Machine Learning Techniques for Microarray Analysis

Simon David Rogers

A dissertation submitted to the University of Bristol
in accordance with the requirements of the degree of
Doctor of Philosophy in the Faculty of Engineering
Department of Engineering Mathematics

July 2004

Abstract

In recent years, the development of high throughput DNA microarray technology has enabled biologists to study cellular activities through measurement of gene expression levels. Such measurements are proving invaluable in understanding the development of diseases such as cancer and in piecing together the systems of regulation that exist between genes. However, the analysis of such data is non-trivial as the number of genes measured in one sample is typically far greater than the number of samples. When presented with such data, many standard data analysis and machine learning techniques are either inappropriate or become computationally infeasible. Therefore, the development of novel algorithms and techniques that can efficiently extract useful information from such datasets is crucial in realising the full potential of microarray technology. The aim of this thesis is to investigate the ability of existing techniques and invent new methods to analyse these datasets. Firstly, we will introduce a novel generative, probabilistic model that is able to decompose microarray data into a set of latent processes. This model overcomes some of the unrealistic assumptions implicitly made by standard unsupervised learning techniques such as clustering. Investigations using several real-life microarray datasets show that the algorithm is able to find meaningful structure in the data and is able to uncover processes of real biological significance. Secondly, we will develop a methodology that enables experimentalists to predict how many microarrays are required to obtain a certain generalisation error when using classifiers to classify between various cellular states. This provides a rigorous procedure for determining whether it is worth taking more measurements to improve performance. It also provides an insight into the difficulty of the classification problem being considered and can be used to compare the efficiencies of different experimental measurement techniques. Finally, we investigate the problem of inferring regulatory networks from microarray data and assess the potential of an existing machine learning technique in this area. This final section corresponds to work in progress and points to many interesting directions for future research.

Acknowledgements

Firstly, I am indebted to my supervisor Colin Campbell for his encouragement, direction and patience over the last three years. I have also benefited from collaborations with various others - Dan Williamson at the institute for Cancer Research, Sayan Mukherjee and friends at MIT and Mark Girolami at the Bioinformatics Research Centre in the University of Glasgow. I'd like to particularly thank the latter, and everybody else at the BRC for making my two visits to the centre so enjoyable and productive.

On a (slightly) less academic note, Chris and Tom in R2A, Richie, Luke and John provided some useful discussions but mainly useful distractions.

Finally, I'd like to thank my family for all the obvious reasons - cake, trips to Edgar street etc.

And, of course, Helen.

Author's Declaration

I declare that the work in this dissertation was carried out in accordance with the regulations of the University of Bristol. The work is original except where indicated by special reference in the text and no part of the dissertation has been submitted for any other degree.

Any views expressed in the dissertation are those of the author and in no way represent those of the University of Bristol.

The dissertation has not been presented to any other University for examination either in the United Kingdom or overseas.

Signed: ....

Date: ....

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 cDNA Microarrays

Proteins are essential components of all living organisms. As well as making up the major structural component of cells they also act as catalysts, initiating a wide range of reactions and processes. The ability to measure the presence and abundance of proteins in a particular cell under certain conditions is a central goal of molecular biology. Such knowledge would provide a very detailed description of the behaviour of organisms at a cellular level and would in turn lead to a better understanding of various diseases and possible treatments. Microarrays allow us to take such measurements. To understand how, we must first briefly review the process of protein synthesis.

The so called central dogma of molecular biology was first suggesed by Frances Crick in the 1950s. Basically it states the following: *DNA makes RNA makes protein.* It is now known that although the complete picture is rather more complicated, this original assumption is generally correct and is certainly sufficient for this introduction.

This synthesis process involves three distinct steps - translation, splicing and transcription. We will now briefly describe these three processes in turn

- **Transcription** During this stage, the double stranded DNA is unwound and the information in one of the strands is copied into complementary preliminary messenger RNA.

