to generate improved PPs (ie, equivalent to ones conditioned on a larger window of data) by combining the PPs from the central pixel and (say) its 4-connected neighbours. This new PP can be called \( q^1 \), ie:

\[
q_C^1 = f(p_C, p_N, p_S, p_E, p_W)
\]

where the subscript C,N,S,E,W stand for Center, North, South etc. The updating function (in all previous work, the same function) is then re-applied to generate \( q^2 \) etc, ie, at the kth iteration:

\[
q_C^{k+1} = f(q_C^k, q_N^k, q_S^k, q_E^k, q_W^k). k > 0.
\] (4.1)
It is then tempting to suggest that the conditioning window will enlarge in a straightforward way, so that (for the 4-connected neighbourhood, dropping the pixel index):

\[
p = D(Y_C|\mathbf{x})
\]

\[
q^1 = D \begin{pmatrix} x \\
\hat{Y}_C | x & x & x \\
x & x & x & x \\
x & x & & x \\
x & & & & x
\end{pmatrix}
\]

\[
q^2 = D \begin{pmatrix} x \\
\hat{Y}_C | x & x & x & x & x \\
x & x & x & x & \\
x & x & x & x & x \\
x & & & & & x
\end{pmatrix}
\]

etc

In fact this suggestion will subsequently be refuted. We will call the above sequence of windows a regularly-increasing window sequence.

Several versions of the updating function \( f \) in 4.1 have been proposed by various authors. That of Rosenfeld already mentioned has received much attention in spite of its lack of a statistical basis. Further developments were made by Peleg (to be discussed later), and Kirby [Kirby80] introduced an updating function based on a product rule (previous functions had essentially involved an arithmetic averaging of the local probabilities). In [Kittler86a], several statistically founded updating functions are derived for a range of model assumptions, though none admit the full 2-D hair-brush model, to be defined in §4.3.1. The latter authors also explicitly caution that these updating functions are valid only for the first iteration, after which they become strictly heuristic. This caution in fact applies to all other previously proposed updating functions, though it is often overlooked.

The updating functions depend on parameters that describe the compatibility of label juxtapositions, and these are typically termed 'compatibility coefficients' (e.g. [Rosenfeld76]). These coefficients may either be assigned heuristically, or be estimated from sample images. It is usually assumed that the update functions depend only on statistics of the true scene — that the pixel data model is not involved; this assumption will be questioned in §4.3.5. It is shown in [Mohn87] that the presence of spatial autocorrelation in the measured data significantly impairs the performance of contextual schemes which ignore the phenomenon. Autocorrelation might be introduced by the sensor [Craig80], or be induced due to the presence of 'sub-classes' in the specification of the problem [Poole88e].

A particular PRL updating function — that of [Haralick83a], may be seen in §5.7.2.
4.3 A study of some theoretical limits on PRL

The purpose of the work presented in this section is to develop a better understanding of PRL within an exact statistical and information theoretic framework. Theorems (or forceful conjectures) will be developed which place upper limits on the performance of any PRL scheme (within the definition alluded to in the previous section), regardless of the choice of the updating function, and that this limit falls short of the 'regularly-increasing window' scheme; further, that to achieve optimal performance, the updating functions must be tailored to both the relaxation iteration and the original distribution of the data.4

The broad approach is to consider the potential of a PRL scheme, which is defined to be optimal under any model, and then to analyze its performance on an information theoretic basis under a particular commonly used image model. Some of the presented proofs apply to only the 1-D (tooth-comb) model — however, convincing arguments are given to justify their extension to the 2-D case.

It should also be stated from the outset that a closed-form expression for the optimal updating function(s) is not delivered, and indeed it is argued that the search for such a function is ill-founded.

4.3.1 Formulation

The image models used in the study are specified in this section. As well as stating the statistical dependencies explicitly, it is useful also to show these as "I-graphs"; some guidance on the comprehension of I-graphs is given in appendix B.

The 'Tooth-comb' model

This is a simple 1-D model. $Y_i$ are the class random variables, and $X_i$ are the associated pixel data. See fig. 4.1. From the diagram we can see that:

• $X_i [\{X_j, Y_j\} | Y_i; \forall i, j : i \neq j$ (class-conditional independence (CCI));

• $Y_i [Y_j | (Y_{i-1}, Y_{i+1}) \forall i, j : i \neq j$ (the Markov property).

Figure 4.1: The I-graph for the tooth-comb model.

\[ \text{\footnotesize 4The main results in this section were reported in [Poole89a] and [Poole90]} \]

\[ \text{\footnotesize 5A} \frac{\text{\footnotesize 5A}}{\text{\footnotesize 5A}} [\text{\footnotesize 5A}] [\text{\footnotesize 5A}] [\text{\footnotesize 5A}] \text{\footnotesize 5 reads as \textquoteleft A is independent of B given C\textquoteleft, or \textquoteleft Once we know C, B tells us nothing more about A (or A about B)}\textquoteleft. Formally, } D(A | B = b, C = c) = D(A | C = c) \text{ for all realizations } b \text{ and } c. \]
and in fact these two statements completely specify the qualitative aspects of the model.

An instance of the model requires the following to be quantified:

- the set of classes — \( \Phi = \{1..m\} \);
- the class priors — \( D(Y_i) (\pi_a = P(Y_i=\alpha), \ \forall i) \)
- the transition probabilities — \( M(Y_{i+1}|Y_i) (P(\alpha|\beta) = P(Y_{i+1}=\alpha|Y_i=\beta) \ \forall i) \);
- the class-conditional density distributions — \( d(X|\alpha) = d(X_i|Y_i=\alpha) \ \forall i. \)

Notice that the pixel index may be dropped in the abbreviated forms since we assume a stationary model. By stationary, we mean that the image statistics are not affected by a spatial shift in the image. Thus the probability distributions \( D(Y_i), D(Y_{i+1}|Y_i), d(X_i|Y_i=\alpha) \) are the same for all \( i \); however the a posteriori probability vector \( D(Y_i|X_i = x_i) \) clearly does depend on \( i \) since it depends on the actual image data. Also note that \( P(Y_{i+1}=\alpha|Y_i=\beta) \) will not equal \( P(Y_i=\alpha|Y_{i+1}=\beta) \), in general, except in the two-class case (see [Devijver88c]), and that even in the two-class case we in general do not have \( P(Y_{i+1}=\alpha|Y_i=\beta) = P(Y_{i+1}=\beta|Y_i=\alpha) \) unless the priors are equal; however one can be calculated from the other if the class priors are known.

Realizations of \( Y_i \) and \( X_i \) are written as \( y_i \) and \( x_i \) respectively. Often, the event \( Y_i = y_i \) is abbreviated to simply \( y_i \).

The a posteriori probability will occur so frequently that a further abbreviation will be adopted:

\[ p_{i,\alpha} = P(Y_i=\alpha|x_i) \]

and the distribution vector —

\[ p_i = D(Y_i|x_i) = (p_{i,1}, p_{i,2}, \ldots, p_{i,m}). \]

We shall wish to treat these a posteriori probabilitys as though they are given data items, and to consider probabilistic quantities conditioned on them. Thus, we consider \( p_{i,\alpha} \) and \( p_i \) to be realizations of the random variables \( P_{i,\alpha} \) and \( P_i \) respectively. Note that \( P_8 \) and \( P_9 \) (say) are different random variables, even though they will possess identical statistics due to the assumed stationarity of the model. We need to treat them as separate random variables so as to be able to create joint random variables, encompassing a number of neighbouring pixels. The following abbreviations are used for such joint random variables, with their realizations in lower case:6

\[ Y_{a,b} = (Y_a \ldots Y_b) \]

---

6There is potential for confusion in our notation, as there are two different situations in which it is used. When considering pixels in isolation, divorced from their spatial relationships, there is just one random
4.3. A STUDY OF SOME THEORETICAL LIMITS ON PRL

\[ y_{a,b} = (y_a \ldots y_b) \]
\[ X_{a,b} = (X_a \ldots X_b) \]
\[ z_{a,b} = (z_a \ldots z_b) \]
\[ P_{a,b} = (P_a \ldots P_b) \]
\[ p_{a,b} = (p_a \ldots p_b) \]

A glossary of notation is given in appendix A.

The 2-D ‘hair-brush’ model

This is the two dimensional version of the tooth-comb model just described, and may be considered a plausible (if simplistic) model for some types of imagery. The I-graph is shown in fig. 4.2.

![Figure 4.2: I-graph for the Hair-brush model.](image)

The qualitative relationship between the classes and the data is just as for the tooth-comb — ie the data is class-conditionally independent. The classes themselves however form a (non causal) Markov random field. The non-causality is caused by the presence of directed cycles in any attempted representation of the field as a directed graph. This leads to serious problems when attempting to compute distributions on a given subset of nodes in an assumed infinite lattice, as the factorization, which is possible in the (causal) Markov variable (for class), \( Y \), with realizations \( y \). A single image may be considered to provide a sample over which statistics may be estimated, and \( i \) indexes into this sample — \( \{ y_i, i = 1 \ldots N \} \). When spatial relationships are involved, however, the random variable is \( Y_{1,N} = (Y_1 \ldots Y_N) \), ie \( i \) indexes the components of the \( N \) dimensional) random variable. A realization of \( Y_{1,N} \) is \( y_{1,N} \) and is a whole image. It is now understood that statistics are calculated over many whole images, but the notation does not provide an index into this sample set. For this reason, the notation may on occasions lack strictness, but should hopefully be clear.
CHAPTER 4. EXPLOITING CONTEXT

chains, cannot be performed on 2-D fields. (See [Besag74], [Besag86] and [Devijver88c] for a detailed discussion.) It is for this reason that the tooth-comb model is used predominantly in this section.

4.3.2 Do per-pixel PPs preserve all relevant information?

The first stage of any PRL scheme is to generate the \textit{a posteriori} probability vector for each pixel \( i \) given its data, ie \( p_i = D(Y_i|x_i) \). Once this has been done, the data \( x_i \), will not be used again. We would like to know, therefore, whether this transformation from data to PPs discards any information which might be relevant to a subsequent object centred-classification. Intuitively we are asking “Could I make as good an estimate for each pixel if allowed to see only the PPs for the whole image, as I could if I were allowed to see the original data for the whole image?” In fact we can assert the following:

\textbf{Theorem 1} When the pixel data is class-conditionally independent, all information relevant to classification is retained by the per-pixel PPs. ie

\[
X_i \bigcup_{i \neq j} \{X_j, Y_j\} \big| Y_i \Rightarrow D(Y_i|x_{a,b}) = D(Y_i|p_{a,b}) \quad \forall a \leq b
\]

\textbf{Proof}

We must show that \( p(Y_i=a|x_{a,b}) \) is a function of \( (p_a \ldots p_b) \), thence, invoking lemma 9 (see appendix C) establishes the theorem.

\[
P(Y_i=a|x_{a,b}) \propto p(Y_i=a, x_{a,b}) = \sum_{\forall x_{a,b}} \sum_{y_{a,b}=a} P(Y_{a,b}=y_{a,b}, x_{a,b})
\]

\[
= \sum_{\forall x_{a,b}} \sum_{y_{a,b}=a} P(Y_{a,b}=y_{a,b}) \prod_{j=a}^{b} p(x_j|y_j) \quad \text{(by the CCI hypothesis)}
\]

\[
= \sum_{\forall x_{a,b}} \sum_{y_{a,b}=a} P(Y_{a,b}=y_{a,b}) \prod_{j=a}^{b} \frac{p_{j|y_j}}{\pi_{y_j}} \quad \text{(constant of prop. is } \prod_{j=a}^{b} p(x_j)}
\]

As the conditioning data now appears only in the context of the per-pixel PPs, we have shown that \( p(Y_i=a|x_{a,b}) \) is indeed a function of \( (p_a \ldots p_b) \). □

Note that no mention is made of the relationship between the classes \( Ys \): only class-conditional independence of the data is assumed, and so the theorem is valid for both the 1-D tooth-comb and the 2-D hair-brush model, and indeed for non MRF models.

\footnote{Here, and throughout this chapter, factors that involve probabilities of the data only (ie \( p(x_i) \)), will frequently be suppressed and ‘hidden’ in the constant of proportionality; since these constants do not involve the labels, they may always be recovered (in principle at least) due to the normalizing constraint on the complete probability vector — see eg equation 4.8.}
4.3. A STUDY OF SOME THEORETICAL LIMITS ON PRL

Discussion

This is a simple, though perhaps not an intuitively obvious, result. Bearing in mind that \( x \) may be multivariate (eg multispectral data) and/or the class distributions may be multimodal, the transformation \( x \rightarrow p \) will in general be irreversible and thus information destroying. However, the theorem tells us that not one piece of information pertaining to \( Y_i \) has been lost. Thus, having replaced all the \( x \)s with \( p \)s we still have a chance of finding \( D(Y_i|x_{i-w} \ldots x_{i+w}) \) for any \( w \). Note that even if the contextual PP for only a single class \( \alpha \) is needed, the entire PP vectors are still required — not simply the components for class \( \alpha \) — ie, \( p_1 \ldots p_n \) is needed, not just \( p_{1,\alpha} \ldots p_{n,\alpha} \). This shows that performance will in general, be lost if a multiclass (\( m > 2 \)) problem is reduced to separate two-class problems, eg \{forest, other\}, \{water, other\} etc. This is because the CCI assumptions that were assumed valid for the multiclass problem will (in general) break down when classes are amalgamated. 8 This is enlarged upon in [Poole88e].

Single-stage, \( w \)-reaching context functions — \( F^w \)

We now derive a function that computes from the per-pixel PPs (\( p_i \)), new PPs conditioned on a window of data \( x_{i-w,i+w} = (x_{i-w} \ldots x_{i+w}) \), for the tooth-comb model.

\[
F^w(p_{i-w, i+w}) = D(Y_i|p_{i-w, i+w}) = D(Y_i|x_{i-w, i+w}) \quad (4.3)
\]

\[
F^w_\alpha(p_{i-w, i+w}) = P(Y_i=\alpha|p_{i-w, i+w}) = P(Y_i=\alpha|x_{i-w, i+w}). \quad (4.4)
\]

Theorem 1 assures us that the final equalities will hold, ie that an assessment based on the per-pixel PPs will be identical to one based on the original data. This function represents the ‘absolute limit of excellence’, against which we can compare any object-centred context exploiting scheme.

Derivation

\[
P(Y_i=\alpha|x_{i-w, i+w}) \propto p(Y_i=\alpha, x_{i-w, i+w}) = \sum_{y_{i-w, i+w}} p(y_{i-w, i+w}, x_{i-w, i+w}) \quad (4.5)
\]

where the sum is over all \( \text{combinations} \) of label values as long as the \( i \)th pixel has the value \( \alpha \). The joint distribution can be factorized beginning at any point — here we choose the central (\( i \))th pixel:

---

8 Intuition: It may be that, say, rivers are spatially correlated with a particular type of grassland, but there is no correlation with grassland in general; the original data would (hopefully) permit the discrimination of the grass type, which the PPs from a simplified model (\( \Phi = \{\text{river,grass,other}\} \)) would have lost.
\[ P(y_{i-w,i+w}, x_{i-w,i+w}) = \]
\[ \pi_{yi} p(x_i|y_i) \prod_{j=i-w}^{i-1} P(y_j|y_{j+1}) p(x_j|y_j) \prod_{j=i+1}^{i+w} P(y_j|y_{j-1}) p(x_j|y_j). \]

Noting that:
\[ p(x_j|y_j) = \frac{P(y_j|x_j)p(x_j)}{P(y_j)} = \frac{p_{yj} p(x_j)}{\pi_{yj}} \]
we have:
\[ p(y_{i-w,i+w}, x_{i-w,i+w}) = \]
\[ \pi_{yi} \prod_{j=i-w}^{i-1} p(x_j) \prod_{j=i-w}^{i-1} \frac{P(y_j|y_{j+1}) p_{yj} p(x_j)}{\pi_{yj}} \prod_{j=i+1}^{i+w} \frac{P(y_j|y_{j-1}) p_{yj} p(x_j)}{\pi_{yj}}. \] (4.6)

Substituting 4.6 into 4.5 and losing the \( \prod p(x_j) \) factor in the proportionality (it does not depend on \( y_{i-w,i+w} \)) gives:
\[ F^w(\alpha, x_{i-w,i+w}) = P(Y_i = \alpha|x_{i-w,i+w}) \propto \]
\[ \pi_{y_{i+w}} \sum_{\forall y_{i-w,i+w}, y_i = \alpha} \prod_{j=i-w}^{i-1} \frac{P(y_j|y_{j+1}) p_{yj} p(x_j)}{\pi_{yj}} \prod_{j=i+1}^{i+w} \frac{P(y_j|y_{j-1}) p_{yj} p(x_j)}{\pi_{yj}}. \] (4.7)

Denote the RHS of 4.7 by a function of \( \alpha \) and \( x_{i-w,i+w} \), i.e:
\[ F^w(\alpha, x_{i-w,i+w}) \propto f(\alpha, x_{i-w,i+w}) \] (4.8)
and then eliminate the proportionality by normalizing the \textit{a posteriori} probability vector, yielding:
\[ F^w(\alpha, x_{i-w,i+w}) = \frac{f(\alpha, x_{i-w,i+w})}{\sum_{\beta} f(\beta, x_{i-w,i+w})}. \] (4.9)

This derivation depends on the easy factorization (4.6) of a Markov chain which cannot be applied to a general Markov field so that the analogue of \( F^w \) is not tractable for 2-D images — which is exasperating. It is for this reason that no updating function has yet been proposed which is valid even for the first iteration and admits the full hair-brush model without further simplifying assumptions. Nevertheless, \( F^w \) is perfectly well defined in the 2-D case, and in much of the subsequent discussions no distinction will be made between the 1 and 2-D versions — we just don’t have an expression for the latter case.

Note also that \( F^w \) does not require knowledge of the class-conditional data distributions, \( d(X|Y=\alpha) \).
If taken literally the calculation of $F^w$ from 4.7 has exponential complexity w.r.t the window width $w$. In fact there is a well know re-formulation, known as ‘Baum’s forward/backward chaining algorithm’ [Baum72] (or see [Devijver88c]), which reduces this to linear order. However, as we wish to perform algebraic rather than arithmetic manipulations of $F^w$, it is simpler for our purpose to work with the form as given.

### 4.3.3 The incrementally optimal updating (IOU) scheme — $g^1 \ldots g^k$

Probabilistic relaxation aims to approach an estimate of $D(Y_i|x_{i-w,i+w})$ by a series of iterations which at each stage consider only current PPs of the $i$th and neighbouring pixels. The existence of $F^w$, (and Baum’s efficient algorithm) makes PRL of little practical interest for 1-D chains (tooth-combs) since the required quantity can easily be calculated directly. This section will nevertheless show that PRL cannot match $F^w$ even in the 1-D case, and the relevance of this result to the full 2-D case will be discussed later in section 4.3.4.

In the 1-D case we will consider a pixel to have only two neighbours, one to the left and one to the right. This is shown in a fig. 4.3.\(^9\) At the $k$th iteration, a function, $g^k$, of three-

---

\(^9\)Note that fig. 4.3 is not an I-graph—rather it represents functional dependence.
CHAPTER 4. EXPLOITING CONTEXT

Consider the case that \( g^k \) is to generate a completely honest probability, and further that it is to be fully refined on the available information. It thus follows (see §2.3) that we should define:

\[
g^k(q_{i-1}^k, q_i^k, q_{i+1}^k) = D(Y_i|Q_{i-1}^k=q_{i-1}^k, Q_i^k=q_i^k, Q_{i+1}^k=q_{i+1}^k)
\]

\[
\tilde{g}^k(q_{i-1}^k, q_i^k, q_{i+1}^k) = P(Y_i = a|Q_{i-1}^k=q_{i-1}^k, Q_i^k=q_i^k, Q_{i+1}^k=q_{i+1}^k)
\]

(4.10)

as is shown in the diagram.

The 'optimal' formulation of 4.10 is in itself rather vacuous, as it gives little hint as to how the function should be computed. This formulation is precisely that adopted by [Peleg80a]. However, he is able to derive an updating function from it, only after imposing strong assumptions. Firstly, in order to compute the new distribution based on all neighbours he is forced to leave the Bayesian domain and adopt a heuristic method (this was observed by Haralick [Haralick83a]). Secondly, even to obtain distributions based on just one other neighbour he assumes class-conditional independence (CCI) of the previous stage probabilities (\( q_i^k \)'s). That the original data satisfies CCI is defined by the hair-brush model, and from this it follows that the original computed PPs will also satisfy CCI. However, the assumption will not hold after the first update by PRL since each new PP will have incorporated evidence from overlapping windows of data. Thus, even if the derivation were valid for the first iteration, it would not be so subsequently. Haralick also makes the second observation (concerning Peleg's CCI assumptions), but considers the approximation to be a reasonable one and uses it himself to obtain an alternative to Peleg's updating function (this is given in §5.7.2). In spite of these objections, the basic approach — of considering probabilities conditioned on the current of PPs — is sound, and it is unfortunate that the method received such harsh criticism, which perhaps discouraged others from developing the ideas.

To say that equation 4.10 defines an optimal updating function is something of a truism; note for example, that it is true for any assumed image model, ie it holds regardless of whether the data is class-conditionally independent, or the labels exhibit the Markov property. Of course the actual form of \( g^k \) will depend on the model. Even if there is texture present, it is still optimal amongst PRL schemes, though the optimum will be a poor one. Note that one may always take equation 4.10 literally and, given sufficient training data, directly estimate the given conditional probability. This will be taken up in §4.4.

Clearly, the sequence of updating functions cascade into one-another, so that for example,

\[^{10}\text{This follows since each PP is a function of the data at a single pixel, and it is a property of independence that } A \big| B \big| C \Rightarrow f(A) \big| f(B) \big| f(C).\]
4.3. A STUDY OF SOME THEORETICAL LIMITS ON PRL

\[ q_i^2 = g^2 \left( g^1 \left( \begin{array}{c} p_{i-2} \\ p_{i-1} \\ p_i \\ p_{i+1} \end{array} \right), g^1 \left( \begin{array}{c} p_{i-1} \\ p_i \\ p_{i+1} \end{array} \right), g^1 \left( \begin{array}{c} p_i \\ p_{i+1} \end{array} \right) \right) \]

however we will abbreviate this by defining: \(^{11}\)

\[ G^2(p_{i-2},i+2) = q_i^2 = g^2 g^1(p_{i-2},i+2) \]

The first updating function — \( g^1 \)

The first updating function is a special case since it is based on information which is, according to our model, class-conditionally independent — i.e., \( P_i \mid P_j \mid Y_i \), \( \forall i,j \). In fact it is clear that \( g^1 \) is identical to \( F^1 \) (see section 4.3.2).

\[ G^1(p_{i-1},i+1) = D(Y_i \mid P_{i-1}, P_i, P_{i+1}) = F^1(P_{i-1}, P_i, P_{i+1}) = D(Y_i \mid x_{i-1}, x_i, x_{i+1}) \]

Definition of the IOU scheme

For succinctness, we can formally define the incrementally optimal updating scheme as follows:

\[
\begin{align*}
q^k &= g^k(q_{i-1}^{k-1}, q_i^{k-1}, q_{i+1}^{k-1}), \quad k > 0 \\
q_i^0 &= P_i \\
g^k(d,e,f) &= D(Y_i \mid Q_{i-1}^{k-1} = d, Q_i^{k-1} = e, Q_{i+1}^{k-1} = f), \quad k > 0 \\
G^k &= g^k \ldots g^1, \quad k > 0.
\end{align*}
\]

When the data is class-conditionally independent, we have by theorem 1:

\[ g^1(d,e,f) = F^1(d,e,f) \quad (4.11) \]

however, we have given a closed expression for \( g^1 \) only for the tooth-comb model.

4.3.4 Does the IOU scheme match a one-stage function?

In this section we will pursue a key result of this chapter, namely that the incrementally optimal updating scheme does not achieve an estimate whose conditioning window increases by exactly one window width (of raw data) at each iteration: i.e., that the \( k \)th iteration of IOU is not equivalent to \( F^k \) which sees the original per-pixel PPs.

Here, we establish the stronger result, that the IOU scheme will have fallen behind by the second iteration. Since we require only a counter-example, it is sufficient to consider only

\(^{11}\)Strictly speaking we should consider \( g^k \) to accept a variable length string as parameter, and to return a string of results whose length is 2 shorter than its argument.
the two-class problem, where $F^k$ gives no more information than $F_1^k$ (since $F_2^k = 1 - F_1^k$) and we may abbreviate the notation:

$$F^k = F_1^k$$
$$G^k = G_1^k$$
$$g^k = g_1^k.$$  

Similarly, the vector notation for the PPs may be dropped and we may replace $p_i$ by $p_i^{11}$, and further abbreviate the latter to $p_i$. It will be useful to simplify the notation further by making the following substitutions:

$$a_1 = p_{i-2}$$
$$a_2 = p_{i-1}$$
$$a_3 = p_i$$
$$a_4 = p_{i+1}$$
$$a_5 = p_{i+2}$$

Similarly, the vector notation for the PPs may be dropped and we may replace $p_i$ by $p_i^{11}$, and further abbreviate the latter to $p_i$. It will be useful to simplify the notation further by making the following substitutions:

$$b_2 = q_{i-1}^1$$
$$b_3 = q_i^1$$
$$b_4 = q_{i+1}^1$$

Also define:

$$a = (a_1, a_2, a_3, a_4, a_5)$$
$$b = (b_2, b_3, b_4)$$
$$c = (c_3)$$

and recall that

$$b_2 = F^1(a_1, a_2, a_3)$$
$$b_3 = F^1(a_2, a_3, a_4)$$
$$b_4 = F^1(a_3, a_4, a_5).$$

Associated random variables are written in upper-case. These random variables should be imagined in an inverted pyramid, extracted from fig. 4.3, as shown in fig. 4.4. Define the spaces from which $a, b, c$ are drawn as follows:

$$a \in A = [0, 1]^5$$
$$b \in B = [0, 1]^3$$
$$c \in C = [0, 1].$$

We may thus write:

$$F^1 : A \rightarrow B$$
$$F^2 : A \rightarrow C.$$  

The three spaces and their relationships are shown in fig. 4.4.

**Theorem 2** There exists at least one tooth-comb model for which $F^2 \neq G^2$.  


4.3. A STUDY OF SOME THEORETICAL LIMITS ON PRL

Figure 4.4: Simplified notation.

Proof outline

The following proof will show that it is not possible to express $F^2$ as the composition of $g^1$ with any function. This is established by finding a pair of values in the domain of $g^1$ that are mapped to the same value by $g^1$ but to different values by $g^2$. The proof is illustrated in fig. 4.5.

Proof

Recall that:

$$G^2 = g^2.g^1 = g^2.F^1$$

and that in our simplified notation:

$$F^1(a) = P(Y_i = 1 | B=b),$$
$$g^2(a) = P(Y_i = 1 | A=a).$$

Thence:

Theorem 2 $\Leftrightarrow$ 3 tooth-comb model : $$(\exists a : P(Y_i=1|A=a) \neq P(Y_i=1|B=b)).$$

By lemma 9 (see Appendix C), $P(Y_i=1|B=b)$ is a function of $b$ alone — call this function $h_t(b)$. So the above equation becomes:

Theorem 2 $\Leftrightarrow$ 3 tooth-comb model : $$(\exists a : P(Y_i=1|A=a) \neq h_t(b)).$$ (4.12)

It is therefore sufficient to find a tooth-comb model in which $P(Y_i=1|A=a)$ is not equal to any function $h$ of $b$, where

$$h : B \rightarrow C.$$
The assertion 4.12 will be established if we can find two points in $A$ which map onto the same point in $B$ under $F^1$, but map to different points in $C$ under $F^2$; $h$ could then not be a function. Formally it is sufficient to show:

$$\exists (a_\alpha, a_\beta \in A) : F^1(a_\alpha) = F^1(a_\beta),$$
$$F^2(a_\alpha) \neq F^2(a_\beta).$$

(4.13)

The situation is shown in fig. 4.5. We require to demonstrate this behavior for one particular instance of the tooth-comb model only. Here are the parameters of the model we will use:

$$D(Y_i | Y_{i-1} = 1) = (1/3, 2/3)$$

and for simplicity we will insist on equal priors which is achieved by:

$$D(Y_i | Y_{i-1} = 2) = (2/3, 1/3).$$

We now re-formulate and expand the definition of $F^1$ given by equations 4.7 to 4.9, in the notation of this proof, for the special case of equal priors, (ie $\pi_1 = \pi_2 = 1/2$) as follows:

$$F^1(a_1, a_2, a_3) = \frac{a_2.S(1, a_1, a_3)}{a_2.S(1, a_1, a_3) + (1 - a_2).S(2, a_1, a_3)}$$

(4.14)

where $S(...)$ represents the summation part of (4.7), which expands as follows:

$$S(y, a_1, a_3)$$

12 This simplification in the presentation of the proof was suggested by Dr. T. H. Westerdale.

13 This model in fact favours an alternation of class 1 and 2, though there is no significance in this.
and $t_{11}$ etc. are the model transition probabilities (recall that in the two class case, $P(Y_i = \alpha | Y_{i-1} = \beta) = P(Y_i = \alpha | Y_{i+1} = \beta)$):

\[
\begin{align*}
t_{11} &= P(Y_i = 1 | Y_{i+1} = 1) = 1/3 \\
t_{12} &= P(Y_i = 1 | Y_{i+1} = 2) = 2/3 \\
t_{21} &= P(Y_i = 2 | Y_{i+1} = 1) = 2/3 \\
t_{22} &= P(Y_i = 2 | Y_{i+1} = 2) = 1/3
\end{align*}
\]

Similarly for $F^2$:

\[
F^2(a_1, a_2, a_3, a_4, a_5) = \frac{a_3.S(1, a_1, a_2, a_4, a_5)}{a_3.S(1, a_1, a_2, a_4, a_5) + (1 - a_3).S(2, a_1, a_2, a_4, a_5)}
\]  \hspace{1cm} (4.15)

where

\[
S(y, a_1, a_2, a_4, a_5) = \begin{align*}
t_{11}t_{1y}t_{1y}t_{11}a_1a_2a_4a_5 + \\
t_{11}t_{1y}t_{1y}t_{21}a_1a_2a_4(1 - a_5) + \\
t_{11}t_{1y}t_{2y}t_{12}a_1a_2(1 - a_4)a_5 + \\
t_{11}t_{1y}t_{2y}t_{22}a_1a_2(1 - a_4)(1 - a_5) + \\
t_{12}t_{2y}t_{1y}t_{11}a_1(1 - a_2)a_4a_5 + \\
t_{12}t_{2y}t_{1y}t_{21}a_1(1 - a_2)a_4(1 - a_5) + \\
t_{12}t_{2y}t_{2y}t_{12}a_1(1 - a_2)(1 - a_4)a_5 + \\
t_{12}t_{2y}t_{2y}t_{22}a_1(1 - a_2)(1 - a_4)(1 - a_5) + \\
t_{21}t_{1y}t_{1y}t_{11}(1 - a_1)a_2a_4a_5 + \\
t_{21}t_{1y}t_{1y}t_{21}(1 - a_1)a_2a_4(1 - a_5) + \\
t_{21}t_{1y}t_{2y}t_{12}(1 - a_1)a_2(1 - a_4)a_5 + \\
t_{21}t_{1y}t_{2y}t_{22}(1 - a_1)a_2(1 - a_4)(1 - a_5) + \\
t_{22}t_{2y}t_{1y}t_{11}(1 - a_1)(1 - a_2)a_4a_5 + \\
t_{22}t_{2y}t_{1y}t_{21}(1 - a_1)(1 - a_2)a_4(1 - a_5) + \\
t_{22}t_{2y}t_{2y}t_{12}(1 - a_1)(1 - a_2)(1 - a_4)a_5 + \\
t_{22}t_{2y}t_{2y}t_{22}(1 - a_1)(1 - a_2)(1 - a_4)(1 - a_5)
\end{align*}
\]

Finally then, two points which satisfy 4.13, for the specific model given above, are:

\[
s_{\infty} = \begin{pmatrix} 17648663 & 705 & 116281 & 705 & 17648663 \\ 17663800 & 1000 & 183362 & 1000 & 17663800 \end{pmatrix}
\]  \hspace{1cm} (4.16)
for which, by 4.14 applied to the three sub-vectors of \( a_\alpha \) and \( a_\beta \) (ie \( \{(a_1, a_2, a_3), (a_2, a_3, a_4), (a_3, a_4, a_5)\} \) we obtain

\[
F^1(a_\alpha) = (1/2, 1/2, 1/2) \quad (4.19)
\]

\[
F^1(a_\beta) = (1/2, 1/2, 1/2). \quad (4.20)
\]

Note that indeed \( F^1(a_\alpha) = F^1(a_\beta). \) We also find, by 4.15:

\[
F^2(a_\alpha) = \frac{8279810308732369}{13661490665376338}. \quad (4.21)
\]

\[
F^2(a_\beta) = \frac{5381680356643969}{13661490665376338}. \quad (4.22)
\]

Note that \( F^2(a_\alpha) \neq F^2(a_\beta). \) This satisfies the conditions set out in 4.13.

The arithmetic calculations were performed by reduce, and may be inspected in appendix D.

\[\square\]

**Discovery of the counter example**

We now describe how the points \( a_\alpha \) and \( a_\beta \) were discovered.

Our approach is to find the *solution set* \( \mathbb{A}_y \subset \mathbb{A} \) of a point \( \mathit{y}' \in \mathbb{R} \), ie:

\[
\mathbb{A}_y = \{ a : a \in \mathbb{A}, F^1(a) = \mathit{y}' \}.
\]

The image of this set under \( F^2 \) we call \( \mathbb{C}_y \), ie:

\[
\mathbb{C}_y = \text{Im}_{F^2}(\mathbb{A}_y).
\]

The crux of the problem is in finding \( \mathbb{A}_y \) which involves finding the solutions for \( a \) to:

\[
F^1(a) = \mathit{y}'.
\]

Recalling the definition of \( F^1 \) given in 4.14, this would be a formidable task if undertaken by hand. However, access to the computer algebra system REDUCE (see [Rayna87]) was kindly provided by the mathematics department at QMC, making the problem tractable.

The steps are outlined below. The actual equations involved are not shown here; they may be seen in the log of the REDUCE run which forms appendix D.

We require to solve \( F^1(a) = \mathit{y}' \) for \( a \), ie:

\[
F^1(a_1, a_2, a_3) = \mathit{b}_2', \quad (4.23)
\]

\[
F^1(a_2, a_3, a_4) = \mathit{b}_3', \quad (4.24)
\]

\[
F^1(a_3, a_4, a_5) = \mathit{b}_4'. \quad (4.25)
\]
Solving for $a_1, a_3$ and $a_5$ in 4.23, 4.24, 4.25 using REDUCE gives us three complicated equations of the form\(^{14}\):

\[
\begin{align*}
    a_1 &= a_1(a_2, a_3, b_2') \\
    a_3 &= a_3(a_2, a_4, b_3') \\
    a_5 &= a_5(a_3, a_4, b_4').
\end{align*}
\] (4.26)

Finally, substituting 4.27 into 4.26 and 4.28 yields equations of the form\(^ {15}\)

\[
\begin{align*}
    a_1 &= a_1(a_2, a_4, b_2', b_3') \\
    a_3 &= a_3(a_2, a_4, b_3') \\
    a_5 &= a_5(a_2, a_4, b_4', b_3').
\end{align*}
\] (4.29)

The complete definitions of equations 4.26 to 4.28 may be seen in appendix D, annotated by the corresponding equation numbers. Equations 4.29 to 4.31 form a parameterized definition of $\mathcal{A}_Y$ in terms of $a_2, a_4$ and $\mathcal{U}$, as follows:

\[
\begin{align*}
    \mathcal{A}_Y &= \{ a : a \in \mathcal{A} \} \\
    a_1 &= a_1(a_2, a_4, b_2', b_3') \\
    a_3 &= a_3(a_2, a_4, b_3') \\
    a_5 &= a_5(a_2, a_4, b_4', b_3').
\end{align*}
\] (4.32)

We are justified in claiming that 4.32 is the whole solution set, since the above arguments show that the set (for a given $\mathcal{U}$) is fully parameterized by $a_2$ and $a_4$, and are valid in the abstract, regardless of the actual form of $F$. Note however that choosing any $a_2$ and $a_4$ in $[0, 1]$ does not guarantee to generate a valid point in $\mathcal{A}_Y$ as this may force one or more of $a_1, a_3$ or $a_5$ out of $[0, 1]$.

**Extension to the hair-brush model**

The above theorem applies to the 1-D tooth-comb model. However, being a negative theorem, it is easy to extend it to the 2-D hair-brush model, by noting that the tooth-comb model may be seen as a special case of the hair-brush model in which there are interactions in only one dimension, so the model degenerates to a collection of independent Markov chains. Examples of the textures generated by two such degenerate models may be seen in fig. 5.4 (c) and (f). Since we have shown that there exists at least one instance of the tooth-comb model for which $F_2 \neq G^2$, we may also state that the corresponding functions relating to the hair-brush model are not, in general, equal.

---

\(^{14}\) 4.26 should be read "$a_1$ is a function of $a_2, a_3$ and $b_2'$."

\(^{15}\) These substitutions are not shown in the REDUCE log since they result in extremely long expressions; it is more economical to perform the substitutions arithmetically.
Generalizing for the $k$th iteration

The above theorem tells us that $\text{IOU}$ has fallen behind the single stage context function (for the tooth-comb model) by the second iteration. But could it perhaps ‘catch up’ subsequently?

We would like to establish:

**Conjecture 3** $G^k \neq F^k$. for $k > 1$. assuming the tooth-comb model.

which seems very plausible. Better still:

**Conjecture 4** $g^{k+1} \neq F^{k+1}$. for $k > 1$. assuming the tooth-comb model.

which would be more powerful and include the previous conjecture.

No tangible progress towards a proof can be reported.

4.3.5 Towards closed-forms for $g^k$

Thus far we have considered the IOU scheme in the abstract, without specifying the actual update functions. The discussions are nevertheless relevant since, as will be seen in the next section, the update functions can be obtained in empirical form using the Lapwing system, and the theorems which have been developed place upper bounds on the performance of the process under particular image model assumptions.

It is tempting nevertheless to search for a closed-form expressions for $g^2, g^3$ etc, at least within the tooth-comb model. We might argue that, even if its computation is costly, then the functions may be tabulated, for given class transition probabilities and priors. The implicit assumption here is that $g^k, k \geq 2$ will not depend on the distribution of the raw data, and so the tables thus compiled will have general applicability. After-all, $g^1 (= F^1)$ did not depend on the distributions of the data.

Let us work towards an expression for $g^2$.

**Lemma 5** $g^2(b) = E[F^2(A)|B=b]$.

**Lemma 6**

$$g^2(b) = \frac{\int_{a \in \Delta_e} F^2(a), p(a)da}{\int_{a \in \Delta_e} p(a)da} \propto \int_{a \in \Delta_e} F^2(a), p(a)da.$$ 

The proofs to these lemmata, for the tooth-comb model and two classes, are given in appendix C.

Notice that $p(a)$, ie the distribution of the per-pixel PPs, appears in this form of $g^2$. As this distribution will clearly depend on the underlying data distributions, it suggests that these distributions must be known for $g^2$ to be fully defined. In fact we can assert:
Theorem 7 There exists at least one tooth-comb model for which the IOU function for the second iteration \( g^2 \) depends on the class-conditional distribution of the data, i.e., on \( d(X|Y=\alpha) \).

The proof is given in appendix C. In outline, the proof assumes lemma 6 and shows that there exists cases when \( p(a) \) in the numerator and denominator does not cancel. This establishes the theorem since \( p(a) \) will in turn depend on \( d(X|Y=\alpha) \). (Recall that due to the stationarity assumption, \( d(X_i|Y_\alpha=a) = d(X|Y=\alpha) \) for all \( i \).)

Discussion

This is a significant and surprising result. It means that whilst it is possible for a system to learn \( g^1 \) from an overlay (e.g., a map) only, it may not so learn \( g^2 \) (and presumably \( g^{12} \)) without a registered example image from the intended data source. Note that this is so even assuming the tooth-comb model. So for example, consider using Lapwing to learn the update functions \( g^1 \) and \( g^2 \) for forest detection from thematic mapper data. Imagine learning two sets of functions, one based on band 1 imagery and the another based on, say, band 5 imagery. The above theorem tells us that while the \( g^1 \) functions would be identical—, the \( g^2 \) functions would not necessarily be so. This theorem is demonstrated by test 13 in chapter 5.

4.3.6 Graphical presentations

We have shown that in order to write \( F^2 \) as a composite \( hF^1 \), then for at least one instance of the tooth-comb model, \( h \) cannot be a function. It would be interesting to know how ‘wide’ the relation is for some given points in \( B \) space; that is to visualize the minimum and maximum value of \( F^2(a) \) which can be associated with a given \( U' \). This is equivalent to asking for the minimum and maximum values of \( C_{U'} \). The difference between these two extremes represents the range of uncertainty in the possible values of \( F^2(a) \). Clearly, these values depend on \( U' \), which is a three dimensional quantity. In order to obtain a graphical presentation of the above mentioned uncertainty, we need to project the \( B \) space to a single dimension in some way. We can think of this as threading a one dimensional line, parameterized by \( \theta \) say, through the three dimensional \( B \) space. It seems reasonable to insist that this line begins at the point \((0,0,0)\) and ends at \((1,1,1)\). We would like it to take a path between these two corners of \( B \) so as to visit the ‘worst’ points of \( B \) space — that is the points where the range of \( C_{U'} \) is at its greatest. In view of symmetry considerations, if there is such a unique path then it must be one that maintains \( b_2 = b_4 \) since these two dimensions may always be exchanged without effecting \( C_{U'} \). In view of these considerations we choose the simplest path — i.e., the one that traverses the leading diagonal in a straight line, and
acknowledge that there may be other paths where a greater range in $C_{b'}$ is observed. We thus define a parameterization as follows:

$$b'(\theta) = (\theta, \theta, \theta), \theta \in [0, 1].$$

We would like to plot $\max C_{b'}(\theta)$ and $\min C_{b'}(\theta)$ (for a given model instance) against $\theta$. We would ideally like to obtain algebraic expressions for these two quantities, in terms of $\theta$. On the face of it, this should be tractable, since we have a parameterized expression for the set $C_{b'}$ in terms of $a_2$ and $a_4$ (via 4.29 to 4.31 and $F^2$), giving a two dimensional optimization (minimize and maximize) problem. This was attempted using REDUCE, but failed for two reasons:

- The sheer size of the parameterized expression for $C_{b'}$ overwhelmed REDUCE — it managed the partial derivatives but failed on their simultaneous solution.