- **Splicing** The splicing stage involves extracting the coding regions from the gene (known as *exons*) from the non-coding regions (*introns*). The fact that the coding regions of the gene are non-continuous creates a demanding problem in the discovery of the different genes in the genome.
- **Translation** The final stage involves translating the now continuous stretch of genomic material into a string of amino acids to produce the protein. Each set of three consecutive bases (a *codon*) codes one particular amino acid. It is worth noting that as there are only 20 amino acids and 4^3 unique codons, this coding process is rather redundant.

The quantity of preliminary messenger RNA that is being transcribed by a particular gene is broadly referred to as the expression level of that gene.

There are many factors that can regulate the abundance of a particular protein in a given cell. For example, in one case the respective gene may not be expressed whilst in another case it may be intensively expressed. Also, the speed of production varies for different proteins, as does the half-life of the mRNA and the protein. This suggests that the relationship between the quantity of mRNA and the abundance of that particular protein is non-trivial. However it is widely accepted that measuring the level of mRNA gives us a reasonable insight into the protein abundances. This measurement can be performed using *microarrays*, the analysis of which is the subject of this thesis.

There are two main forms of microarray in general use; Spotted microarrays and Oligonucleotide microarrays, both of which rely on the same binding property of DNA. DNA and RNA are examples of nucleic acids, one characteristic of which is their tendency to form double stranded molecules through complementary base pairing. The best known example of this is the double helix structure of human DNA molecules first suggested by Watson and Crick in the early 1950s [78]. This tendency of nucleic acids to form double stranded molecules is known as hybridisation and plays an important role in the measurement of mRNA abundance. Let us consider one specific gene and its mRNA product. Given a sample of this mRNA, it is possible to *reverse transcribe* it to single stranded complimentary DNA (cDNA) which would hybridise to a single strand of the gene's original DNA. It is this hybridisation that underlies the operation of microarrays.

A spotted microarray consists of a small slide onto which the DNA sequences of the genes to be measured are printed in pre-defined location (spots). Each spot contains many copies of the sequence of one gene. mRNA is then extracted from the cell of interest and a separate control cell. Reverse transcription is used to transform the mRNA molecules into cDNA molecules. The two samples are then labelled with fluorescent dyes (often visualised as red and green although not actually visible to the naked eye) and are then washed over the slide and left to hybridise for a set period of time. Once this time has elapsed, the slides are rinsed and are ready to be analysed. The dyes are excited individually by a laser and for each spot, a fluorescence reading can be made for both *colours*. The ratio of these readings provides us with a relative level of expression for our sample with respect to the control. For example, if for a particular gene there is much more mRNA in our sample relative to the control, the dye corresponding to our sample will fluoresce much more than the dye for the control and we will have a high ratio. These ratios are normally logged (base 2) to preserve the symmetry between over and under expression.

Oligonucleotide arrays use a slightly more complicated procedure, but do not need a separate control sample and hence provide absolute rather than relative expression values. For each gene that is being measured, 20 small sections of the gene's DNA are printed at various locations around the array (these are referred to as perfect match (PM) probes). Next to each of these, the same sequence is printed but with the middle base switched (mismatch (MM) probes). As with the spotted arrays, our sample is labelled with a fluorescent dye and washed over the array. Various algorithms exist to combine these 40 probe values into one expression value (for example [1]).

The problem of transforming an image file showing all of the dots and their fluorescence values, to a dataset for further analysis is in itself an interesting area for research. Recently, various machine learning techniques have been used in this area. For example, Lawrence et L [41] use a Bayesian approach to improve the procedure of extracting expression values from spots in spotted arrays. Milo et al [49] suggest a probabilistic approach to extracting expression values from Oligonucleotide arrays. Although it would be very interesting in the future to augment such approaches with downstream analysis, this thesis will focus just on the downstream analysis and take the data as given.