- It appears that the stationary points $C_{b'}$ w.r.t $a_2$ and $a_4$ do not always lie inside $A$, so that the optimizing value may be at the boundary of $A$. This adds the complication that the optimizing function will be piece-wise.

However, it appears from the symmetry of the parameterization, that an optimal at other than $a_2 = a_4$ cannot be justified. This reduces the problem to one dimension, permitting a program solution — a crude, one dimensional search, at an adequate resolution to plot the graph. Figure 4.6 shows two such graphs, plotted for different transition probabilities. Observe the large gap between the minimum and maximum plots; this shows the degree of uncertainty in approximating $F^2$ by any PRL scheme.
4.4. TRAINED PROBABILISTIC RELAXATION (TPR)

The graph also shows the function $F^1(\theta')$; note how this falls outside the range of $C_b$ in places — the implications of this will be discussed in the next section.

4.3.7 The effect of iterating $F^1$

As was mentioned earlier, many proposed PRL schemes use a single updating function that is applied at all iterations. In the work already cited, the function is statistically justified for the first iteration only. We would like to show that $g^2$ cannot be $F^1$ for the tooth-comb model.

**Theorem 8** There exists at least one tooth-comb model for which $g^2 \neq F^2$.

**Proof**

This follows directly from theorem 7 which says that $g^2$ will in general depend on the data distributions; $F^1$ does not, and so the two cannot, in general, be equal. □

An alternative argument that is more intuitively satisfying, goes as follows:

By lemma 5 —

$$g^2(b) = E[F^2(A)|B=b].$$

The expected value of any function must lie between the minimum and maximum values of the function. Examining fig. 4.6, we see that there are $b$ values for which $F^1(b)$ lies outside the maximum and minimum values of $F^2(a)$ for any $a$ and given $b$. Thus $F^1(b)$ cannot equal $g^2(b)$ for all $b$. However this argument cannot stand as a proof of the theorem due to the unsupported assumption (that $a_2 = a_4$) that went into producing the graph — the range could in fact be greater and so encompass $F^1(b)$.

**Discussion**

This theorem makes it clear that any PRL scheme that simply re-applies $F^1$ at the second iteration must be sub-optimal. Also, the graph shows that $F^1(b)$ is consistently optimistic in that it returns probabilities that are closer to 0 (when $\theta < 0.5$) or 1 (when $\theta > 0.5$) than is justifiable. Thus, a probability image resulting from such a relaxation process would appear more ‘black and white’ than the contextual data supported.

4.4 Trained probabilistic relaxation (TPR)

Trained probabilistic relaxation is the name given (by the author) a practical implementation of the IOU scheme. TPR involves directly estimating the distribution $D(Y_C|q^k_C, q^k_N, q^k_S, q^k_E, q^k_W)$ from training data, for each required $k$. For this purpose, $q^k_C$ etc are treated just as if they
were pixel data, and presented to Lapwing's LEARNER program as a probability image (or a stack of probability images in the multiclass case).

To achieve this it is necessary to scale the *a posteriori* probabilitys \((q_{ik}^e \text{ etc})\) to lie in the range 0 to 255 — which is the output representation used by CLASSY and the LAP in any case. However, since these quantities no longer represent pixel intensities, the arguments used to derive the \(Em^\alpha\) criteria loose their validity. In particular, the assumption that pattern vectors will be transposed by a shift in the direction of the diagonal vector (see 2.11) is inappropriate. Never-the-less, the author believes that the general motivation for the \(Em^\alpha\) — to discourage splits through dense regions of the pattern space — is still valid, and so the \(Em^\alpha\) criteria is used, with \(\sigma\) and \(\sigma'\) (see 2.28) set to zero and 8 respectively (the latter selected from experience), so that the criteria no longer discourages splits perpendicular to \(U\).

The TPR algorithm is specified in data-flow form in fig. 4.7 and as functional pseudo-code in fig. 4.8. The reader may also find it useful to look at some of the sequences of images that are produced by TPR in chapter 5. Note that training data is reused, only the two training and pruning images are required throughout.

The training phase of TPR produces a sequence of probability trees \(T^1, T^2\) etc that directly implement the updating functions \(g^1, g^2\) etc (but see the caveats below). These trees may then be sequentially applied, after the initial tree \(T^0\), to subsequent images.

4.4.1 Claims to optimality

In so far as TPR is an implementation of the IOU scheme, it can claim to represent the best that can be achieved by any local probability updating scheme. This claim holds *providing* there exists a means of estimating the required *a posteriori* distributions. Chapter 2 has described one way that this may be done, but certainly makes no claim to being optimal — Lapwing's main design criteria were flexibility and efficiency at classification time. TPR could be implemented via other estimation techniques — the k-NN for example.

In practice the waters are further muddied. The derivation of the initial PPs is based on a window of pixels (see §2.2.1) and not simply on single pixel data. In subsequent iterations too, the neighbourhood model used by Lapwing will generally be larger than the four-connected neighbourhood that has been assumed in the previous theoretical work. Theses differences (should) cause TPR to perform better than the IOU scheme that assumes a four-connected neighbourhood, especially when the image does not conform to the convenient hair-brush model — ie almost always.
4.5. Summary

The main contribution of this chapter has been a theoretical analysis of probabilistic relaxation labelling (PRL). The incrementally optimal updating (IOU) scheme has been defined...
{ '***' is a macro that can be instantiated to 'training', 'prune' or 'test' }

function ***_IMAGE(i) of type Image;
{ Return a probability image for *** at the ith iteration. }
    ***_IMAGE(i) = if i=0 then
        Raw_***_data
    else
        CLASSY (TREE (i-1), ***_IMAGE(i-1));

function TREE (i) of type Probability tree;
{ Return the probability tree that generates ***_IMAGE(i+1) from ***_IMAGE(i) }
    TREE(i) = PRUNE ( prune.IMAGE (i),
        prune.overlay,
        LEARNER (training.IMAGE (i),
            training.overlay));

function LEARNER (image, overlay) of type Probability tree;
{ Generate a probability tree from the given image and overlay }

function PRUNE (image, overlay, tree) of type Probability tree;
{ Prune the given probability tree on the basis of the given image and overlay }

function CLASSY (tree, image) of type Probability image.
{ Apply the given tree to the image to produce another probability tree }

Figure 4.8: Functional pseudo-code describing TPR
4.5. SUMMARY

to be optimal — ie a sequence of updating functions tailored to a particular instance of particular image model and iteration, delivering the a posteriori class estimate given the local neighbourhood of current PPs. Since the IOU scheme is definitively optimal, anything it cannot achieve cannot be achieved by any PRL scheme. Although no closed-form expression has been obtained for the updating functions, a number of theorems have been established, and are summarized below.

- When data is class-conditionally independent, replacing the original data with the per-pixel PP vectors does not discard any information relevant to subsequent contextual classification (theorem 1).

- IOU cannot in general attain PP estimates that are equivalent to PPs conditioned on the original data, for the tooth-comb model (theorem 2).

- At the second iteration (and presumably beyond) the IOU functions depend on the data distributions, not only on the parameters of the Markov field (theorem 7).

- The IOU function for the second iteration (and presumably beyond), is not the same as the first over the generality of tooth-comb models (theorem 8). Re-applying the first iteration function appears to generates optimistic PPs.

It has been proposed that the IOU functions may be directly estimated in exactly the same way that the 'update' functions which saw the original data were estimated — using Lapwing. This results in a sequence of probability trees which may then be sequentially applied to production images. This technique has been called trained probabilistic relaxation (TPR).
Chapter 5

Results and Evaluation

5.1 Introduction

The results presented in this chapter are intended to demonstrate the capability and limitations of the Lapwing system described in chapters 2 and 3, including its use with TPR as described in section 4.4. In this introductory section the general conduct of the tests is described and performance metrics are defined. In sections 5.2 to 5.6, the application of Lapwing to a number of artificial and real problem domains is described, with discussion at the end of each group of tests. In §5.7 various modifications are made to the way in which the probability trees are grown or used, in order to sharpen our understanding of the method, and to demonstrate some of the theorems which were developed in chapter 4.

5.1.1 Artificial vs real test images

Many of the tests in this chapter involve artificially generated data; typically an image is constructed by masking two (different) uniformly textured images with a hand drawn binary overlay. See for example fig. 5.1. This procedure has a number of conveniences; no image capture facilities are required; the job of hand classifying training data is circumvented (as the image is produced from the overlay), and the character and difficulty of the classification task can be tightly controlled and repeated — by others if required.

However, such tests are something of a cheat — the real world is invariably more awkward than even our most contrived test cases. Further, artificial tests can often flatter the system; their un-natural character can make them appear particularly difficult to the human eye.¹

Any practical system must be demonstrated on real problems of interest. To this end, tests 7 and 8 involve classification problems from remote sensing and cytology.

¹Though some tasks of interest are un-natural; eg the eye (probably) did not evolve for the purpose of discriminating cultivated forest from 700 km up.
5.1.2 General notes on the conduct of tests

Each test requires three sample images with classification overlays, to be used in tree growing (training), tree validation (pruning) and test. 'Stage 1' results are those obtained by training and classification from the original image data. Subsequent stages are the results from trained probabilistic relaxation (TPR) as describers in section 4.4. Thus 'stage 2' results are obtained from the first iteration of TPR.

Some notes on image display conventions. All the tests concern two-class problems, enabling overlays and classification results to be shown in a single image. Class-1 \((Y = 1)\) is often referred to as foreground and class-2 as background. The image overlays show foreground in black, background in white. Intensities in input images are negated — 255 is represented by black, 0 by white. Similarly, probability-images are shown with 1.0 represented by black and 0.0 by white.\(^2\) \(^3\).

5.1.3 Performance statistics

For each test the following statistics are tabulated against classification stage (see eg fig. 5.1)

- Actual percentage classification error.
- Predicted percentage classification error.
- Conditional entropy of true scene given the probability-image.
- Internal estimate of the above entropy.
- The total number of nodes in the pruned tree.

In addition, the class priors and entropies are given for the training, pruning and test overlays.

These quantities are defined in the following sub-sections.

Actual classification error

This is the percentage of pixels incorrectly classified (as compared with a true scene overlay) using the MAP classification rule \(\mathcal{D}(q_i)\) based on the PPs in the probability-image.\(^4\)

\(^2\)These negations are to save the printer's ink!

\(^3\)When discussing performance statistics we are no longer interested in the spatial relationships of pixel — we are interested in their correct or incorrect classification in isolation. We may thus consider a whole probability image (at the \(k\)th iteration of TPR) and classification overlay to represent a statistical sample. The individual units of the sample are indexed by \(i\) and consist in the pair \((q^k_i, y_i)\). We may thus consider the probability image/classification overlay to represent a sample from the pair of random variables \((Q^k, Y)\)

\(^4\)Reminder: \(q^k_i\) is the \(a\ posteriori\) probability vector at pixel \(i\), at the \(k\)th classification stage, \(q^k_{i\alpha}\) is the class \(\alpha\) component of that vector. Random variables for these are written \(Q^k\) and \(Q^k_{\alpha}\) respectively. We shall often drop the \(k\).
Predicted classification error

This statistic represents the classification error one would expect to achieve for a given image, under the assumption that the \( q_t \) vectors in the probability image are in fact (overall) honest assessments of \( \mathbb{D}(Y|A_t) \), where \( A_t \) represents some unspecified quantity of image data around the \( i \)th pixel.

\[
\text{Predicted error-rate} = \frac{E[\# \text{ of incorrectly classified pixels}|q_1 \ldots q_N]}{N}
\]

\[
= \left(1 - \frac{E[\# \text{ of correctly classified pixels}|q_1 \ldots q_N]}{N}\right)
\]

\[
= \left(1 - \frac{\sum_i P(i \text{th pixel is correctly classified}|q_i)}{N}\right)
\]

\[
= \left(1 - \frac{\sum_i q_{i\mathcal{D}(q_i)}}{N}\right) \text{ by definition of } q_i
\]

where \( q_{i\mathcal{D}(q_i)} \) is the component of the \textit{a posteriori} probability vector selected by the decision rule \( \mathcal{D}(q_i) \); for the MAP decision rule this is simply the highest valued component.

The computation of 5.5 thus reduces to simply computing (one minus) the arithmetic mean of the highest \textit{a posteriori} probability in each of the vectors in the test image. The degree to which this predicted error-rate matches the actual error rate gives an indication as to the overall honesty of the generated PPs. See §2.3 for further discussion.

Actual conditional entropy

The conditional entropy \( H(Y|Q) \) is a measure of the uncertainty in \( Y \) after \( Q \) is known (see [Jones79] for an introduction to information theory). When computing \( H(Y|Q) \) we ignore the probabilistic meaning of \( Q \) and treat it simply as given data. Since in implementation the components of \( Q \) are quantized in 255ths, we avoid the problems of dealing with continuous variables, and consider \( Q \) to be discrete. By the definition of conditional entropy:

\[
H(Y|Q) = - \sum_{\alpha \in \Phi} \sum_q P(Y=\alpha|Q_\alpha=q)P(Q_\alpha=q)\ln P(Y=\alpha|Q_\alpha=q)
\]

where the second sum runs from 0.0 to 1.0 in 255ths. The quantity \( P(Y=\alpha|Q_\alpha=q) \) may be read as "the probability that a pixel has a particular class \( \alpha \) given that the \( \alpha \)th component of it assigned \textit{a posteriori} probability vector has a value \( q \)." 6 Thus the computation involves

---

3Intuition: If an honest (two class) classifier delivered a probability image in which all of the pp vectors had the value (0.1, 0.9), then we can see that the MAP decision rule would assign class two to every pixel, and that we would expect roughly 10% of them to be wrong.

6There is a subtle assumption here; strictly, to compute \( H(Y|Q) \) we should consider the whole \( Q \) vector when computing the given conditional probability — ie we should write \( \sum_q P(Y=\alpha|Q=q) \). The form
(for a two class problem) first constructing a $256 \times 2$ table for $D(Y|Q_\alpha)$ and a $256 \times 1$ table of $D(Q_\alpha)$, the values being estimated from the probability image under investigation and true scene overlay. The sum can then be computed directly.

**Predicted conditional entropy**

This is an *internal* estimation of $H(Y|Q)$ under the assumption that $Q$ is in fact represent an honest PP vector, ie that

$$q_\alpha = D(Y_i|\Delta_i).$$

(5.7)

Under this assumption we have, by lemma 10 (see appendix C):

$$P(Y=\alpha|Q_\alpha=q_\alpha) = P(Y=\alpha|\Delta) = q_\alpha, \text{ for each } \alpha.$$  (5.8)

This enables us to re-write equation 5.6 as:

$$H(Y|Q) = - \sum_{\alpha \in \Phi} \sum_{q} q_\alpha P(Q_\alpha=q_\alpha) \ln q_\alpha.$$  (5.9)

Note that 5.9 does not involve $Y$ and so the true scene overlay is not required. We expect this internal estimate of entropy to closely match the actual figure (from 5.6) for an honest classifier.

The relative merits of the above statistics will be discussed in section 5.8.

**Tree size**

The total number of nodes (terminal plus non-terminal) is given for each pruned tree.

**Prior statistics**

Two prior statistics are given for the three true scene overlays (training, pruning and test) used in each trial:

- the class prior probability vector $D(Y)$;
- the class entropy $H(Y) = \sum_y P(y) \ln P(y)$.

These are estimated directly from the classification overlay of the training sample. If the classifier fails to extract *any* information from the image data then we expect to see classification error equal to the lowest class prior of the test image and $H(Y|Q)$ to equal $H(Y)$.

---

used assumes that $Q$ contains no more information concerning the likelyhood of a pixel being class $\alpha$ than does $Q_\alpha$ alone. In two class problems this is clearly true (since $q_2 = 1 - q_1$); however, if $Q$ is the true *a posteriori* probability it claims to be, then it should also be so in multiclass problems. In fact all the results presented relate to two class problems in any case.

---

7 More precisely, in the no information case we expect the classification error to be equal to the prior of the test image corresponding to the class with the lowest prior in the pruning image. Usually, the lowest
5.2 Regular noisy texture discrimination

The tests in this section are composed of a regular texture corrupted with Gaussian noise.

Test 1 — Vertical vs horizontal lines with low (50%) noise

The underlying textures for this and the following tests are vertical and horizontal lines, one pixel wide and one pixel apart. The intensity of line and space is 170 and 86 respectively, i.e. the 'signal' is 84 units peak to peak. To this basic texture is added white Gaussian noise, \( \mu = 0, \sigma = 25 \) — i.e. the noise SD is 50% of the intensity variation in the texture. The input images are shown in fig. 5.1 and the resulting probability-images for the test data are shown in fig. 5.2.

<table>
<thead>
<tr>
<th>Stage</th>
<th>% Error</th>
<th>Entropy</th>
<th>Tree size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Actual</td>
<td>Predicted</td>
<td>Actual</td>
</tr>
<tr>
<td>1</td>
<td>4.51 (11.04)</td>
<td>3.48 (17.08)</td>
<td>0.134 (0.199)</td>
</tr>
<tr>
<td>2</td>
<td>1.44 (2.00)</td>
<td>0.93 (1.66)</td>
<td>0.053 (0.059)</td>
</tr>
<tr>
<td>3</td>
<td>1.33 (1.93)</td>
<td>0.85 (2.81)</td>
<td>0.050 (0.066)</td>
</tr>
<tr>
<td>4</td>
<td>1.38 (2.19)</td>
<td>1.48 (2.96)</td>
<td>0.062 (0.064)</td>
</tr>
<tr>
<td>5</td>
<td>1.42 (2.41)</td>
<td>0.98 (4.06)</td>
<td>0.054 (0.085)</td>
</tr>
<tr>
<td>6</td>
<td>1.42 (2.34)</td>
<td>0.96 (3.49)</td>
<td>0.055 (0.070)</td>
</tr>
</tbody>
</table>

Table 5.1: Results for test 1 — Directional texture, 50% noise (test 9 in paren.).

Test 2 — Vertical vs Horizontal lines with high (100%) noise

This is as for the previous test, but with a higher noise level — \( \sigma = 84 \), i.e. 100% of the texture signal. The test image and classification results are shown in fig. 5.3. The training images are omitted — they are as for the previous test, except for the addition of the higher noise levels.

Prior class will be the same in each case; if they are not then a classification error-rate of greater than 50% is possible.
CHAPTER 5. RESULTS AND EVALUATION

(a) Training image  (b) Training overlay

(c) Pruning image  (d) Test image

Figure 5.1: Inputs for test 1 — vertical vs horizontal lines with 50% noise.

5.2.1 Discussion of noisy texture tests

Test 1 with only 50% noise, presents the system with little difficulty. Even after the first stage the error-rate is less that 5%. The fact that the classes occur in large areas (strong contextual dependencies) enables the error-rate to be reduced to 1.4% after two applications of the trained relaxation process.

The addition of 100% noise means that $\approx 31\%$ of the pixels take on values on the ‘wrong’ side of the mean signal value (128); the eye has some difficulty in identifying the textures. The stage one error-rate of 29% (see table 5.2) is scarcely better than would be achieved with the ‘assign to highest prior’ rule (ie just assign every pixel to the class which was in the majority in the training sample), which would yield 43% error. It is therefore somewhat
5.2. REGULAR NOISY TEXTURE DISCRIMINATION

(a) Stage 1 probability-image  (b) Stage 2 probability-image

(c) Stage 6 probability-image  (d) True classification

Figure 5.2: Outputs for test 1 — vertical vs horizontal lines with 50% noise.

It is surprising that the trained relaxation process is able to improve things so dramatically, reducing the error to 3% on the first application and to 3% by the 7th stage. This is because of the strong spatial correlation in the true scene.

Note how after stage 3 there is little appreciable improvement in error rates, but neither is there any deterioration; the trees get smaller and have less and less effect. It is also satisfying to observe that except for the first stage (to be discussed later in §5.8.2), the actual and predicted error-rates match closely; the classifier is generating honest probabilistic assessments.

Lapwing has clearly performed well on this test; it must be remembered however that the test is highly artificial, tightly controlled, and in many ways is precisely the type of problem
on which we would expect the system to do well on. In fact, a tree with just 3 branch and
4 terminal nodes can produce reasonable first stage classification (though the probability
assignments would be coarse so subsequent stages would do less well).

5.3 Markov random field textures

The tests in this section involve the discrimination of first order binary Markov random field
(MRF) textures.
5.3. MARKOV RANDOM FIELD TEXTURES

<table>
<thead>
<tr>
<th>Stage</th>
<th>% Error</th>
<th>Entropy</th>
<th>Tree size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Actual</td>
<td>Predicted</td>
<td>Actual</td>
</tr>
<tr>
<td>1</td>
<td>29.10 (22.48)</td>
<td>20.66 (7.93)</td>
<td>0.520 (0.493)</td>
</tr>
<tr>
<td>2</td>
<td>4.72 (7.48)</td>
<td>4.89 (6.03)</td>
<td>0.138 (0.193)</td>
</tr>
<tr>
<td>3</td>
<td>3.37 (4.91)</td>
<td>3.37 (2.66)</td>
<td>0.093 (0.132)</td>
</tr>
<tr>
<td>4</td>
<td>3.07 (4.13)</td>
<td>3.17 (1.83)</td>
<td>0.085 (0.126)</td>
</tr>
<tr>
<td>5</td>
<td>2.99 (4.33)</td>
<td>3.37 (1.41)</td>
<td>0.085 (0.124)</td>
</tr>
<tr>
<td>6</td>
<td>4.43 (1.25)</td>
<td>(0.131)</td>
<td>(0.031)</td>
</tr>
</tbody>
</table>

Table 5.2: Results for test 2—Directional texture, 100% noise (test 10 in paren.).

5.3.1 Generation of MRF textures

The MRF textures used were generated by a form of the Metropolis algorithm as described in [Cross83] (similar techniques are described in [Hassner80] and [MonneSl]). In order to ensure that discrimination is only possible by textural characteristics, the prior probabilities of black/white are held constant at 0.5/0.5. Any first order binary MRF can then be specified by just two parameters — referred to in [Cross83] as \( b(1,1) \) and \( b(1,2) \).\(^8\) These control the degree of correlation in the horizontal and vertical directions respectively. Positive values encourage clumping, negative values encourage a black/white alternation and zero produces un-correlated pixels. If \( b(1,1) \) and \( b(1,2) \) are equal then the resulting textures are isotropic; unequal values produce an-isotropic, ie, directional, textures. Figure 5.4 shows some example textures as 128 square images. The generation process is fully described in the cited paper; the following is a brief summary — an understanding of the algorithm is not necessary for the purposes of this thesis.

An image is initialized randomly with (in our case) equal numbers of black and white pixels. A pair of pixels is selected at random\(^9\) and their values possibly exchanged. The exchange is performed if the ratio \( p(\text{new})/p(\text{old}) \) is greater than a random number selected uniformly from \([0,1]\), where \( p(\text{new}) \) and \( p(\text{old}) \) is the probability new (pixels exchanged) and original image respectively, given the two texture parameters. Thus, an exchange resulting in a more probable configuration is always performed, others may be. As a result of the Hammersly–Clifford theorem (see [Besag74]), the required ratio of probabilities can be easily calculated from an expression involving only the exchanged pixels and their immediate neighbours.

\(^8\)The notation is intended to deal with higher order fields, hence the apparent redundancy.

\(^9\)pixels of the same value are avoided.
CHAPTER 5. RESULTS AND EVALUATION

Test 3 — Correlated vs un-correlated Markov textures

The test comprises an un-correlated foreground texture (ie \( b(1,1) = b(1,2) = 0 \)) on a background MRF with parameters \( b(1,1) = b(1,2) = 2 \). Thus both textures are isotropic. See figures 5.5 5.6 and table 5.3.

<table>
<thead>
<tr>
<th>Stage</th>
<th>% Error</th>
<th>Entropy</th>
<th>Tree size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Actual</td>
<td>Predicted</td>
<td>Actual</td>
</tr>
<tr>
<td>1</td>
<td>11.06</td>
<td>13.18</td>
<td>0.296</td>
</tr>
<tr>
<td>2</td>
<td>2.79</td>
<td>2.12</td>
<td>0.092</td>
</tr>
<tr>
<td>3</td>
<td>2.48</td>
<td>1.74</td>
<td>0.081</td>
</tr>
<tr>
<td>4</td>
<td>2.66</td>
<td>1.68</td>
<td>0.078</td>
</tr>
<tr>
<td>5</td>
<td>2.49</td>
<td>1.74</td>
<td>0.083</td>
</tr>
</tbody>
</table>

Table 5.3: Results for test 3 — Correlated vs un-correlated MRF textures.

Test 4 — Directional small-scale Markov textures

Here the task is to distinguish anisotropic textures oriented at 90 degrees to one another. The foreground has parameters are \( b(1,1) = 1, b(1,2) = 0 \), and the background \( b(1,1) = 0, b(1,2) = 1 \). Thus the textures are relatively fine grained, each with slight correlation in one direction and none in the other. See fig. 5.7 and table 5.4

<table>
<thead>
<tr>
<th>Stage</th>
<th>% Error</th>
<th>Entropy</th>
<th>Tree size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Actual</td>
<td>Predicted</td>
<td>Actual</td>
</tr>
<tr>
<td>1</td>
<td>38.92</td>
<td>23.01</td>
<td>0.620</td>
</tr>
<tr>
<td>2</td>
<td>13.17</td>
<td>11.33</td>
<td>0.329</td>
</tr>
<tr>
<td>3</td>
<td>7.48</td>
<td>6.87</td>
<td>0.189</td>
</tr>
<tr>
<td>4</td>
<td>6.33</td>
<td>5.58</td>
<td>0.163</td>
</tr>
<tr>
<td>5</td>
<td>6.37</td>
<td>5.24</td>
<td>0.158</td>
</tr>
</tbody>
</table>

Table 5.4: Results for test 4. — Directional small-scale MRF textures.

Test 5 — Directional large-scale Markov textures

This is similar to the previous test, but with much coarser graining. The foreground parameters are \( b(1,1) = 1.2, b(1,2) = 2.8 \), the background is the reverse. See figure 5.8 and table 5.5
5.3. MARKOV RANDOM FIELD TEXTURES

<table>
<thead>
<tr>
<th>Stage</th>
<th>% Error</th>
<th>Entropy</th>
<th>Tree size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Actual</td>
<td>Predicted</td>
<td>Actual</td>
</tr>
<tr>
<td>1</td>
<td>35.46</td>
<td>21.46</td>
<td>0.605</td>
</tr>
<tr>
<td>2</td>
<td>14.81</td>
<td>12.78</td>
<td>0.383</td>
</tr>
<tr>
<td>3</td>
<td>9.58</td>
<td>8.64</td>
<td>0.255</td>
</tr>
<tr>
<td>4</td>
<td>7.72</td>
<td>7.69</td>
<td>0.218</td>
</tr>
</tbody>
</table>

(Priors as in test 1)

Table 5.5: Results for test 5. — Directional large-scale MRF textures.

5.3.2 Discussion of Markov texture tests

Whereas the previous set of tests (1 and 2) were particularly well suited to the Lapwing method, the irregular nature of Markov textures makes them, in some ways, particularly unsuited. No simple first stage tree exists as it did for the former tests; and the system can do little better than to chop up the pattern space as finely as is permitted by the available samples. Note the large tree sizes, eg 731 nodes in test 5, and 409 for test 3, the latter of which the eye actually finds quite easy.¹⁰

Tests 4 and 5 involve the discrimination of directional textures. In both cases the directionality is only slight, giving the eye considerable difficulty. As with test 2, the first stage results are very poor — with error-rates little better than that obtainable from no information. There is considerable improvement after trained relaxation — in test 4 the error-rate is reduced to approximately a fifth of its first stage value (from 39% to 6%).

¹⁰Conclusion: the eye isn’t restricted to linear operators!
Figure 5.4: Example MRF textures.
Figure 5.5: Inputs for test 3 — Correlated vs un-correlated MRF textures.
Figure 5.6: Outputs for test 3— Correlated vs un-correlated MRF textures.
Figure 5.7: Results for test 4 — Directional small-scale MRF textures.
Figure 5.8: Outputs for test 5 — Directional large-scale MRF textures.
5.4 Edge detection

Test 6 — Detection of straight edges in 50% additive noise

The task here is to identify edges of (closed) dark objects on a light background, corrupted by white Gaussian noise. The foreground has an intensity level of 64, the background level is 192 and the additive Gaussian noise has $\sigma = 64$ — ie 50% of the foreground/background intensity variation. The edge is considered to lie to the brighter side of the transition. The same basic image is used for training, pruning and test, but with independently generated noise (ie different random number seeds). For this test a different neighbourhood model was used — a $5 \times 5$ window at one resolution — ie $N_w = 5$, $N_t = 1$.\textsuperscript{12} The input images are shown in fig. 5.9 and the probability-image results in fig. 5.10. Performance statistics are given in table 5.6.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figures/figure5.9.png}
\caption{Inputs for test 6 — Edge detection.}
\end{figure}

5.4.1 Discussion of edge detection test

There are two particular problems associated with using supervised pattern recognition techniques for the detection of linear features. Firstly the priors will invariably be highly disparate so that a typical image will show a paucity of positive examples. Secondly, the approach does not allow for 'near misses' — a correct detection misplaced by one pixel will considered by the system to be as bad as any other misclassification.

\textsuperscript{11}Edge detection is a huge field — suggested starting points are [Marr82] and [Canny86]

\textsuperscript{12}Performance with the usual neighbourhood model ($N_w = 3$, $N_t = 2$), in fact were only slightly poorer — achieving a final error-rate of 3.90%.
Possibly for the above reasons these results are not startling. Certainly they do not compare with a purpose designed edge operator. For example, Petrou and Kittler [Petrou88] carry out a similar experiment, but with twice the noise level, to test various modifications to the Canny [Canny86] edge detector, and visually their results are somewhat better.\footnote{However their performance figures cannot be compared, since in the cited paper, near misses (detection within a 3 x 3 window of the true edge) are accepted as being correct.}

Observe the un-remarkable but steady improvement produced by TPR and the good agreement between actual and predicted error-rates, indicating that the PPs have a genuine statistical meaning.
Table 5.6: Results for test 6 — Edge detection.

<table>
<thead>
<tr>
<th>Stage</th>
<th>% Error</th>
<th>Entropy</th>
<th>Tree size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Actual</td>
<td>Predicted</td>
<td>Actual</td>
</tr>
<tr>
<td>1</td>
<td>4.43</td>
<td>4.45</td>
<td>0.131</td>
</tr>
<tr>
<td>2</td>
<td>3.78</td>
<td>3.69</td>
<td>0.111</td>
</tr>
<tr>
<td>3</td>
<td>3.43</td>
<td>3.30</td>
<td>0.105</td>
</tr>
<tr>
<td>4</td>
<td>3.29</td>
<td>3.16</td>
<td>0.102</td>
</tr>
<tr>
<td>5</td>
<td>3.15</td>
<td>2.91</td>
<td>0.102</td>
</tr>
</tbody>
</table>

Priors: \( D(Y) = (0.94, 0.06), \) \( H(Y) = 0.217 \) (nats).
5.5 Remotely sensed imagery

Test 7 — Urban area detection in Airborne Thematic Mapper (ATM) data

The purpose of this test is to assess Lapwing's usefulness in a practical application. The imagery used is of an area around Blewbury, UK, acquired by the Natural Environment Research Council (NERC) on the 4th of June 1984. The imagery comprises 11 spectral bands spanning the visible and infra-red, at a scale of $5 \times 5$ per pixel.\(^\text{14}\)

Reference data planes were prepared at the NERC unit for Thematic Mapper Information Systems (NUTIS), identifying 18 cover types. The data is described in detail in [Settle87b]. The author is grateful to NUTIS for making available the imagery and the invaluable reference data to this project.

For this trial only bands 7 and 11 are used. Classification overlays are derived from the class NUTIS call 'Residential (built up)'. Three (slightly overlapping) $256 \times 256$ sub-scenes were extracted from the supplied $512 \times 512$ image, to be used in training, pruning and test.

As the input image consists of two bands, the pattern vector has $N_b = 2$, with $2 \times 18 = 36$ components (see §2.2.1) in the first stage; subsequent stages revert to $N_b = 1$.

See figures 5.11, 5.12 and table 5.7.

<table>
<thead>
<tr>
<th>Stage</th>
<th>% Error</th>
<th>Entropy</th>
<th>Tree size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Actual</td>
<td>Predicted</td>
<td>Actual</td>
</tr>
<tr>
<td>1</td>
<td>9.27 (9.27)</td>
<td>7.91 (7.91)</td>
<td>0.239 (0.239)</td>
</tr>
<tr>
<td>2</td>
<td>8.38 (8.38)</td>
<td>6.63 (6.63)</td>
<td>0.215 (0.215)</td>
</tr>
<tr>
<td>3</td>
<td>7.81 (7.91)</td>
<td>5.76 (6.13)</td>
<td>0.219 (0.203)</td>
</tr>
<tr>
<td>4</td>
<td>7.81 (7.72)</td>
<td>5.13 (5.86)</td>
<td>0.209 (0.202)</td>
</tr>
<tr>
<td>5</td>
<td>7.83 (7.64)</td>
<td>5.64 (5.64)</td>
<td>0.205 (0.205)</td>
</tr>
</tbody>
</table>

Table 5.7: Results for test 7 (test 11 in paren.).

\(^\text{14}\)The whole image has already been seen in the introductory chapter.
5.5. REMOTELY SENSED IMAGERY

(a) Training image — layer 1  (b) Training image — layer 2

(c) Urban area training overlay

Figure 5.11: Inputs for test 7 — ATM.

5.5.1 Discussion of ATM tests

Urban area detection is recognized as being a difficult task for pixel classifiers (eg [Palgen70]) due to its spectral variability and thus the necessity to use textural/spatial features. Urban area is an excellent example of a 'super-class' as discussed in [Poole88e], with the intrinsic 'sub-classes' including concrete, slate, trees and grass (in parks and gardens) — indeed it is its very variability which gives perhaps the easiest clue to its identification. The class is difficult to define — eg, is a park within a city to be classified as urban?

This problem of definition is discussed by [Settle87b] in relation to the preparation of the reference data used in this trial. They decided not to attempt a detailed segmentation of
the class, effectively including all pixels within the town boundary.\textsuperscript{15} Looking closely at the test image it can be seen that some regions have been included in the urban area class that are quite indistinguishable from the non-urban class; quite possibly these are parks or large areas of waste ground. The author is aware of no generally accepted definition of urban area, and so that assumed by NUTIS seems as good as any. There is clearly a benefit in using their reference data, un-modified, since this permits objective comparisons between different techniques.

In light of the above discussion, the final error-rate of just below 8\% seems satisfactory. Notice also how even after the third application of trained relaxation, (ie by stage 4), regions of grey persist — ie the system continues up to admit to its uncertainty. This is also reflected in the reasonable agreement of actual and predicted error-rates.\textsuperscript{16}

The performance figures should however be considered in the light of two important qualifications:

1. Training and test image came from (different parts of) the same image involving (different parts of) the same town. The test is thus immune to problems resulting from varying atmospheric effects\textsuperscript{17}, changing seasons or differing character of terrain.

2. The test required two 256 × 256 classification overlays for training/pruning. Due to the cost of obtaining accurate ground truth most supervised classification techniques presently in use in remote sensing must make do with less training data than this.

\textsuperscript{15}Though they did segment out a pond which can be seen at the bottom of fig. 5.12(f).

\textsuperscript{16}These results were first reported in [Poole88d].

\textsuperscript{17}Though these factors can to some extent be compensated for by pre-processing — see eg [Lillesand87].
Figure 5.12: Results for test 7 — ATM.
5.6 Biological Images

Test 8 — Labeling of the cell nucleus in electron micrographs

This test is based on an electron micrographs of a human pancreas cell, and was kindly provided by the Medical Research Council (MRC) Human Genetics Unit, Edinburgh, UK. The task is to label the nucleus. The original 850 x 1000 pixel image was first reduced to half these dimensions by pixel averaging, and then three 256 x 256 sub-images selected for training, pruning and test. As these three sub-images had to overlap to some extent, the pruning and test images were rotated by 90 and 180 degrees respectively in order to increase their independence. The image overlays were prepared by hand by the author. The input images are shown below in fig. 5.13 and the resulting probability-images for the test image (5.13(d)) are shown in fig. 5.14.

<table>
<thead>
<tr>
<th></th>
<th>D(Y)</th>
<th>H(Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>(.68, .32)</td>
<td>.627</td>
</tr>
<tr>
<td>Pruning</td>
<td>(.43, .57)</td>
<td>.684</td>
</tr>
<tr>
<td>Test</td>
<td>(.67, .33)</td>
<td>.637</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stage</th>
<th>% Error</th>
<th>Entropy</th>
<th>Tree size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Actual</td>
<td>Predicted</td>
<td>Actual</td>
</tr>
<tr>
<td>1</td>
<td>18.53 (18.53)</td>
<td>21.79 (21.79)</td>
<td>0.290 (0.290)</td>
</tr>
<tr>
<td>2</td>
<td>7.84 (7.84)</td>
<td>10.97 (10.97)</td>
<td>0.189 (0.189)</td>
</tr>
<tr>
<td>3</td>
<td>5.16 (5.70)</td>
<td>6.89 (4.11)</td>
<td>0.143 (0.146)</td>
</tr>
<tr>
<td>4</td>
<td>4.14 (5.47)</td>
<td>5.05 (2.30)</td>
<td>0.121 (0.140)</td>
</tr>
<tr>
<td>5</td>
<td>4.24 (5.61)</td>
<td>5.17 (1.76)</td>
<td>0.112 (0.153)</td>
</tr>
</tbody>
</table>

Table 5.8: Results for test 8(test 12 in paren.).

5.6.1 Discussion of biological image test

The use of supervised pixel classifiers is less wide-spread in biological image processing than in remote sensing. This may be due to the fact that much of the imagery is effectively 'single band' — eg electron micrographs, ultra-sound scans, X-ray images etc, and so single pixel techniques can do little more than threshold slicing. Also, biological images are generally more structured so that model based approaches (ie knowing what you're looking for) will be more fruitful. Biological vision tasks are more naturally treated as problems of segmentation than of pixel classification.
In applying Lapwing to the problem of nucleus recognition, it is appreciated that the task could more sensibly tackled by other means. For example, since the nucleus boundary can be easily identified, and we know that the nucleus is a single closed region, a technique involving boundary detection and area fill seems appropriate; certainly this seems to be the way the eye tackles the problem. However, we should not fall into the trap of assuming that the way the eye does it is the only or even the best way — especially in such un-natural domains.

The final error-rate of just over 4% by stage 4 is encouraging. As with the ATM test, the trial lacks realism in that only one basic image was used throughout training and test. Note the considerable improvement produced by trained relaxation — from over 18% error.
On this occasion, the classifier is being consistently pessimistic, in estimating its error to be slightly higher than the actual value. This is probably because the test sub-image is slightly 'easier' than was the pruning image — containing more of the easily identified boundary region.

This test also shows that TPR may increase errors, albeit slightly (from 4.14% to 4.24% between stages 4 and 5). This results from a breakdown of the assumption that training, pruning and test (production) sample are drawn independently from the same population. In fact the three samples are obtained from three different regions of one image, and each region may have its own peculiar characteristics. This effect becomes more apparent as the number of iterations of TPR increases, since we are asking for statistical agreement over a
5.7. SOME EXPERIMENTS

larger and larger dimensioned pattern space. It is interesting that in this case, the predicted error-rate also began to increase at the same time as did the actual rate. It is not clear whether this would generally or usually be the case; if it were then the predicted error-rate could be used on production runs (since it is an internal estimate) to indicate when to stop the relaxation process.

5.7 Some experiments

5.7.1 The effect of training on an image which is more or less noisy

The following two experiments re-uses the images and probability trees from test 1 and 2.

Test 9 — Applying a tree grown for noisy data, to less noisy data

The trees grown in test 2 for 100% noise were used to classify the 50% noise image of test 1. The performance figures are tabulated in small print in the same table as the figures for test 1 (table 5.1), for comparison.

Test 10 — Applying a tree grown for less noisy data, to noisy data

The trees grown in test 1 for 50% noise were used to classify the 100% noise image of test 2. The performance figures are tabulated in small print in the same table as the figures for test 2 (table 5.2), for comparison.

Discussion of tests 9 and 10

In both cases, the final actual performance is poorer than for the correctly conducted tests. Where the tree intended for 50% noise is applied to 100% noise data (test 9), the classifier is over-confident in its predicted error-rates. The converse is the case in test 10, as would be expected.

Notice how in test 10 the first stage figures for actual error-rate are better than for the original test, but that they quickly fall behind.

Repeated application of the first TPR tree

Here we are interested in the effects of re-using the stage two tree (ie the first iteration of trained relaxation), for all subsequent stages, rather than growing a tree specifically for that stage. This is done for the ATM and Electron micrograph test set.
CHAPTER 5. RESULTS AND EVALUATION

Test 11 — Re-application on ATM data

Results are shown in table 5.7, in paren.

Test 12 — Re-application on Electron micrographs

Results are shown in table 5.8, in paren.