1.2 Aims of Microarray Analysis

At this point it is worthwhile being a bit more specific about the aims of microarray analysis. We will introduce several such aims below - this is by no means an exhaustive list.

1.2.1 Identification of differentially expressed genes

One of the original and main uses for microarrays is in discovering differentially expressed genes. An obvious application is the identification of bio-markers corresponding to various disease states. For example, we may have gene expression measurements for some healthy prostate cells and similar measurements from tumour cells. Genes that have consistent differential expression across these two groups may be ultimately useful for diagnosis or help find possible drug targets. In a more short term outlook, they will likely be interesting targets for future biological research and their identification may help to unravel the complex biological processes responsible for diseases such as cancer.

It is also interesting to find genes whose expression varies as the result of cellular processes (e.g. cell division) or as a result of exposure to some external stimulus. For example, in the seminal paper of Spellman et al [71], some 800 yeast genes that appear to be involved in the cell cycle are identified through microarray analysis. Gasch et al [22] investigated yeast genes whose expression varied when the sample was exposed to various sudden changes in condition such as heat shock.

1.2.2 Cellular Classification

One area of microarray analysis that has certainly drawn a lot of attention from the machine learning community is cellular classification. Here we are interested in whether it is possible to use the gene expression measurements to classify samples into groups such as healthy/tumour or relapse/non-relapse. Apart from the obvious diagnostic tools that could be developed from such classifiers (impractical presently due to the high cost of data generation) they can also be used to identify differentially expressed genes. For example, Li et al [44] used a Bayesian classifier that enforced sparsity over features (i.e. genes) and found subsets of genes that gave very good classification performance and hence were interesting candidates for future investigation.

1.2.3 Disease subtype identification

Unsupervised techniques such as clustering have been used to identify tentative sub-types in groups of samples that were previously thought of as being homogeneous. For example, Alizadeh et al [4] identified previously unknown sub-types of lymphoma by using hierarchical clustering on gene expression data. Bhattacharjee et al [6] performed a similar analysis of lung carcinomas. Such sub-types, that may be hard to distinguish in a traditional manner, can correspond to very different clinical outcomes. For example, these may be different reactions to various forms of treatments and hugely different survival rates (for example, see [4], figure 5 and [6], figure 4).

1.2.4 Inference of gene regulatory interactions

A possible use of gene expression data that has recently started to receive more interest is the reverse engineering of so called genetic regulatory networks. These networks describe the regulatory interactions between various genes. In simplistic terms, the protein produced by one gene can bind to the promoter region of another gene, causing a change in its transcription. Learning the structure of such networks is one of the great challenges of functional genomics. Knowledge of the relationships between genes is crucial in finding possible targets for new drugs. We will discuss the task of inferring regulatory networks in more detail later in chapter 6.

1.3 Standard Machine Learning Approaches

In this section, we will look at some of the machine learning techniques that have been used previously in the analysis of microarray data. These approaches can be placed into two categories - supervised techniques and unsupervised techniques. We will deal with these categories separately below. Some of this work has previously been published in [65].

1.3.1 Unsupervised Techniques

Clustering algorithms are possibly the most widely used machine learning algorithms for the analysis of microarray data. Such algorithms find a partitioning of the data samples that is in some way optimal. A seminal paper in this area was published by Eisen et al [16] where the use of hierarchical clustering for discovering structure in microarray data was investigated. The software made available with this paper and the speed and intuitive nature of this technique have made it very popular. The technique proceeds in a recursive manner, at each stage combining the two data points (genes or cell samples depending on whether we are interested in finding similarly expressed genes or samples) that are the closest with respect to some distance metric (for example Euclidean distance or Pearson correlation). The combined points are then considered to be an individual point and the process is repeated. There are several methods for combining points, see [16] for details. We will give more information on hierchichal clustering in chapter 4.

Other unsupervised techniques have been used in this area with limited success. We will review some of them in chapter 4.

1.3.2 Supervised Techniques

The main supervised learning technique that has been used is classification. Typically microarray datasets have a large number of features and a small number of examples. Thus, for example, the prostate cancer dataset of Singh et al [69] consists of 50 normal and 52 tumour samples with 12600 features each. Given that the data can therefore be viewed as a sparse set of points in a high-dimensional space (corresponding to a large number of features), it is not surprising that binary class datasets of this type are usually linearly separable: a hyperplane can readily be found which correctly separates both classes. This suggests a preference for simpler algorithms. For example, the perceptron algorithm and its variants can efficiently handle linearly separable problems and can readily be used with these datasets. However, other factors are important. For example, in addition to assigning a class label to a new test point, it would be worth stating a confidence measure too. For some classifiers a confidence or probability measure is given and this will be an advantage in practical applications. A large number of classification algorithms could potentially be used, ranging from discriminant methods, to Gaussian Processes and classification trees. In this section we will only describe two popular choices for illustration: k-nearest neighbours and perceptrons. Support vector machines (SVM's) are also a popular choice for classifying microarray data and will be used extensively in this thesis. Therefore, we overlook them here and give a detailed introduction in the next chapter.

K-nearest neighbour classifiers

One of the simplest classifiers available is k-Nearest Neighbours (kNN). This classifier requires no training and the class of a new point is simply predicted to be the most common class among the k nearest neighbours. In its simplest form and for binary classification with $y_i = \pm 1$, the decision function is:

$$y = sign\left(\sum_{i|x_i \in \mathbf{K}} y_i\right) \tag{1.1}$$

where $\mathbf{K}$ is the set of neighbours closest to the new point, $\mathbf{x}$. This method can easily be extended to multi-class classification with the class of a new point determined by the consensus of its neighbours.

The set of nearest neighbours is determined by a distance metric that is usually the Euclidean distance in input space. Generally it is best to scale the influence of each neighbour depending on distance from the new point. This is easily accomplished by multiplying the class label of each neighbour by a weighting term, e.g. the reciprocal of the distance between the points. k-nearest neighbours was introduced in 1951 and since then there have been many extensions and variations proposed. One variation is the probabilistic nearest neighbour model (see e.g. [29]): not only does this assign a probability to the predicted label but it also automatically calculates the value of k resulting in a fully autonomous classifier.

Perceptron classifiers

The perceptron is a simple and effective classifier which can readily handle linearly separable datasets (Fig. 1.1). Indeed, using the idea of kernel substitution that will be introduced in chapter 2, it can also handle non-linearly separable datasets. The *perceptron convergence theorem* states that the perceptron algorithm will converge on a solution after a finite number of passes through the dataset, provided a solution exists. If $\mathbf{x}_i$ are the feature vectors and z are binary-valued outputs then the decision function is:

$$z = sign(\mathbf{w} \cdot \mathbf{x}_i + b). \tag{1.2}$$

and the learning task is to find a weight vector $\mathbf{w}$ and bias b such that training feature vectors $\mathbf{x}_i$ map correctly to the corresponding labels y_i. This is achieved using an iterative training process with multiple passes through the data using the weight changes:

$$\Delta \mathbf{w} = y_i \mathbf{x}_i H \left[y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b \right) \right] \tag{1.3}$$

where $H(\theta)$ is the Heaviside Step function, we use $y_i = \pm 1$ and the bias b can be found by adding an extra component (say 0) to the feature vectors fixed at $x_0 = +1$ and with the bias given as $b = w_0$. The weight vector is therefore only corrected by an additive factor $y_i \mathbf{x}_i$ if the decision function gives the wrong label z on presentation of the ith feature vector.