Discussion of tests 11 and 12

The purpose of this experiment is to demonstrate theorem 8 (§4.3.7), namely that the incrementally optimal updating functions must be tailored for each iteration of the process. In these two experiments we are effectively setting $g^2$ and $g^3$ to be $g^1$.18

The effect is more pronounced in test 12. We can see that the classifier becomes overconfident, whilst its actual performance has deteriorated. The over-confidence can be explained in terms of the classifier paying too much attention to neighbours, believing them to be more independent than they actually are; tailored updating functions are needed to deal with the reduced degree of independence which is an inevitable consequence of previous relaxation stages.

Test 13 — An experiment to demonstrate theorem 7

Recall that theorem 7 states that, under the tooth-comb model, the IOU function for the second iteration depends on the distribution of the data — not only on the parameters of the MRF. In order to demonstrate this for the hair-brush model an image was constructed which precisely conformed to this model. The true scene is a binary, isotropic MRF texture generated by the method described in §5.3.1, with parameters $b(1,1) = b(1,2) = 3$, and equal priors. (see fig. 5.15 (d)). The pixel data was generated from a Gaussian distribution for each class; class one has a $\mu_1 = 128$, $\sigma_1 = 30$; class two has $\mu_2 = \mu_1$ and $\sigma_2 = 60$. The PPs for first stage probability image (ie that which is assumed to be based on the per-pixel data) were computed directly by a simple program which was given the above Gaussian parameters, rather than by using Lapwing. (though Lapwing could have been used). Probability trees were then grown for the subsequent update stages using a version of Lapwing with only a simple 4-connected (C,N,S,E,W) neighbourhood model, ie with $d = 5$, in keeping with the assumed neighbourhood model of IOU. Training and pruning images are not shown — they are similar to test the image shown in fig. 5.15, which also shows the generated probability-images. The results are shown in table 5.9.

18 Reminder — $g^2$ advances from stage 2 to stage 3 in these tests.
In order to test the theorem another image was generated in the same way as above but with the distributions exchanged — i.e., $\mu_1 = 60, \mu_2 = 30$. Again the initial PPs were calculated from the exchanged distributions and the same trees as above applied to generate stages 2 to 5. Theorem 7 tells us that the optimal updating functions for stages 2 and beyond should be different between the two cases — thus by using a tree sequence which was not grown specifically for this distribution, we expect to see inferior results. The generated images and true scene are shown in fig. 5.16 and the performance statistics in table 5.10.

![Test image](image1)

![Stage 1 probability-image](image2)

![Stage 5 probability-image](image3)

![True classification](image4)

Figure 5.15: Outputs for test 13 — 'A'.
(a) Stage 1 probability-image  
(b) Stage 2 probability-image  
(c) Stage 3 probability-image  
(d) Stage 5 probability-image

Figure 5.16: Outputs for test 13 — 'B'.
5.7. SOME EXPERIMENTS

<table>
<thead>
<tr>
<th>Stage</th>
<th>% Error</th>
<th>Entropy</th>
<th>Tree size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Actual</td>
<td>Predicted</td>
<td>Actual</td>
</tr>
<tr>
<td>1</td>
<td>33.95 (33.95)</td>
<td>33.46 (33.46)</td>
<td>0.599 (0.599)</td>
</tr>
<tr>
<td>2</td>
<td>23.56 (23.61)</td>
<td>23.11 (21.16)</td>
<td>0.485 (0.482)</td>
</tr>
<tr>
<td>3</td>
<td>18.69 (18.91)</td>
<td>17.77 (8.42)</td>
<td>0.403 (0.408)</td>
</tr>
<tr>
<td>4</td>
<td>16.75 (17.27)</td>
<td>15.90 (3.52)</td>
<td>0.372 (0.400)</td>
</tr>
<tr>
<td>5</td>
<td>15.91 (16.72)</td>
<td>14.87 (2.04)</td>
<td>0.355 (0.407)</td>
</tr>
</tbody>
</table>

Table 5.9: Results for test 13 — ‘A’ (test 14 results in paren.).

<table>
<thead>
<tr>
<th>Stage</th>
<th>% Error</th>
<th>Entropy</th>
<th>Tree size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Actual</td>
<td>Predicted</td>
<td>Actual</td>
</tr>
<tr>
<td>1</td>
<td>33.95</td>
<td>33.58</td>
<td>0.599</td>
</tr>
<tr>
<td>2</td>
<td>32.40</td>
<td>30.13</td>
<td>0.539</td>
</tr>
<tr>
<td>3</td>
<td>33.61</td>
<td>20.64</td>
<td>0.490</td>
</tr>
<tr>
<td>4</td>
<td>35.71</td>
<td>16.01</td>
<td>0.455</td>
</tr>
<tr>
<td>5</td>
<td>38.19</td>
<td>12.88</td>
<td>0.438</td>
</tr>
</tbody>
</table>

Table 5.10: Results for test 13 — ‘B’.

Discussion of test 13

The original and switched versions of this test will be denoted ‘A’ and ‘B’ respectively. If the updating function did not depend on the data distributions then we would expect to see comparable results for A and B since the class statistics are identical in each case. Clearly this is far from the case; the A results are very respectable showing a steady improvement and maintaining acceptable overall honesty (agreement between actual and predicted error-rates). In B however, results from the third stage onwards show a deterioration whilst the predicted error-rates become wildly dishonest. Remember that the stage 3 results are those generated by $g^2$ and their gross dishonesty is indication that the function is not valid on this distribution, as is predicted by the theorem.

Recall that by theorem 1, $g^1$ (from stage 1 to 2) is not expected to depend on the distributions. It is at first disconcerting, therefore, that the second stage results do not agree between A and B (24% against 32% error). However, the crucial point is that the results remain honest (32% actual and 30% predicted error). This shows that a valid probabilistic
quantity is still being generated. The fact that the error-rate is poor shows that the data is not being fully exploited. This is because when the tree was grown, certain areas of the pattern space were sparsely populated with samples and so could not be very finely partitioned. In B however that area turned out to be strongly represented, but the coarse partitioning did not exploit significant variations in the a posteriori distributions.

The test gives a striking demonstration of theorem 7 — notice the visible deterioration of the images in ‘B’, showing that the effect cannot be treated as insignificant.

It is worth noting that the data distributions chosen for this experiment were asymmetric w.r.t class. It is suspected that a symmetrical change does not affect $g^2$.

### 5.7.2 Comparing TPR with a computational PRL scheme

Here we compare TPR with a particular PRL scheme developed by Haralick [Haralick83a]. In our notation, assuming a four connected neighbourhood, Haralick’s update function is as follows:

$$q_{C\alpha}^{k+1} \propto q_{C\alpha}^k \prod_{i \in N} \sum_{\beta \in \Phi} q_{\beta}^k R_{C,i}(\alpha,\beta)$$

(5.10)

where $N = \{N, S, E, W\}$ is the neighbourhood set, and $R_{i,j}(\alpha,\beta)$ are correlation coefficients defined as:

$$R_{i,j}(\alpha,\beta) = \frac{P(Y_i=\alpha, Y_j=\beta)}{P(Y_i=\alpha)P(Y_j=\beta)}$$

$i, j \in N$. The constant of proportionality is removed by normalization.

For a binary, isotropic MRF we have $R_{C,N}(\cdot,\cdot) = R_{C,S}(\cdot,\cdot) = R_{C,E}(\cdot,\cdot) = R_{C,W}(\cdot,\cdot)$ — ie the subscripts may be dropped. Under these restrictions it is thus only necessary to determine $R(1,1)$, $R(0,0)$ and $R(1,0)$.

This function has been implemented as a simple Pascal program, that generates the $k + 1$th image from the $k$th, given the above three coefficients. For brevity this particular implementation of PRL (ie equation 5.10) will be referred to as ‘H-PRL’.

**Test 14 — H-PRL applied to the hair-brush model image**

The update function just described was applied to the image from the previous test (fig. 5.15(b)). The three correlation coefficients were estimated from the true scene (fig. 5.15(d)); they were $R(1,1) = 0.493, R(0,0) = 0.493, R(1,0) = 0.61$. The generated images are omitted but the performance figures may be seen in paren. in table 5.9. The calibration curves (see §2.3) for the third iteration of TPR and Haralick’s PRL are shown in fig. 5.17.

---

19 these coefficients can almost certainly be expressed in terms of the parameters which originally generated the MRF (see §5.3.1) — an exercise for the reader!
Test 15 — H-PRL applied to the ATM data

In this test, Haralick's PRL function is applied to the stage 1 probability image (fig. 5.12 (c)) delivered by Lapwing. The correlation coefficients were estimated from the training overlay (fig. 5.11 (c)); they were $R(1, 1) = 3.72, R(0, 0) = 1.36, R(1, 0) = 0.015$. The performance figures shown in table 5.11.

<table>
<thead>
<tr>
<th>Stage</th>
<th>% Error</th>
<th>Entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Actual</td>
<td>Predicted</td>
</tr>
<tr>
<td>1</td>
<td>9.27</td>
<td>7.90</td>
</tr>
<tr>
<td>2</td>
<td>10.35</td>
<td>1.68</td>
</tr>
<tr>
<td>3</td>
<td>10.68</td>
<td>0.37</td>
</tr>
<tr>
<td>4</td>
<td>10.77</td>
<td>0.17</td>
</tr>
<tr>
<td>5</td>
<td>10.84</td>
<td>0.12</td>
</tr>
<tr>
<td>10</td>
<td>10.99</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 5.11: Results for test 15 — H-PRL applied to ATM data.
Discussion of test 14 and 15

The main point to notice in these tests is the high degree of overall dishonesty in the generated PPs — the actual and predicted error-rates rapidly diverge, the latter being over confident. The calibration curve further exhibits this. In test 14, where the image is constructed precisely to match the assumptions of the hair-brush model, the actual performance of H-PRL is not significantly different from that of TPR. However, in test 15 which represents a more realistic application, H-PRL actually increases the error-rate, even at its first application. After ten iterations it had increased the error-rate from 9% to 11% while predicting 0%.

The two tests represent opposite extremes in their expected propriety to PRL. In the latter, errors are highly correlated, violating the assumptions upon which the scheme was derived. Even so, the fact that in neither case did (this particular version of) PRL outperform TPR, and that in both cases it led to highly over-confident PPs, does nothing to recommend the method. Note also that each iteration took approximately 6 minutes, which is slightly slower than TPR using Lapwing.

5.8 Entropy vs Classification error

Throughout this chapter the discussion of performance has been in terms of classification error-rate — $P(\text{error}|\Delta)$; it is the most obvious and commonly used metric for assessing classifiers (see eg [Congalton83]). Yet the metric has some problems associated with it. The first indication of the problem is visible in the definition both of predicted (5.5) and actual error. The fact that they both involve the max operator (in the MAP decision rule $D()$) indicates that they are discontinuous step functions. When a classifier returns only a categoric decision there is little option but to use this metric, but when a probabilistic assessment is made there may be more information than is indicated by $P(\text{error}|\Delta)$.

Since we require a measure of information it is expedient to consider information theoretic measures such as entropy or conditional entropy $H(Y|\Delta)$ which was introduced earlier. Here $\Delta$ can stand either for the PPs ($Q$), or the underlying data on which they were calculated, since $Q = P(Y = 1|\Delta)$ and by lemma 11 we have $H(Y|Q) = H(Y|\Delta)$. The fact that the definition of $H(Y|\Delta)$ is continuous at least, is promising. We know that when $H(Y|\Delta) = 0$, $\Delta$ removes all uncertainty in $Y$ and thus perfect classification is possible. When $H(Y|\Delta) = H(Y)$ then $\Delta$ has no relevance whatsoever to $Y$.

---

20The work described in this section was carried out jointly with Derek Long and Paul Otto. Originality is not claimed (though specific references have been difficult to find, but see [Vajda69]). It is included here for the insight it gives to the relationship between entropy and error-rate.
5.8.1 Relationship between $P(\text{error}|\Delta)$ and $H(Y|\Delta)$

How is $H(Y|\Delta)$ related to $P(\text{error}|\Delta)$? For example, do any of the following hold:

- $P(\text{error}|\Delta) = 0 \iff H(Y|\Delta) = 0$?
- $P(\text{error}|\Delta) = \min_y \pi_y \iff H(Y|\Delta) = H(Y)$?
- $P(\text{error}|\Delta_a) = P(\text{error}|\Delta_b) \Rightarrow H(Y|\Delta_a) = H(Y|\Delta_b)$?

In order to gain an intuitive understanding of the relationship between these quantities, a simple program was written. Both predicted error-rate and entropy depend only on the transition matrix $M(\Delta|Y)$ and priors $D(Y)$. On a given run of the program, the number of states of $Y$ and $\Delta$ was fixed at $m$ and $l$ respectively and the prior distribution $D(Y)$ specified. Different relationships between $y$ and $\Delta$ can then be postulated by randomly generating $M(\Delta|Y)$, subject to the usual normalizing constraints. For each version of the matrix the metrics $P(\text{error}|\Delta)$ and $H(Y|\Delta)$ are calculated and plotted one against the other. Typically 10000 such points are plotted. The scatter plots produced from six runs of the program are shown in fig. 5.18. From these results we may infer the following:

1. $H(Y|\Delta) \mapsto P(\text{error}|\Delta)$ is a many-to-many relation.

2. $H(Y|\Delta) = 0 \iff P(\text{error}|\Delta) = 0$ in all cases since all graphs converge at 0,0.

3. When the prior distribution is uniform, (the three plots on the left), worst-case entropy $\Leftrightarrow$ worst-case error-rate, since these points converge to a point at the top right extremity.

4. When the priors are unequal (plots on the right), worst-case error rate corresponds to a range of possible entropy values.

5. When $l > 2$, the upper bound on error-rate appears to be a linear function of entropy, since the upper envelopes on the plot seems to be a straight line.

6. When $m = 2$ the lower envelopes appear to be the same, regardless of $l$ or $Y$ priors.

Points 1 to 4 can be easily demonstrated analytically. The 5th point has only been proven, by the author, for the $m = l = 2$, equal priors case, and this limited proof is omitted. The gradient of the line is clearly worst-case error / worst-case entropy. It is a curious and rather unexpected result.

The final point stands on the 'solid' evidence provided by overlaying several plots and holding them to the light!
In addition we have equations for the upper and lower (cf point 6 above) envelopes in the \( m = l = 2 \), equal priors case ([Long88]). Abbreviating entropy to \( H \) and error-rate to \( p \), the curves are expressed as functions from \( p \) to \( H \):

\[
H_{\text{max}} = -(p \ln p + (1-p) \ln(1-p)) \quad \text{(5.11)}
\]

\[
H_{\text{min}} = \frac{1}{2} (1 + 2p) \ln(1 + 2p) - p \ln(2p). \quad \text{(5.12)}
\]

These curves are overlaid on plot (a).

In practical terms it is perhaps the fourth point which is of most interest. It warns us that some data might provide useful information (ie \( H(Y|\Delta) < H(Y) \)), and yet still fail to elevate the error-rate above the worst-case \( (1 - \max_y \pi_y) \) level. The phenomenon is more pronounced the more unequal are the priors, as a piece of evidence must be the more convincing in order to overthrow the classifier's 'prejudice' for simply selecting the majority class.

This 'latent' information is visible through \( H(Y|\Delta) \) \(^{21}\) and may show itself in improved error-rate when combined with other data. This is particularly relevant in our case, as the contextual combination of PPs by trained relaxation, may substantially reduce a poor, or even worst-case error-rate, and may eventually show itself in improved error-rate (see eg test 4).

5.8.2 Discussion of entropy metric for preceding tests

The above discussion enables us to better explain some of the preceding results. Consider the results for test 4 shown in table 5.4. The figures for actual and predicted error in the first generation are rather disparate — 39% and 23% respectively, indicating a seriously optimistic classification at that stage, the disparity reducing to reasonable level as the trained relaxation process proceeds. However corresponding figures for entropy — actual and predicted, agree very closely throughout. The probable explanation is as follows. In this test the raw image data is very 'poor', so that at the first stage there is never enough evidence to dislodge the classifier from its prior notion that 'its probably background (white)' which had a prior probability of 0.79 in the pruning image, hence the predicted error-rate of \( 1 - .79 \approx 22\% \). \(^{22}\)

\(^{21}\)If \( \Delta_a \) and \( \Delta_b \) are conditionally dependent given \( Y \), then it is possible for each to individually contain no information w.r.t \( Y \) (ie \( H(Y|\Delta_a) = H(Y) \)) and yet to completely disambiguate \( Y \) in combination — ie, \( H(Y|\Delta_a, \Delta_b) = 0 \). In image terms, this indicates a dependence on texture. Note however that such texture, information cannot be extracted when \( \Delta_a \) and \( \Delta_b \) are seen only through their PPs \( D(Y|\Delta) \).

\(^{22}\)This is anthropomorphic; to be more precise, the PPs produced from the evidence of one window of data, ie \( P(Y = 2|x) \), never exceeded the decision threshold of 0.5. Alternatively, the MAP decision rule for this particular problem can be written as:

\[
D(x) = 2 \quad \text{if} \quad \frac{p(x|Y = 2)}{p(x|Y = 1)} > \frac{0.79}{0.21} \quad \text{else} \quad 1.
\]

Clearly the ratio of class-conditional distributions must be pronounced for the \( 0.79 \) \( 0.21 \) threshold to be crossed.
Unfortunately for the classifier, the test image turned out to have a rather lower proportion
of background — 0.56, and so greater errors were produced. The conditional entropy metric
does not have this discontinuous character and so is more immune to the effects of priors —
hence the better agreement of these figures.

Test 2 show a similar behavior.

A rather different anomaly is exhibited by the biological image test (test 8, see table 5.8).
In this case actual and predicted error-rates agree reasonably, but the entropies do not, the
predicted value being rather pessimistic. A possible explanation is as follows. In real imagery,
there may well be a great variability within classes, and this variation may occur as distinct
modes (or sub-classes) — eg the nucleus seems to be made up of at least two different types of
regions — dark close to the nuclear membrane and textured internally. This multi-modality
does not give Lapwing any problems per-se — the probability tree approach is particularly
suited to this type of problem. However, not all sub-classes may be represented in the
training image, and thus if it appears in test, the pixels will (hopefully) make their way to a
leaf node which assigns a non-committal PP (0.43 say). Now, if no other leaf node assigns
0.43 (quite possible given $\frac{1}{255}$ quantization), then we have $P(\text{nucleus}|Q=0.43) = 1$, and this
will force down the apparent actual entropy.

The above suggests that it may be expedient to smooth the probabilities in some way
before calculating the entropy metric, though more work is needed to justify the degree of
smoothing required.

5.9 Summary

In this chapter Lapwing, including the trained probabilistic relaxation (TPR) technique, has
been applied, to several image classification problems. The test titles are listed below:

Test 1 — Vertical vs horizontal lines with low (50%) noise

Test 2 — Vertical vs Horizontal lines with high (100%) noise

Test 3 — Correlated vs un-correlated Markov textures

Test 4 — Directional small-scale Markov textures

Test 5 — Directional large-scale Markov textures

Test 6 — Detection of straight edges in 50% additive noise

Test 7 — Urban area detection in Airborne Thematic Mapper (ATM) data

Test 8 — Labeling of the cell nucleus in electron micrographs
CHAPTER 5. RESULTS AND EVALUATION

Test 9 — Applying a tree grown for noisy data, to less noisy data

Test 10 — Applying a tree grown for less noisy data, to noisy data

Test 11 — Re-application on ATM data

Test 12 — Re-application on Electron micrographs

Test 13 — An experiment to demonstrate theorem 7

Test 14 — H-PRL on a hair-brush model image

Test 15 — H-PRL applied to the ATM

Overall, the results are encouraging, with TPR proving to be effective in exploiting context to reduce classification error while maintaining honest PPs (as shown by the reasonable agreement between actual and predicted error-rates). However, the tightly controlled nature of the tests must be borne in mind. The demonstration of theorem 7 by test 13 is particularly satisfying.

The use of conditional entropy as a performance metric has been considered, with the suggestion that it may be a more useful indicator of performance than classification error. A brief study of the relationship between conditional entropy and classification error has been presented.
(a) $m = 2, l = 2, D(y = (.5,.5)$

(b) $m = 2, l = 2, D(y = (.26,.74)$

(c) $m = 2, l = 5, D(y = (.5,.5)$

(d) $m = 2, l = 5, D(y = (.26,.74)$

(e) $m = 3, l = 5, D(y = (.33,.33,.33)$

(f) $m = 3, l = 5, D(y = (.11,.31,.56)$

Figure 5.18: Scatter plots of $H(Y|\Delta)$ (x-axis) vs $P(\text{error}|\Delta)$ (y-axis). Plot (a) also shows equations 5.11 (lower) and 5.12 (upper).
Chapter 6

Conclusions

This thesis has two distinct strands:

1. The method of \textit{a posteriori} probability estimation in a high dimensioned pattern space, Lapwing, described in chapter 2.

2. The theoretical study of probabilistic relaxation labelling (PRL) in chapter 4, resulting in the proposal for trained probabilistic relaxation — TPR.

The unifying link between these strands is that one way of implementing TPR is through Lapwing. In fact, TPR suggested \textit{itself} once Lapwing had been constructed, and this then motivated the investigation into its properties, leading to the assertion that (sampling issues aside) TPR represented an optimum amongst PRL schemes, and stimulated questions about just what this optimum was.