From (1.2) we see that the separatrix in the decision function is the linear hyperplane

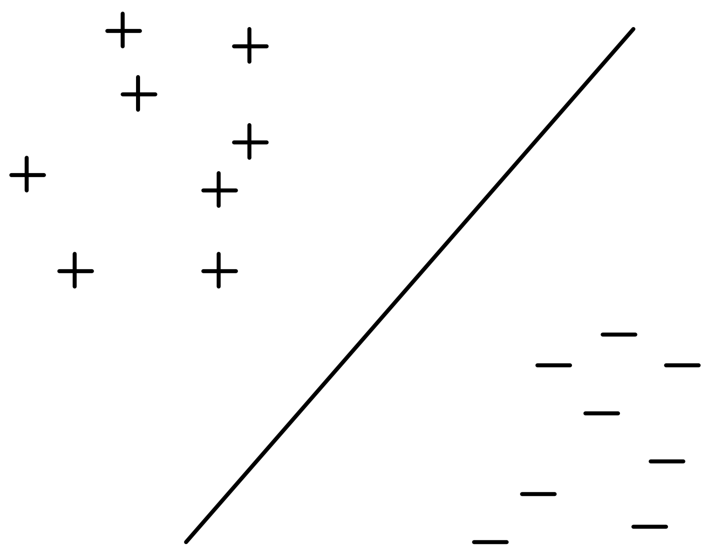


Fig. 1.1: In input space a linear classifier finds a hyperplane separating the two classes

$\mathbf{w} \cdot \mathbf{x} + b = 0$ depicted in Figure 1.1. In an alternative view (the *geometric dual*) hyperplanes become points and points become hyperplanes (Figure 1.2). This alternative representation gives insights about the solution found for both the perceptron and SVMs. In this dual representation we have a *version space* inside which each point represents a possible hyperplane correctly separating the datapoints in the original input space (Figure 1.1). The boundaries of version space derive from the data-points. The iterations of the perceptron algorithm can be viewed as a trajectory which terminates inside version space starting from outside if the initial weights do not classify the training data correctly. Unfortunately, this indicates that the solution can be biased since it can depend on the starting point and the order of presentation of the patterns in (1.3). In addition, there is no simple way of implementing a confidence measure for class prediction with the perceptron. In the next chapter, we will describe SVM classifiers which have a unique solution independent of the order of presentation - in addition a confidence can be assigned to the class label.

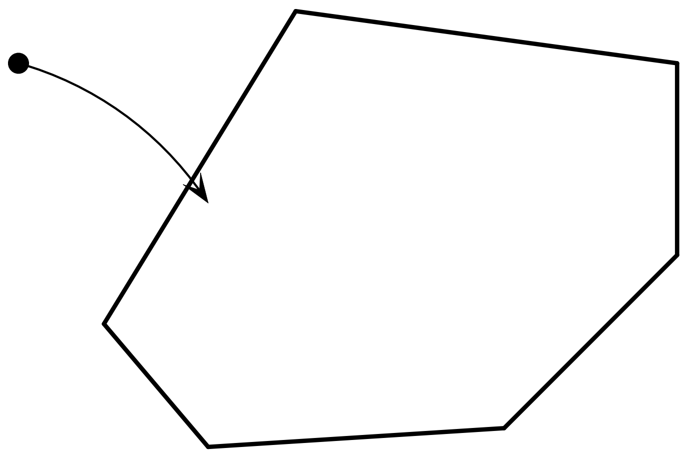


Fig. 1.2: For the dual representation of Figure 1.1 datapoints become hyperplanes and hyperplanes correctly separating the data become points inside the *version space* (pictured, version space can be open in general). Thus the iterations of the perceptron algorithm corresponds to a trajectory into version space (shown) where the algorithm terminates with zero training error.

1.4 Thesis Outline

In chapter 2 we introduce the Support Vector Machine, a machine learning algorithm that is used extensively in chapter 5. We also present results from a study of relapse versus non-relapse classification for Wilms tumours.

Chapter 3 gives some background in the area of probabilistic models that is required for chapter 4.