For the purposes of drawing conclusions and pointing to future work, however, the two strands will be considered separately.

6.1 Lapwing

6.1.1 Putting Lapwing into context

The essential elements of the system are as follows:

- It is intended for supervised, image pixel classification, in particular where textural dependence is involved.

- There is very little sophistication in the feature extraction — raw data values are plucked from the local neighbourhood of each pixel. All the difficulties are handed on to a sub-system which must perform \textit{a posteriori} probability estimation in a high dimensioned pattern space.
CHAPTER 6. CONCLUSIONS

- The adopted solution involves the hierarchically partitioning of the pattern space — ie, growing a probability tree.

- Partitions are restricted to hyperplanes, although these may be in any orientation.

- The problem of hyperplane selection at each node is posed as one of optimization, the two main criteria being:
  - purity (by class) in each partition;
  - avoidance of high density regions of the pattern space, and preference for an orientation of the hyperplane that will be insensitive to a change in overall pixel intensity.

- A stochastic optimization strategy is used — a genetic algorithm.

6.1.2 Further work (1)

Improvements to the detailed design and implementation

The following would probably be of benefit:

- Permit a tree to deal directly with multiple class problems — this would be more efficient than growing/applying multiple trees.

- Permit training on multiple images and keep ephemeral statistics for each image (such as average intensity) which may be incorporated into the pattern vectors. It would also then be possible to make better use of training data by presenting the image/overlay in four orientations, where rotational invariance was required.

- Consider using non-linear partitioning functions — eg ranking operators (but note that this would invalidate the derivation of the $E_m$ criterion).

- Give the system a user interface!

'Semi-supervised learning'

It has already been noted that $E_m$ is an unsupervised criterion as it does not require knowledge of the true scene. This opens up the possibility of using unlabelled samples in training. In many applications it may be too costly to prepare wall-to-wall training overlays; however, unlabelled samples could still contribute to the $E_m$ criterion, with labelled ones contributing to both $E_m$ and Gini as usual. The unlabelled samples might be useful in defining significant pattern clusters that are not adequately portrayed by the labelled data. Some nodes might accumulate a large number of samples, none or few of which were labelled;
such nodes could be brought to the attention of an operator (by highlighting the pixels on
the training image), who might then wish to assign a subjective probability vector to that
node. Taken to the extreme, when no pixels are originally labelled, this strategy becomes one
of pure unsupervised clustering with post-classification of the clusters, and is recommended
by several authors (eg [Patrick72]). It would be interesting to see how Lapwing performed
in this completely unsupervised mode, using only the $E_m$ criterion.

**Sampling issues**

An important issue not explored in this thesis, is that of sampling, in particular the criterion
upon which the probability tree is pruned. Remember that in the present system, a split
is simply removed if either of its decedents captures fewer than some arbitrarily decided
number of validation samples. It must be possible to improve on this! A $\chi^2$ test of signif­
icance perhaps? How should the pruning criterion be affected by the dimensionality of the
pattern space (ie the width/depth/resolution of the neighbourhood model), and the number
of degrees of freedom available to the partitioning functions?\(^1\)

Such questions are the bread-and-butter of statistical sampling theory and standard
results must surely exist. Note, however that the problem is complicated by the statistical
dependencies within the training and validation sample sets; pattern vectors are correlated
(marginally and class-conditionally) because of their overlap in extraction, and classes are
marginally correlated because of the (assumed) contextual interactions. These correlations
could be avoided by using samples from only one coding set [Besag86], although it seems a
pity to waste training samples.

While this thesis is unable to offer any theoretical arguments, it can at least point to test
results, and in particular the good agreement between actual and predicted error rates, to
suggest that sampling problems are not a major problem in practice.

6.2 PRL, IOU and TPR

In chapter 4 a novel method of exploiting contextual dependencies has been proposed —
*trained probabilistic relaxation* (TPR), which is related to other proposed versions of proba­
bilistic relaxation labelling (PRL). Why yet another PRL scheme?

6.2.1 Comparing PRL and TPR

Conventional PRL schemes ‘fail’ at three quite distinct levels:

\(^1\)A recent paper by Fukunaga and Hayes [Fukunaga89] may offer a useful starting point, though it has
not been studied.
1. They impose a particular image model from the outset — typically the hair-brush model with its first order MRF model of class interactions and class-conditionally independent data. This is a restrictive model for many domains, especially when the classification task is rather ill-posed, with classes in reality being composed of distinct sub-classes, or where the noise is correlated for some other reason.

2. No computationally-feasible updating function has yet been proposed for the first iteration which does not impose further restrictive assumptions upon the MRF model. This is essentially because of the intractable nature of the non-causal field.

3. The update function derived for the first iteration is re-applied at subsequent iterations, when the assumptions that were made to derive it are no longer valid. In particular, whereas the PPs input to the first iteration are derived from (assumed) class-conditionally independent data, this will not be the case at subsequent iterations as they will by then be based on an overlapping window of data. Furthermore, these shortcomings (in particular 1. and 3.), appear to err in the same way; they conspire to generate PPs that are over confident. Thus, the component PPs all go scurrying off toward 0 or 1, but lose their statistical meaning — they are not honest PPs. This over confidence causes the PRL process to converge rapidly, and indeed this is often reported as being a desirable feature. We ought to be asking — "... it might converge, but does it converge to the right answer?" [Otto88]. In previous work the honesty of the generated PPs is seldom addressed, and as may been seen in tests 14 and 15, is often lacking in proposed implementations.

TPR suffers from none of the above shortcomings, ie:

1. It makes no model assumptions so they cannot be invalid.

2. The problems of non-causality do not concern it as the method is not computational.

3. Separate probability trees are grown for each iteration based on 'data' that is representative for that iteration.

So, is TPR a panacea for all ills? Hardly! Assuming that 'there's no such thing as a free lunch', we must determine just what we have traded for the above gains.

The relationship between PRL and TPR is similar to that between parametric and non-parametric, density estimation methods. In the former, a restrictive model is fitted — eg the univariate Gaussian — which has a small number of parameters (μ and σ). The assumed model may be inappropriate, but at least only a few samples are needed to fit the distribution with a reasonable degree of confidence. As a model allows greater flexibility, so it must involve a greater number of parameters, which must, therefore, require more samples to fit it to a
given degree of confidence. The choice of which model to use in any given circumstance will depend (computational problems aside) on the model's propriety to the problem and the number of samples available to fit it. This is a familiar mathematical modelling problem: is an accurate solution to an approximate model as good as an approximate solution to an accurate model? Where the number of samples is small an approximate model may yield more accurate estimations than a (poorly fitted) general one.

PRL represents a parametric approach — the parameters of the model being the 'compatibility coefficients' which control the updating functions. These are typically few in number (especially if the MRF is assumed to be isotropic) and can easily be estimated from a small true scene (note that the image data is not needed). Methods also exist for estimating these parameters from the image data alone, with no true scene — an obvious bonus. (See for example [Peleg80b], [Besag86], [Chittineni81], [Chellappa82] or [Yamamoto79].) TPR on the other hand needs a large amount of training data (image and true scene overlay) and in some applications this may not be available. Worse still, there is as yet, little guidance to indicate just how many samples are likely to be needed. We see therefore that TPR trades modeling, for sampling difficulties.

The discussion so far has been concerned with the modelling difficulties in conventional PRL (ie our original point 1 above). The second point, however, is not a statistical but a computational problem. Suppose we are prepared to accept the hair-brush model. As was remarked in section §4.3.2, the first stage update function is a perfectly respectable function, depending only on the parameters of the MRF. Indeed, it can be approximated to any required degree of accuracy (by summing over all combinations of label realizations in a suitably wide window), but at enormous computational cost. Perhaps an appropriate analogy is the analytic intractability of integrating the Gaussian; that problem is (presumably) solved by numerical integration and tabulation of the results (or fitting a polynomial). Could perhaps a similar approach be applied to our problem? (See the Further work section below.) TPR avoids these problems, exchanging computational difficulties for still more sampling overheads.

At subsequent iterations we know that the optimal updating function will depend not only on the parameters of the MRF but also on the underlying data distributions (theorem 7). Attempts were made to obtain a computationally based update function for the second iteration (ie $g^2$), under the tooth-comb model, but this met with little success and is not reported. TPR side-steps the problem — again by direct estimation. Clearly therefore, the estimation problems become more and more acute at each iteration.

---

$^2$Observing lemma 7, $g^2$ depends on the joint distribution $p(g)$. Even establishing the marginal distribution $p(a_i)$ for two classes with Gaussian distributed data is a considerable challenge which the reader may like to contemplate. It is tractable
6.2.2 Further work (2)

Finding an approximation to $F^1$ for the hair-brush model

(See above). A two class, binary MRF may be specified by just two parameters plus one of the priors ([Cross83], or see 5.3.1). Denote these parameters by $\theta$. The function has five parameters — $(p_C, p_N, p_S, p_E, p_W) = \theta$. We require an approximation to:

$$F^1_\theta(p) = D(Y_C|P=p)$$

There appears to be three sub-problems:

1. Discovering some means of normalizing the dependence on $\theta$ — ie deriving a (hopefully simple) function $h$ such that $F^1_\theta(p) = F^1_\theta(h_\theta(p))$. (Analogous to $\Phi_{\mu,\sigma}(x) = \Phi_{0,1}(\frac{x-\mu}{\sigma})$). It might turn out that $h$ can reduce the dimensionality of $p$ from 5 to 3, in view of symmetry.

2. Finding a computer large enough to calculate $F^1_\theta(.)$ to a sufficient accuracy at a number of points in the parameter space.

3. Fitting a polynomial to the tabulated values.

This is highly speculative.

Why use any form of relaxation labelling?

Recall that the object-centered view of context aims to achieve a classification which is conditioned on all the data in the system (all the pixels in the image). We have seen that the theoretical IOU scheme cannot achieve this (theorem 2). Computational approaches are unable to match IOU for even simple models, and TPR seems to have considerable (as yet little understood) sampling problems associated with it, which compound at each iteration. We might write, informally, $PRL < TPR < IOU < object-centered$ classification.

All this begs the question — why bother? Why not perform the estimation in one go, directly from a suitably large window of raw data? The usual answer to this is that direct estimation in such a large pattern space is 'impractical'. Yet this thesis has suggested that the computational problems associated with large pattern spaces can be tackled. The sampling problems are more fundamental. Is it possible to justify the use of TPR over direct estimation, given a finite sample size, on statistical arguments alone?

For a sufficiently large sample size we know that TPR will be inferior to direct estimation, so the answer to this question is not a forgone conclusion.
6.2. PRL, IOU AND TPR

Tighter theorems

The theorems that have been developed are rather ‘weak’ in that they generally say only that such-and-such is not equal something else — they are essentially negative theorems. It might be possible to put tighter bounds on the performance of the IOU scheme, within an information theoretic framework. For example

- Can we find expressions, in terms of tooth-comb (or hair-brush) model parameters, for the following:
  
  1. \( H(Y_i|{\mathcal{P}}_{i-w,i+w}) = H(Y_i|{\mathcal{F}}^w({\mathcal{P}}_{i-w,i+w})) \);
  
  2. \( H(Y_i|{\mathcal{G}}^w({\mathcal{P}}_{i-w,i+w})) \);
  
  3. \( \lim_{w \to \infty} H(Y_i|{\mathcal{P}}_{i-w,i+w}) \);
  
  4. \( \lim_{w \to \infty} H(Y_i|{\mathcal{G}}^w({\mathcal{P}}_{i-w,i+w})) \).

In particular, the difference between 3. and 4. would indicate the shortfall of information extracted by the IOU scheme, as compared with a full object-centred classification.

- To what degree do the IOU functions depend on the original data distributions? Can it be shown that the IOU functions are constant within some family of data distributions?

- Does the IOU scheme converge towards the message-centered solution (unlikely)?

Better proofs

The proofs presented in chapter 4 (especially for theorems 2 and 7) have an ‘unsatisfactory’ feel to them. The use of computer algebra to solve page-long equations certainly lacks subtlety, and as a result it is difficult to learn anything new from the proofs. A good proof should convince the intuition, which these do not. There can be little doubt that more elegant, simple proofs exist (based on graph-theoretic arguments perhaps). Note also that some of the theorems still require formal extension into the 2-D and multi-class case, and conjectures 3 and 4 have yet to be established.

Extension to general (irregular) Markov networks

It might be difficult to justify the additional work outlined above if the results are restricted in application to image data — ie to the tooth-comb or hair-brush model. However, these are just specific examples of general Markov networks which are of interest to probabilistic reasoning in artificial intelligence [Pearl88]. When the variable interactions can be represented as an acyclic belief network then elegant computational solutions exist. The presence of cycles, however, present similar theoretical and practical difficulties to those encountered
in the hair-brush model. Could the theory of incrementally optimal updating be extended
to general Markov networks? This would be an ambitious goal, which would require as a
pre-requisite, the ‘better proofs’ alluded to above.

6.3 Concluding remarks

The most significant contribution of this thesis, is the study of optimal PRL presented in
section 4.3. This has developed the theory of PRL within a precise statistical framework.
The resulting theorems make it clear that a full object-centered classification is unattainable
by any PRL scheme.

This is not to say that PRL should rejected as an approach to contextual classification
— it might still be the best practical technique available. Rather, these results should direct
the attention of future research towards finding feasible implementations of the IOU scheme,
rather than striving for an unattainable goal via ‘messy’ approximations. Particularly useful,
is the warning that the IOU functions (beyond the first stage) will depend on the actual data
distributions — even within the qualitative constraints of the hair-brush model; this should
save others some futile effort!

The thesis does, however, have other facets. Following is a list of those aspects the author
holds to be original and/or of particular interest:

• The concept of classifier honesty (§2.3) is not new to statisticians (or meteorologists)
  but does not appear to be well-know within the pattern recognition community.

• It has been demonstrated that probability trees offer a practical solution to the problem
  of obtaining fast, a posteriori probability estimates in a high dimensioned pattern space
  (§2.5). Points of specific interest are :

  — the general approach to finding partition surfaces by the optimization of criteria
    which include an unsupervised measure for ‘cluster phobia’;
  — the application of a genetic algorithm to this optimization task; more appropriate
    strategies must exist, but the robustness of the GA allowed unfettered investiga-
    tion of the criteria function.

• The direct implementation of a probability tree on an SIMD machine is novel (§3.2).

• The theoretical study of optimal PRL, already discussed, is original (§4.3).

• Trained probabilistic relaxation (TPR) implemented via Lapwing (§4.4), has been
  shown to be effective, and may be recommended.
• The analysis of the relationship between conditional entropy and classification error (§5.8) is not original, but the presentation in terms of scatter-graphs is instructive.

Taken together, these contributions offer improved practical pattern recognition techniques, and advance our understanding of contextual image classification.
Appendix A

Glossary

A.1 Abbreviations

ATM  airborne thematic mapper
CCI  class-conditional independence
GA  genetic algorithm
GML  Gaussian maximum likelihood
IOU  incrementally optimal updating
k-NN  kth nearest neighbour
LAP  linear array processor
MAP  maximum a posteriori probability
MD  minimum-distance
MRF  Markov random field
NERC  Natural Environment Research Council
NUTIS  NERC unit for thematic-mapper information systems
NPL  National Physical Laboratories
NN  nearest neighbour
PRL  probabilistic relaxation labeling
PP  a posteriori probability
PPL  picture processing language
QMC  Queen Mary College (London)
RAM  random access memory
SIMD  single-instruction, multiple data
TM  thematic mapper
TPR  trained probabilistic relaxation
### A.2 Notation introduced in chapter 2

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td>random variable (r.v) for extracted pattern vector</td>
</tr>
<tr>
<td>$x_i$</td>
<td>pattern vector extracted at $i$</td>
</tr>
<tr>
<td>$\Omega$</td>
<td>pattern space, $X \in \Omega$</td>
</tr>
<tr>
<td>$d$</td>
<td>dimensionality of $\Omega$</td>
</tr>
<tr>
<td>$l$</td>
<td>maximum pixel intensity ($\Omega = [0..l]^d$)</td>
</tr>
<tr>
<td>$Y$</td>
<td>r.v for a pixel’s class label</td>
</tr>
<tr>
<td>$y_i$</td>
<td>class realization at $i$</td>
</tr>
<tr>
<td>$m$</td>
<td>Number of classes</td>
</tr>
<tr>
<td>$\Phi$</td>
<td>Set of classes ${1...m}$, $Y \in \Phi$</td>
</tr>
<tr>
<td>$N_w$</td>
<td>basic width of neighbourhood model</td>
</tr>
<tr>
<td>$N_r$</td>
<td>number of resolution levels in neighbourhood model</td>
</tr>
<tr>
<td>$N_b$</td>
<td>number of images in “stack”</td>
</tr>
<tr>
<td>$t$</td>
<td>a tree node</td>
</tr>
<tr>
<td>$T_t$</td>
<td>sub-tree below and including $t$</td>
</tr>
<tr>
<td>$\Omega_t$</td>
<td>pattern sub-space represented by node $t$</td>
</tr>
<tr>
<td>$S_t(x)$</td>
<td>scalar function</td>
</tr>
<tr>
<td>$c_t$</td>
<td>threshold applied to $S_t(x)$</td>
</tr>
<tr>
<td>$v_t$</td>
<td>convolution vector at node $t$</td>
</tr>
<tr>
<td>$D_t(x)$</td>
<td>partitioning function at node $t$, $= [S_t(x) &lt; c_t]$</td>
</tr>
<tr>
<td>$i(t)$</td>
<td>Gini impurity index at node $t$</td>
</tr>
<tr>
<td>$Em$</td>
<td>proportion mis-split</td>
</tr>
<tr>
<td>$Em''$</td>
<td>refined criterion based on $Em$</td>
</tr>
<tr>
<td>$Z(v_t)$</td>
<td>simplicity criterion of convolution vector $v_t$</td>
</tr>
<tr>
<td>$I(t)$</td>
<td>impurity criterion at node $t$</td>
</tr>
<tr>
<td>$N_T$</td>
<td>number of pixels in test image</td>
</tr>
<tr>
<td>$N_C$</td>
<td>number of pixels in training image</td>
</tr>
<tr>
<td>$T^k$</td>
<td>set of image pixels at kth iteration of TPR</td>
</tr>
<tr>
<td>$\delta_i$</td>
<td>some unspecified data around pixel $i$</td>
</tr>
<tr>
<td>$\delta$</td>
<td>some unspecified data</td>
</tr>
<tr>
<td>$\Delta$</td>
<td>r.v representing some unspecified data</td>
</tr>
<tr>
<td>$D(\delta)$</td>
<td>decision rule based on data $\delta$, $D(.) \in \Phi$</td>
</tr>
<tr>
<td>$N_{vm}$</td>
<td>minimum number of validation samples allowed in a node</td>
</tr>
<tr>
<td>$D(Y)$</td>
<td>distribution (or probability vector) over r.v $Y$</td>
</tr>
<tr>
<td>$d(Y)$</td>
<td>density distribution over continuous valued r.v $X$</td>
</tr>
</tbody>
</table>
## A.3 Notation introduced in chapter 4

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_i$</td>
<td>r.v for data at $i$</td>
</tr>
<tr>
<td>$x_i$</td>
<td>realization of data at $i$</td>
</tr>
<tr>
<td>$X_{a,b}$</td>
<td>$(X_a \ldots X_b)$</td>
</tr>
<tr>
<td>$x_{a,b}$</td>
<td>$(x_a \ldots x_b)$</td>
</tr>
<tr>
<td>$Y_i$</td>
<td>r.v representing class at $i$</td>
</tr>
<tr>
<td>$y_i$</td>
<td>realization of class at $i$</td>
</tr>
<tr>
<td>$Y_{a,b}$</td>
<td>$(Y_a \ldots Y_b)$</td>
</tr>
<tr>
<td>$y_{a,b}$</td>
<td>$(y_a \ldots y_b)$</td>
</tr>
<tr>
<td>$\alpha, \beta$</td>
<td>class realizations — $\alpha, \beta \in \Phi$</td>
</tr>
<tr>
<td>$\pi_\alpha$</td>
<td>$P(Y=\alpha)$</td>
</tr>
<tr>
<td>$p_{i\alpha}$</td>
<td>$P(Y_i=\alpha</td>
</tr>
<tr>
<td>$P_{i\alpha}$</td>
<td>r.v for above</td>
</tr>
<tr>
<td>$p_i$</td>
<td>$D(Y_i</td>
</tr>
<tr>
<td>$P$</td>
<td>r.v for above</td>
</tr>
<tr>
<td>$p_{a,b}$</td>
<td>$(p_a \ldots p_b)$</td>
</tr>
<tr>
<td>$q_{i\alpha}^k$</td>
<td>$P(Y_i=\alpha</td>
</tr>
<tr>
<td>$q_i^k$</td>
<td>$D(Y_i</td>
</tr>
<tr>
<td>$Q^k$</td>
<td>r.v for above</td>
</tr>
<tr>
<td>$q_{i-w,i+w}^k$</td>
<td>$(q_{i-w}^k \ldots q_{i+w}^k)$</td>
</tr>
<tr>
<td>$p(x_i</td>
<td>\alpha)$</td>
</tr>
<tr>
<td>$P(\alpha</td>
<td>\beta)$</td>
</tr>
<tr>
<td>$g_\alpha^k$</td>
<td>IOU function for class $\alpha$ at $k$th stage</td>
</tr>
<tr>
<td>$g^k$</td>
<td>abbreviation for $g_1^k$</td>
</tr>
<tr>
<td>$G_\alpha^k$</td>
<td>$g_\alpha^k g_{\alpha-1}^k \ldots g_1^\alpha$</td>
</tr>
<tr>
<td>$G^k$</td>
<td>abbreviation for $G_1^k$</td>
</tr>
<tr>
<td>$G^k$</td>
<td>IOU distribution function for $k$th stage</td>
</tr>
<tr>
<td>$F_\alpha^w$</td>
<td>w-wide contextual smoothing function for class $\alpha$</td>
</tr>
<tr>
<td>$F^w$</td>
<td>abbreviation for $F_1^w$</td>
</tr>
<tr>
<td>$F^w$</td>
<td>w-wide contextual smoothing vector-function</td>
</tr>
<tr>
<td>$\Pi$</td>
<td>$A \Pi B$ reads &quot;$A$ is statistically independent of $B$&quot;</td>
</tr>
</tbody>
</table>
A.3.1 Abbreviated notation used in proofs

\[ A_i = P_i \]
\[ B_i = Q_i^1 \]
\[ C_i = Q_i^2 \]
\[ A = (P_1 \ldots P_5) \]
\[ B = (Q_1^1 \ldots Q_4^1) \]
\[ C = (Q_3^2) \]
\[ a_i = p_i \]
\[ b_i = q_i^1 \]
\[ c_i = q_i^2 \]
\[ a = (p_1 \ldots p_5) \]
\[ b = (q_1^1 \ldots q_4^1) \]
\[ c = (q_3^2) \]
\[ \xi_{ab} = P(X_i = a | X_{i+1} = b) \]
Appendix B

Statistical independence diagrams
(‘I-graphs’)

These are a way of representing graphically the statistical interdependencies between a collection of related random variables. We introduce them here informally, by example. See [Pearl88] for full details.

![Diagram of an example I-graph](image)

Figure B.1: An example I-graph.

When two nodes are completely disconnected then they are marginally independent of one another. Eg, in the above diagram :-

\[ E \perp \!\!\!\perp G, \quad \{A, B, C, D\} \perp \!\!\!\perp \{F, G\}. \]

If a node (or set of nodes) \( Z \) separates two nodes \( X \) and \( Y \), then \( X \) and \( Y \) are independent given \( Z \) — written \( X \perp \!\!\!\perp Y | Z \). Eg :-

\[ B \perp \!\!\!\perp E | D, \quad A \perp \!\!\!\perp D | (C, B), \quad \{A, B, C, D\} \perp \!\!\!\perp E | D. \]

Note that the converse implication does not necessarily hold — ie there may be (marginal or conditional) independencies which the graph does not represent.
Appendix C

Proof of theorems

C.1 Some theorems in second level probabilities

In this section we establish some very basic theorems \(^1\) which will enable us to manipulate conditional probabilities where the conditioning quantity is its self a probability — eg

\[
P(Y = y | Q = q), \text{ where } Q = P(Y = y | X).
\]

The reader should keep in mind that regardless of the identities developed in this section, such quantities as the above have a literal meaning, in terms of a limiting estimate from a suitable experiment — TPR (§4.4) does precisely this. The following lemma provides us with a means of testing the equality of two two probabilistic quantities (on \(Y\) say), one of which is conditioned directly on a random variable (\(X\) say, ie \(P(Y = y | X = x)\)), the other on some 'munging' function of the same random variable (\(h(X)\) say, ie \(P(Y = y | h(X) = h(x))\)). That this equality will hold when the function is reversible, is clear enough, but the lemma establishes a more powerful result, that for the equality to hold it is necessary and sufficient to show that the original conditional probability can be expressed as a function of the munged random variable which does not depend (directly) on \(x\).

The issue appears closely related to the idea of a 'sufficient statistic' (see eg [Mood74], pg 168), and indeed it ought to be possible to use the theorems from this area; however those available in the cited text did not seem to match our requirements, and so reluctantly, we establish the following:

**Lemma 9** For (discrete) random variable \(Y\) with any realization \(y\) and (discrete or continuous) random variable \(X\) with realization \(x \in \Omega\), and any (discrete or continuous) function \(h\) with domain \(\Omega\):

\(^1\)Originality is not claimed for these results, and indeed the author offers apologies for having been unable to find established theorems which must surely exist.
\[ \forall x \in \Omega \ P(Y = y | h(X) = h(x)) = P(Y = y | X = x) \iff \exists f : (\forall x \in \Omega, f(h(x), y) = P(Y = y | X = x)). \]

**Proof**

The following proof assumes \( X \) to be discrete — to obtain the proof for continuous \( X \), replace \( \sum \) with \( \int \) and \( P(X = x) \) with \( p(X = x)dx \).

\[ \Rightarrow' : f : q, y \mapsto P(Y = y | h(X) = q) \text{ satisfies result} \]

\[ \Leftarrow' : \]

Let \( Q = h(X) \), then, for any given \( q \in \text{Im}_h \Omega \)

\[
P(Y = y | Q = q) = \frac{P(Y = y, Q = q)}{P(Q = q)} = \frac{P(Y = y, Q = q)}{\sum_{h(x) = q} P(X = x)} P(Y = y, X \in (\bigcup_{h(x) = q} x)) = \frac{\sum_{x : h(x) = q} P(Y = y | X = x) \cdot P(X = x)}{\sum_{h(x) = q} P(X = x)} \]

(Xs mutually exclusive) = \[
\sum_{x : h(x) = q} P(Y = y | X = x) \cdot P(X = x)
\]

(by hypothesis, \( P(Y = y | X = x) = f(q, y) \)) = \[
\sum_{h(x) = q} f(q, y) \cdot P(X = x) \frac{\sum_{h(x) = q} P(X = x) \cdot f(q, y)}{\sum_{h(x) = q} P(X = x)} = f(q, y).
\]

(q fixed) = \[
P(Y = y | Q = q) = P(Y = y | X = x)
\]

\[ \square \]

The following lemma may be obvious or subtle to the reader; to the author, it shifts strangely in and out of focus ... .

**Lemma 10** \( Q = P(Y = y | X) \Rightarrow P(Y = y | Q = q) = q. \)
Proof

This follows from a direct application of lemma 9, setting \( h(x) = P(Y = y | X = x) \), and the required \( f \) on the R.H.S is simply \( q, y \to q \).

The following lemma concerns conditional entropy \( H(.|.) \). The lemma is not required for the main argument of the thesis, but mentioned ‘in passing’ in §5.8, and is interesting in its own right.

Lemma 11 \( Q = D(Y' | X) \Rightarrow H(Y' | Q) = H(Y' | X) \).

Proof

\[
H(Y' | Q) = - \sum_y \int_q P(Q = q) p(Y' = y | Q = q) \ln(P(Y = y | Q = q)) \, dq
\]

(by lemma 10)

\[
= - \sum_y \int_q p(Q = q) q_y \ln(q_y) \, dq
\]

(Xs mut. excl.)

\[
= - \sum_y \int_q \sum_{x : Q(x) = q} P(X = x) q_y \ln(q_y) \, dq
\]

(by hypothesis)

\[
= - \sum_y \sum_x P(X = x) P(Y = y | X = x) \ln(P(Y = y | X = x)) \, dq
\]

\( (q_y \text{ is a fn. of } x) \)

\[
= - \sum_y \sum_x P(X = x) P(Y = y | X = x) \ln(P(Y = y | X = x))
\]

\[
= H(Y' | X)
\]

\( \square \)

C.2 Proofs for chapter 4

Lemma 5 \( g^2(b) = E[F^2(a) | b] \)

Proof

Note that as \( b \) is a function of \( a \) (ie \( a \) determines \( b \)), then the event \( [A=a] \) is equivalent to the joint event \( [A=a, B=b] \). Now,

\[
E[F^2(a) | b] = \int_{a \in A} F^2(a) . p(a | b) da
\]

\( (F^2(a) = P(Y = 1 | a)) \)

\[
= \int_{a \in A} P(Y = 1 | a) . p(a | b) da
\]

\[
= P(Y = 1 | b) = g^2(b)
\]

\( \square \)
**Lemma 6**

\[ g^2(b) = \frac{\int_{a \in A_b} F^2(a) \cdot p(a) \, da}{\int_{a \in A_b} p(a) \, da} = \int_{a \in A_b} F^2(a) \cdot p(a) \, da \]

Proof

\[ g^2(b) \]

(by lemma 5) \[ = E[F^2(a) | b] \]

\[ = \int_{a \in A} F^2(a) \cdot p(a) | b) \, da \]

(by Bayes) \[ = \int_{a \in A} F^2(a) \cdot \frac{P(b | a) \cdot p(a)}{p(b)} \, da \]

but \( p(b | a) \) is 1 or 0 depending on whether \( a \in A_b \) as \( a \) determines \( b \); we may thus incorporate it into the limits of the integral, giving

\[ g^2(b) = \int_{a \in A_b} \frac{F^2(a) \cdot p(a)}{p(b)} \, da \]

however,

\[ p(b) = \int_{a \in A_b} p(a) \, da \]

which establishes the result. □

**Theorem 7** There exists at least one tooth-comb model for which the IOU function for the second iteration \( g^2 \) depends on the class-conditional distributions of the data, i.e., on the \( \mathbf{d}(X | Y = 1) \).

Proof outline

The following proof will show that it is possible to specify two different distributions on \( A \) which each have a single but different non-zero point in \( A \) belonging to the solution set under \( F_1 \) of a chosen point in \( B \). It will then be shown that these two points in \( A \) map to different points in \( C \) under \( F_2 \). This situation is illustrated in figure C.1, with the non-zero points of the two distributions shown as dots and crosses. From this it is clear that the IOU function \( g^2 \) should map the single point in \( B \) to different points in \( C \) depending on which of the two \( A \) distributions is in force.

Proof

It is possible to produce practically any (one dimensional) density on the \( A_b \)'s, providing the densities at \( A_t = 0 \) and \( A_t = 1 \) are permitted to be non-zero, by a suitable choice of
class conditional data distributions. For example, with the step-function distributions shown in fig. C.2, and assuming equal priors on \( Y \), \( p(A_i) \) will be zero at all but three points — \( A_i = 1, A_i = 2/3 \) and \( A_i = 0 \), where the density will take on a delta function. For the distribution shown, pixel data \( (x_i) \) between the values 0 and \( p \) give rise to a non-zero density at \( A_i = 1 \), values between \( q \) and \( r \) contribute to the density at \( A_i = 0 \) and values between \( p \) and \( q \) contribute to the density at \( A_i = 2/3 \) (since when \( x \) is in the range \( p \) to \( q \), \( p(x|Y = 1) \) is twice the value of \( p(x|Y = 2) \)).

However we may not take such liberties with the joint density distribution \( p(A) \) since this is subject to strong stationarity constraints and correlations via the class interactions which we are not at liberty to modify. However, we do know that if the \( d(A_i) \)s (which must be equal for all \( i \)) are zero at a point then the density at corresponding ‘corners’ in \( A \) must also be zero. Correspondingly, when the \( d(A_i) \)s have only \( r \) non-zero points then \( d(A) \) will have at most \( r^5 \) such points positioned symmetrically around the leading diagonal of \( A \).

For a particular tooth-comb model, let \( d_{[x,y,z]}(A_i) \) be a (one dimensional) density function on \( A_i \) which is zero everywhere except at points \( x, y \) and \( z \). (For example, figure C.2 shows how \( d_{[0,2/3,1]} \) may be produced). Similarly, let \( d_{[x,y,z]}^5(A) \) be the corresponding density function on \( A \). Denote by \( \{x, y, z\}^5 \subset A \) the set of points in \( A \) obtained by any 5-way combination of \( x, y \) and \( z \) — ie \( \{x, y, z\}^5 \) is the set of points in \( A \) at which \( d_{[x,y,z]}^5(A) \) is non-zero.

Now, choose \( m \in [0,1] \) and let

\[
b_m = F^1(m, m, m)
\]
Figure C.2: (a): Example data distributions; (b): resulting distribution on $A_i$.

\[ b_m = (b_m, b_m, b_m) \]  \hspace{1cm} (C.2)

\[ m = (m, m, m, m, m) \]  \hspace{1cm} (C.3)

$A_{b_m}$ be the solution set of $b_m$ under $F^1$. \hspace{1cm} (C.4)

Clearly $b_m \in A_{b_m}$ by construction.

Show that $m$ is the only point in $A_{b_m} \cap \{0, m, 1\}^5$. This is done by evaluating $F^1$ (the 5 parameter form) for each point in $\{0, m, 1\}^5$ and verifying that only $(m, m, m, m, m)$ evaluates to $b_m$. In fact due to symmetry, it is sufficient to show that for each point in $\{0, m, 1\}^3$, only $(m, m, m)$ evaluates to $b_m$ under $F^1$ (the 3 parameter form).

Next, find values $n, k$ and $l$ such that

\( (k, n, l, n, k) \neq (m, m, m, m, m) \). and

\[ F^1(k, n, l, n, k) = b_{sn} \]
In fact this can be achieved by choosing some \( n \neq m \) and setting

\[
\begin{align*}
  k &= a_1(n, n, b_m, b_m) \quad (C.5) \\
  l &= a_3(n, n, b_m). \quad (C.6)
\end{align*}
\]

where the functions \( a_1 \) and \( a_3 \) are those given in 4.29 and 4.30 of chapter 4.

This will ensure that the point \((k, n, l, n, k)\) will also lie in \( A_{b_m} \).

Now show that \((k, n, l, n, k)\) is the only point in \( A_{b_m} \cap \{1, k, n, l, 0\}^5 \). This is done by showing that \( F^1(x, y, z) \) where \( x, y, z \) are each selected from \( \{1, k, n, l, 0\} \) is equal to \( b_m \) only when \((x, y, z) \in \{(k, n, l), (n, l, n), (l, n, k)\} \) — being the three triples which 'overlap' to give \((k, n, l, n, k)\).

The above construction will ensure that the points \((m, m, m, m, m)\) and \((k, n, l, n, k)\) will both evaluate to \( b_m \) under \( F^1 \), and further that each is the only non-zero points of the distributions \( d_{[0,m,1]} \) and \( d_{[1,k,n,l,0]} \) respectively.

Finally, show that:

\[
F^2(m, m, m, m, m) \neq F^2(k, n, l, n, k). \quad (C.7)
\]

The particular tooth-comb model is the same as was used in the proof of theorem 2, ie:

\[
\begin{align*}
  D(Y) &= (1/2, 1/2) \\
  D(Y_i | Y_{i+1} = 1) &= (1/3, 2/3) \\
  D(Y_i | Y_{i+1} = 2) &= (2/3, 1/3).
\end{align*}
\]

for which the above conditions are satisfied when

\[
\begin{align*}
  m &= 2/10 \\
  n &= 3/10 \\
  k &= \frac{91811}{166645} \\
  l &= \frac{1521}{6145}
\end{align*}
\]

giving

\[
\begin{align*}
  F^1(m, m, m, m, m) &= (9/25, 9/25, 9/25) \quad (C.8) \\
  F^1(k, n, l, n, k) &= (9/25, 9/25, 9/25) \quad (C.9) \\
  F^2(m, m, m, m, m) &= \frac{361}{1145} \quad (C.10) \\
  F^2(k, n, l, n, k) &= \frac{65048080127121}{176349060932305} \quad (C.11)
\end{align*}
\]
The construction may be validated by studying the second part of the REDUCE log given in appendix D. It is easy to show that $\forall i, k, F^1(i, 1, k) = 1$ and $F^1(i, 0, k) = 0$, and also that by symmetry, $\forall i, j, k, F^1(i, j, k) = F^1(k, j, i)$; these identities have been used to shorten the list of points it is necessary to evaluate in order to verify the above conditions. The reader may, if desired, check the calculations performed by REDUCE using the formulations of $F^1$ and $F^2$ given in the proof of theorem 2.

Now consider the two densities on $\Delta = d_{[0,m,1]}$ and $d_{[1,k,n,l,0]}$. and recall lemma 6 :-

$$g^2(b) = \frac{\int_{a \in \Delta_b} F^2(a) p(a) da}{\int_{a \in \Delta_b} p(a) da}.$$  

When the previously laid out conditions are satisfied, there is only the single point $(m, m, m, m, m)$ within $\Delta_{b_m}$ at which $d_{[0,m,1]}(A)$ is non-zero, and only the single point $(k, n, l, n, k)$ within $\Delta_{b_m}$ at which $d_{[1,k,n,l,0]}(A)$ is non-zero. Thus, when either of these two densities are used the integrals in the expression in lemma 6 collapse and the densities $p(a)$ cancel.

Denote by $g^2_{[0,m,1]}$ and $g^2_{[1,k,n,l,0]}$ the IOU functions valid under distribution $d_{[0,m,1]}$ and $d_{[1,k,n,l,0]}$ respectively, we then have :-

$$g^2_{[0,m,1]}(b_m) = F^2(m, m, m, m, m)$$
$$g^2_{[1,k,n,l,0]}(b_m) = F^2(k, n, l, n, k)$$

but by C.7 the RHSs are not equal, implying that $g^2_{[0,m,1]}(b_m) \neq g^2_{[1,k,n,l,0]}(b_m)$. which shows that the correct probabilistic assignment will be different for the two distributions, ie, $g^2$ depends on the distribution $d(A)$. thence on the distribution $d(A_i)$ and thence on $d(X_i|Y_i=1)$. □
Appendix D

REDUCE program and output

% ---------------------------------------------------------------
% -------- REDUCE LOG RELATING TO THE PROOFS OF THEOREMS 2 and 7 -------
% ---------------------------------------------------------------

let n=5$ % Dimensionality of A space
let ymax = 2; % Number of classes
let middle = 3; % Center pixel in (1 2 3 4 5)
array y (n); % Classes (1,2)
array py (ymax); % Prior probabilities
array pygvnxi (n, ymax); % A posteriori probabilities P(Yi | xi)
array pygvnxi (ymax,ymax); % Transition probabilities

% Now initialise the arrays
py(1) := ppy$
py(2) := 1-ppy$
pygvnxi (1,1) := p1$
pygvnxi (2,1) := p2$
pygvnxi (3,1) := p3$
pygvnxi (4,1) := p4$
pygvnxi (5,1) := p5$
for i := 1:n do pygvnxi (i,2) := 1-pygvnxi (i,1);

% tt is the transition probability that adjacent ys are different.
pygvnxi (1,1) := ppy*tt$
pygvnxi (1,2) := ppy*(1-ppy*tt)/(1-ppy)
pygvnxi (2,2) := 1-pygvnxi (1,2)$
pygvnxi (2,1) := 1-pygvnxi (1,1)$

procedure inity;
for i := 1:n do y(i) := 1$

procedure next(i)$
% advance the array y to the next combination, but leave the 'middle'
BEGIN
if i <= n then <<
  if i=middle then i := i+1;
  if y(i) = 1 then y(i) := 2 else <<y(i) := 1; next (i+1)>>;>>;

163
procedure pyallandxall$
% Returns the prob k*p(Yall and Xall)
(for i := 2:n product pygvnxi (y(i), y(i-1)) / py(y(i))
* (for i := 1:n product pygvnxi (i, y(i)));$

procedure yprint; for i := 1:n do write (y(i))$

procedure pymidandxall (ymid)$
BEGIN SCALAR thesum;
inity();
y (middle) := ymid;
for count := 1:2^(n-1) DO <<thesum := thesum + pyallandxall();
  next(1)>>;
return thesum;
END;

procedure k;
for ymid := 1 : ymax sum pymidandxall (ymid)$

procedure pymidgvnxall (ymid)$
pymidandxall (ymid) /k();

procedure p3midgvnxall (p1,p2,p3)$
BEGIN
  n := 3;
middle := 2;
pygvnxi (1,1) := p1;
pygvnxi (2,1) := p2;
pygvnxi (3,1) := p3;
for i := 1:3 do pygvnxi (i,2) := 1-pygvnxi (i,1);
return pymidgvnxall (1);
END;

procedure p5midgvnxall (p1,p2,p3,p4,p5)$
BEGIN
  n := 5;
middle := 3;
pygvnxi (1,1) := p1;
pygvnxi (2,1) := p2;
pygvnxi (3,1) := p3;
pygvnxi (4,1) := p4;
pygvnxi (5,1) := p5;
for i := 1:5 do pygvnxi (i,2) := 1-pygvnxi (i,1);
return pymidgvnxall (1);
END;

procedure main$
BEGIN
  b2 := p3midgvnxall (a1,a2,a3);
b3 := p3midgvnxall (a2,a3,a4);
b4 := p3midgvnxall (a3,a4,a5);
END;
main();
define f1 = p3y mid gvn x all;
define f2 = p5y mid gvn x all;

% This does the proof that g^{-2} is a relation ......

b2 := f1(a1,a2,a3);
b3 := f1(a2,a3,a4);
b4 := f1(a3,a4,a5);
solve(b2-b2d,a1);
a1 := part(ws,1,2);  % ☯ Eq. (4.26)

A1 := (PPY*TT*A2 - PPY*TT*B2D - PPY*TT*A2*A3 + 3*PPY*TT*A2* 
       B2D - 2*PPY*TT*A2 + PPY*TT*A3*B2D + PPY*TT*A2*A3 - 2*PPY * 
       TT*A2*B2D - 2*PPY*TT*2 - PPY*TT*A2*A3*B2D - PPY*TT*A2*A3 + 4*PPY* 
       TT*B2D - 2*PPY*TT*A2 - PPY*TT*A3*B2D + 2*PPY*TT*A2 + PPY*TT* 
       A2*B2D + 2*PPY*TT*A2*A3 - PPY*TT*A2*B2D + 2*PPY*TT + 
       A2*B2D + 3*PPY*TT*A2*A3*B2D - PPY*TT*A2*A3 + 2*PPY*TT* 
       A2*A3 - 2*PPY*A3*B2D - A2*A3*B2D - A2*A3 + 4*PPY*B2D - 
       TT*A2*A3*B2D + TT*A2*A3 + A2*A3 + 2*PPY*TT*A2 + PPY*TT*B2D - 
       TT*B2D - PPY*TT*A2 + PPY*TT*2 - PPY*TT*A2 + PPY* 
       TT*A2*B2D - 2*PPY*TT*A2 + PPY*TT*A2*A3 - 3*PPY*TT*A2*B2D + PPY*TT*A2 - 2*
APPENDIX D. REDUCE PROGRAM AND OUTPUT

\[
\begin{align*}
PPY \ast TT \ast A3 \ast B2D + 2 \ast PPY \ast TT \ast B2D - PPY \ast A2 \ast A3 + PPY \ast A2 \ast B2D + PPY \ast A2 + PPY \ast A3 \ast B2D - 2 \ast PPY \ast B2D - 2 \ast PPY \ast TT \ast A2 \ast A3 \ast B2D + 2 \ast PPY \ast TT \ast A2 \\
A2 \ast PPY \ast A3 \ast B2D + PPY \ast TT \ast A2 + 4 \ast PPY \ast TT \ast A2 \ast A3 \ast B2D - 4 \ast PPY \ast TT \\
A2 \ast A3 + PPY \ast TT \ast A2 - PPY \ast TT \ast B2D - 2 \ast PPY \ast A2 \ast A3 \ast B2D + 2 \ast PPY \ast A2 \ast A3 + \\
PPY \ast A2 \ast B2D - 2 \ast PPY \ast A2 + PPY \ast B2D + TT \ast A2 \ast A3 \ast B2D - TT \ast A2 \ast A3 - 2 \ast TT \\
\ast A2 \ast A3 \ast B2D + 2 \ast TT \ast A2 \ast A3 + TT \ast A2 \ast B2D - TT \ast A2 + A2 \ast A3 \ast B2D - A2 \ast A3 - \\
A2 \ast B2D + A2)\end{align*}
\]

solve (b4-b4d, a5)$

a5 := part (ws, 1,2);  % ***** Equation (4.28)

\[
\begin{align*}
A5 := (PPY \ast B4D \ast TT - PPY \ast TT \ast A4 - PPY \ast B4D \ast TT \ast A3 - 2 \ast PPY \ast B4D \ast TT \ast \\
A4 + PPY \ast B4D \ast TT \ast A3 + 2 \ast PPY \ast B4D \ast TT \ast A4 - 4 \ast PPY \ast B4D \ast TT + PPY \ast \\
TT \ast A3 \ast A4 + 2 \ast PPY \ast TT \ast A4 - PPY \ast TT \ast A3 \ast A4 + 2 \ast PPY \ast TT \ast A4 + 2 \\
PPY \ast B4D \ast TT \ast A3 \ast A4 + PPY \ast B4D \ast TT \ast A4 - 3 \ast PPY \ast B4D \ast TT \ast A3 \ast A4 + 2 \\
2 \ast PPY \ast B4D \ast TT \ast A3 + 2 \ast PPY \ast B4D \ast TT \ast A4 + 2 \ast PPY \ast B4D \ast TT + PPY \ast B4D \\
A3 \ast A4 - 2 \ast PPY \ast B4D \ast A3 - 3 \ast PPY \ast B4D \ast A4 + 4 \ast PPY \ast B4D - 2 \ast PPY \ast \\
TT \ast A3 \ast A4 - PPY \ast TT \ast A4 + PPY \ast TT \ast A3 \ast A4 - 4 \ast PPY \ast TT \ast A4 + PPY \ast \\
A3 \ast A4 - PPY \ast A4 - PPY \ast B4D \ast TT \ast A3 \ast A4 - PPY \ast B4D \ast TT \ast A3 - 2 \ast PPY \ast \\
B4D \ast TT \ast A4 + PPY \ast B4D \ast A3 \ast A4 + PPY \ast B4D \ast A3 + 2 \ast PPY \ast B4D \ast A4 - 4 \ast PPY \ast \\
B4D + PPY \ast TT \ast A3 \ast A4 + PPY \ast TT \ast A3 \ast A4 + 2 \ast PPY \ast TT \ast A4 - 2 \ast PPY \ast A3 \ast A4 \\
+ 2 \ast PPY \ast A4 + B4D \ast TT \ast A3 \ast A4 - B4D \ast A3 \ast A4 + B4D - TT \ast A3 \ast A4 + A3\end{align*}
\]
\[
\begin{aligned}
&\frac{3}{2} - \frac{3}{2} \frac{2}{2} - \frac{3}{2} \frac{2}{2} \\
&\frac{A4 - A4}{(PPY *B4D*TT - PPY *B4D*TT - PPY *TT *A4 + PPY *TT)} \\
&\frac{2}{2} - \frac{2}{2} - \frac{2}{2} \\
&\frac{A4 - PPY *B4D*TT *A3 - 2*PPY *B4D*TT *A4 + 2*PPY *} \\
&\frac{2}{2} - \frac{2}{2} - \frac{2}{2} \\
&B4D*TT*A3 + 3*PPY *B4D*TT*A4 - 2*PPY *B4D*TT - PPY * \\
&\frac{2}{2} - \frac{2}{2} - \frac{2}{2} \\
&B4D*A3 - PPY *B4D*A4 + 2*PPY *B4D + PPY *TT *A3*A4 + \\
&\frac{2}{2} - \frac{2}{2} - \frac{2}{2} \\
&2*PPY *TT *A4 - 2*PPY *TT*A3*A4 - PPY *TT*A4 + PPY \\
&\frac{2}{2} - \frac{2}{2} - \frac{2}{2} \\
&A3*A4 - PPY *A4 + 2*PPY*B4D*TT *A3*A4 + PPY*B4D*TT \\
&A4 - 4*PPY*B4D*TT*A3*A4 + PPY*B4D*TT + 2*PPY*B4D*A3 \\
&\frac{2}{2} - \frac{2}{2} - \frac{2}{2} \\
&A4 - PPY*B4D*A4 - PPY*B4D - 2*PPY*TT *A3*A4 - PPY* \\
&\frac{2}{2} - \frac{2}{2} - \frac{2}{2} \\
&TT *A4 + 4*PPY*TT*A3*A4 - PPY*TT*A4 - 2*PPY*A3*A4 + \\
&\frac{2}{2} - \frac{2}{2} - \frac{2}{2} \\
&2*PPY*A4 - B4D*TT *A3*A4 + 2*B4D*TT*A3*A4 - B4D*TT \\
&\frac{2}{2} - \frac{2}{2} - \frac{2}{2} \\
&A4 - B4D*A3*A4 + B4D*A4 + TT *A3*A4 - 2*TT*A3*A4 + \\
&TT*A4 + A3*A4 - A4) \\
\end{aligned}
\]

\text{solve (b3-b3d, a3)$\$
\]

\text{a3 := part (ws,1,2); % ***** Equation (4.27)}

\[
\begin{aligned}
&4 \quad 2 \quad 3 \quad 2 \quad 3 \quad 2 \quad 3 \quad 3 \\
&A3 := (B3D*(PPY *TT - PPY *TT *A2 - PPY *TT *A4 + PPY *TT*A2 + PPY *) \\
&3 \quad 2 \quad 2 \quad 2 \\
&TT*A4 - 4*PPY *TT + PPY *TT *A2*A4 - 2*PPY *TT*A2*A4 + 2* \\
&2 \quad 2 \quad 2 \quad 2 \\
&PPY *TT*A2 + 2*PPY *TT*A4 + 2*PPY *TT + PPY *A2*A4 - 2* \\
&2 \quad 2 \quad 2 \\
&PPY *A2 - 2*PPY *A4 + 4*PPY - PPY*TT*A2 - PPY*TT*A4 + \\
&4 \quad 2 \quad 3 \quad 2 \\
&PPY*A2 + PPY*A4 - 4*PPY + 1))/((PPY *TT + 2*PPY *TT *B3D
\end{aligned}
\]
APPENDIX D. REDUCE PROGRAM AND OUTPUT

\[
\begin{align*}
&3\ 2\ 3\ 2\ 3\ 2\ 3\ 3 \\
&- \text{PPY} * \text{TT} * \text{A2} - \text{PPY} * \text{TT} * \text{A4} - 2*\text{PPY} * \text{TT} - 2*\text{PPY} * \text{TT} * \text{B3D} + \text{PPY} * \text{TT} \\
&3\ 3\ 2\ 2\ 2\ 2\ 2\ 2 \\
&* \text{A2} + \text{PPY} * \text{TT} * \text{A4} - 2*\text{PPY} * \text{TT} - 2*\text{PPY} * \text{TT} * \text{B3D} * \text{A2} - 2*\text{PPY} * \text{TT} * \text{B3D} * \\
&2\ 2\ 2\ 2\ 2\ 2\ 2\ 2 \\
&\text{A4} - \text{PPY} * \text{TT} * \text{B3D} + \text{PPY} * \text{TT} * \text{A2} * \text{A4} + 2*\text{PPY} * \text{TT} * \text{A2} + 2*\text{PPY} * \text{TT} * \text{A4} \\
&2\ 2\ 2\ 2\ 2 \\
+ \text{PPY} * \text{TT} + 3*\text{PPY} * \text{TT} * \text{B3D} * \text{A2} + 3*\text{PPY} * \text{TT} * \text{B3D} * \text{A4} - 2*\text{PPY} * \text{TT} * \text{B3D} \\
&2\ 2\ 2 \\
- 2*\text{PPY} * \text{TT} * \text{A2} * \text{A4} - \text{PPY} * \text{TT} * \text{A2} - \text{PPY} * \text{TT} * \text{A4} + 4*\text{PPY} * \text{TT} - \text{PPY} * \text{B3D} \\
&2\ 2\ 2 \\
* \text{A2} - \text{PPY} * \text{B3D} * \text{A4} + 3*\text{PPY} * \text{B3D} + \text{PPY} * \text{A2} * \text{A4} - \text{PPY} * \text{A2} - \text{PPY} * \text{A4} + \\
&2\ 2\ 2 \\
\text{PPY} + 2*\text{PPY} * \text{TT} * \text{B3D} * \text{A2} * \text{A4} + \text{PPY} * \text{TT} * \text{B3D} * \text{A2} + \text{PPY} * \text{TT} * \text{B3D} * \text{A4} - 2* \\
&2 \\
\text{PPY} * \text{TT} * \text{A2} * \text{A4} - \text{PPY} * \text{TT} * \text{A2} - \text{PPY} * \text{TT} * \text{A4} + 4*\text{PPY} * \text{TT} * \text{B3D} * \text{A2} * \text{A4} + 2* \\
\text{PPY} * \text{TT} * \text{B3D} + 4*\text{PPY} * \text{TT} * \text{A2} * \text{A4} - \text{PPY} * \text{TT} * \text{A2} - \text{PPY} * \text{TT} * \text{A4} - 2*\text{PPY} * \text{TT} + 2 \\
* \text{PPY} * \text{B3D} * \text{A2} * \text{A4} - \text{PPY} * \text{B3D} * \text{A2} - \text{PPY} * \text{B3D} * \text{A4} - 2*\text{PPY} * \text{B3D} - 2*\text{PPY} * \text{A2} * \text{A4} \\
&2 \\
+ 2*\text{PPY} * \text{A2} + 2*\text{PPY} * \text{A4} - 2*\text{PPY} - \text{TT} * \text{B3D} * \text{A2} * \text{A4} + \text{TT} * \text{A2} * \text{A4} + 2*\text{TT} * \\
\text{B3D} * \text{A2} * \text{A4} - \text{TT} * \text{B3D} * \text{A2} - \text{TT} * \text{B3D} * \text{A4} - 2*\text{TT} * \text{A2} * \text{A4} + \text{TT} * \text{A2} + \text{TT} * \text{A4} - \\
\text{B3D} * \text{A2} * \text{A4} + \text{B3D} * \text{A2} + \text{B3D} * \text{A4} + \text{A2} * \text{A4} - \text{A2} - \text{A4} + 1) \\
\end{align*}
\]

% The expressions for \(a_1\), \(a_3\), and \(a_5\) in terms of \(a_2\) and \(a_4\) (plus the
% \(b\) values) are now available, however they are not printed here because
% the expressions for \(a_1\) and \(a_5\) are very long (3 pages each) in their
% fully expanded form.

% We now instantiate the priors, transition probabilities
% and the point in \(B\) space in order to obtain an expression \(f_2\) in terms
% of \(a_2\) and \(a_4\) alone. The instantiations are necessary only because
% REDUCE cannot cope with the size of the fully general expression.

let ppy=1/2$
let tt=2/3$
let b2d=1/2$
let b3d=1/2$
let b4d=1/2$
f2 (a1, a2, a3, a4, a5);
% Notice that the expression for f2 is not a constant, % but depends on a2 and a4. There is thus a continuum of % points in C space which relate to the SINGLE point (.5, .5, .5) % in B space.

% COUNTER EXAMPLE FOR THEOREM 2
% ----------------------------------------
% % Following are the calculations relating to the particular counter example % used in the proof of theorem 2.
% % The first point - "a"

let a2a=705/1000$
let a4a=705/1000$
let a2=a2a$
let a4=a4a$

a1a := a1;
17648663
A1A := ---------
17663800

a3a := a3;
116281
APPENDIX D. REDUCE PROGRAM AND OUTPUT

A3A := ---------
        183362

a5a := a5;
        17648663
A5A := ---------
        17663800

f1 (a1a, a2a, a3a); % ***** Expression (4.19)
  1 / 2

f1 (a2a, a3a, a4a); % ***** Expression (4.19)
  1 / 2

f1 (a3a, a4a, a5a); % ***** Expression (4.19)
  1 / 2

f2 (a1a, a2a, a3a, a4a, a5a); % ***** Expression (4.21)

8279810308732369
------------------------
13661490665376338

% Now the second point - "b"

let a2b=295/1000$
let a4b=295/1000$
let a2=a2b$
let a4=a4b$

a1b:=a1;
        15137
A1B := ---------
        17663800

a3b:=a3;
        67081
A3B := ---------
        183362

a5b:=a5;
        15137
A5B := ---------
        17663800

f1 (a1b, a2b, a3b); % ***** Expression (4.20)
  1 / 2

f1 (a2b, a3b, a4b); % ***** Expression (4.20)
\[ f_1(a_3 b, a_4 b, a_5 b); \quad \text{****** Expression (4.20)} \]
\[ f_2(a_1 b, a_2 b, a_3 b, a_4 b, a_5 b); \quad \text{****** Expression (4.22)} \]

\[ 5381680356643969 \]
\[ \text{----------------------------------} \]
\[ 13661490665376338 \]

\[
\% \quad \text{COUNTER EXAMPLE FOR THEOREM 7} \\
\% \quad \text{----------------------------------} \\
\% \quad \text{The following relates to the proof of theorem 7.} \\
\% \quad \text{The constants referred to as k, l, m, n in the text are written as} \\
\% \quad \text{z_k, z_l, z_m, z_n here.} \\
\]

let zm=2/10;
let zn=3/10;
let bzm=f_1(zm,zm,zm); \quad \text{****** Equation (C.1)}
let b2d=bzm; \quad \text{****** Equation (C.2)}
let b3d=bzm; \quad \text{" " "}
let b4d=bzm; \quad \text{" " "}

f_1(1,zm,1);
\[ 1 / 17 \]

f_1(zm,zm,zm); \quad \% \text{Verify that this value is unique in amongst these 6} \\
\% \text{****** c.f Equation (C.8)}
\[ 9 / 25 \]
f_1(1,zm,zm);
\[ 3 / 19 \]
f_1(zm,zm,0);
\[ 3 / 7 \]
f_1(1,zm,0);
\[ 1 / 5 \]
f_1(0,zm,0);
\[ 1 / 2 \]
let a2=zn;
let a4=zn;
let zk=a1; % ***** Equation (C.5)
let zl=a3; % ***** Equation (C.6)

% From the list below, verify that only the values marked by "*" are
% equal to the value of f1(zm, zm, zm) above. Notice that the marked
% sub patterns will each be found in the pattern (zk, zn, zl, zn, zk).

f1(1,zn,1);
3 / 31

f1(1,zn,zn);
51 / 233

f1(1,zn,zk);
34497 / 206801

f1(1,zn,zl);
32307 / 139631

f1(1,zn,0);
3 / 10

f1(1,zl,1);
1521 / 20017

f1(1,zl,zn);
117 / 661

f1(1,zl,zk);
122429853 / 919163549

f1(1,zl,zl);
16379649 / 87274817

f1(1,zl,0);
1521 / 6145

f1(1,zk,1);
\[ f_1(z_k, zn); \]
\[ f_1(z_k, zk); \]
\[ f_1(z_k, z_l); \]
\[ f_1(z_k, 0); \]
\[ f_1(z_n, zn, zn); \]
\[ f_1(z_n, zn, zk); \]
\[ f_1(z_n, zn, z_l); \]
\[ f_1(z_n, z_l, zn); \]
\[ f_1(z_n, z_l, z_k); \]
\[ f_1(z_n, z_k, zn); \]
\[ f_1(z_n, z_k, zk); \]

\[ f_1(z_n, zn, z_l); \]
\[ f_1(z_n, zn, 0); \]
\[ f_1(z_n, z_l, z_k); \]
\[ f_1(z_n, z_l, 0); \]
\[ f_1(z_n, z_k, z_l); \]
\[ f_1(z_n, z_k, 0); \]

\[ f_1(z_n, zn, zk); \]
\[ f_1(z_n, zn, z_k); \]
\[ f_1(z_n, z_l, zn); \]
\[ f_1(z_n, z_l, z_k); \]
\[ f_1(z_n, z_k, zn); \]
\[ f_1(z_n, z_k, z_l); \]
7390142823 / 12320277175
f1(zn,zk,zl);
988712659 / 1427407175
f1(zn,zk,0);
91811 / 120424
f1(zk,zn,zn);
586449 / 1706425
f1(zk,zn,zk);
2776767021 / 10198934125
f1(zk,zn,zl); % * (zk, zn, zl, --, --) See Equation (C.9)
9 / 25
f1(zk,zn,0);
34497 / 77573
f1(zk,zl,1);
122429853 / 919163549
f1(zk,zl,zn);
9417681 / 32851025
f1(zk,zl,zk);
9854746157529 / 44174846846425
f1(zk,zl,zl);
31941 / 105925
f1(zk,zl,0);
122429853 / 321613277
f1(zk,zk,zn);
7390142823 / 12320277175
f1(zk,zk,zk);
594854766251739 / 1150285219312475
f1(zk,zk,zl);
  1928031 / 3125375

f1(zk,zk,0);
  7390142823 / 10613692207

f1(zl,zn,zn);
  549219 / 1246825

f1(zl,zn,zl);
  347914083 / 759266975

f1(zl,zn,0);
  32307 / 59138

f1(zl,zl,zn);
  125973 / 3345125

f1(zl,zl,zl);
  176392440081 / 448133619025

f1(zl,zk,zn);
  988712659 / 1427407175

f1(zl,zk,zl);
  1064744624771 / 15045257910475

f1(zl,zk,0);
  988712659 / 1275551381

f1(0,zn,zn);
  102 / 193

f1(0,zn,0);
  12 / 19

f1(0,zl,zn);
117 / 253
fl(0,zl,0);

1521 / 2677
fl(0,zk,zn);

91811 / 120424
fl(0,zk,0);

183622 / 221039

% Verify that f2(zm,zm,zm,zm,zm) differs from f2(zk,zn,zl,zn,zk).

f2(zm,zm,zm,zm,zm); % ***** Equation (C.10)
361 / 1145

f2(zk,zn,zl,zn,zk); % ***** Equation (C.11)
65048080127121 / 176349060932305

% -------------------------- End of Reduce log --------------------------
Bibliography


BIBLIOGRAPHY


BIBLIOGRAPHY


[Geman87c] D Geman, S Geman, C Graffigne and et al, “Boundary detection by constrained optimisation”, University of Massachusetts,


BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