Chapters 4 and 5 contain the main contributions of this thesis. Chapter 4 introduces a novel algorithm for the analysis of microarray data - Latent Process Decomposition. This work was performed in collaboration with Mark Girolami and Rainer Breitling at the University of Glasgow and has been accepted for publication [64] as well as appearing as a poster presentation at RECOMB 2004 [63].

Chapter 5 introduces a methodology for inferring relationships between classification performance and sample size when using Support Vector Machines to classify tissue samples. This work was performed in collaboration with Sayan Mukherjee et al (MIT) and has previously been published in [51].

Chapter 6 describes work done in the Bioinformatics Research Centre at the University of Glasgow on the inference of genetic regulatory networks. It investigates the potential of two machine learning algorithms in this application. Although results are presented, this essentially represents work in progress and opens up many interesting questions and directions for future research.

Finally, we present a conclusion and some future research directions.

1.5 Publications

1. Class Prediction with Microarray Datasets. S. Rogers, R. Williams and C. Campbell. Chapter in *Bioinformatics with computational intelligence paradigms.* Editor Udo Seiffert. Springer-Verlag 2004.

2. Estimating Dataset Size Requirements for Classifying DNA microarray Data. S. Mukherjee, P. Tamayo, S. Rogers, R. Rifkin, A. Engel, C. Campbell, T. Golub and J. Mesirov. *Journal of Computational Biology* 10(2) 2003 pp 119-142.

3. Expression Profiling Targeting Chromosomes for Tumor Classification and Prediction of Clinical Behavior. Y. Lu, D. Williamson, R. Wang, B. Summersgill, S. Rodriguez, S. Rogers, K. Pritchard-Jones, C. Campbell and J. Shipley. *Genes Chromosomes and Cancer* 38 2003 pp 207-214

4. Prognostic Classification of Relapsing Favorable Histology Wilms Tumor Using cDNA Microarray Expression Profiling and Support Vector Machines. R. Williams, S. Hing, B. Greer, C. Whiteford, J. Wei, R. Natrajan, A. Kelsey, S. Rogers, C. Campbell, K. Pritchard-Jones and J. Khan. *Genes Chromosomes and Cancer* 41 2004 65-79.

5. The Latent Process Decomposition of DNA Microarray data. S. Rogers, M. Girolami, C. Campbell and R. Breitling. *IEEE Transactions on Bioinformatics and Comutational Biology* Special Issue - Machine Learning in Bioinformatics. In press.

6. Sparse Bayesian Approach for inferring Genetic Regulatory Networks. S. Rogers and M. Girolami. *Bioinformatics.* In preparation.

7. Query Learning with Gaussian Processes. S. Rogers and C. Campbell. *Proceedings of the 2003 UK Workshop on Computational Intelligence.* Edited by J. Rossiter and T. Martin. pp 45-52.

8. A Latent Process Decomposition Model for Interpreting cDNA Microarray Datasets. S. Rogers, M. Girolami and C. Campbell. *Currents in Computational Molecular*

Biology 2004. Eighth Annual International Conference on Research in Computational Molecular Biology (RECOMB 2004). Poster presentation.

Chapter 2

Support Vector Classification

2.1 Introduction

Kernel methods is a relatively new area in machine learning with its origin firmly rooted in statistical learning theory. Support Vector Machines are perhaps the most well known kernel method and are the approach we will mainly be interested in here. Support Vector Machines (SVM's) have been applied relatively successfully to a variety of (mainly classification) learning tasks (for many examples, see [2]). In bioinformatics, they have proved very useful in the classification of gene expression datasets due to the fact that their complexity and generalisation ability tends to be limited by the number of training samples and not their dimensionality (the largest gene expression datasets currently available have the order of hundreds of samples but tens of thousands of features).

Consider the following learning process. We have an empirical data set

$$(x_1, y_1), \ldots, (x_m, y_m) \in \mathcal{X} \times \{\pm 1\} \tag{2.1}$$

where $\mathcal{X}$ is some space from which the x_i are assumed to have been sampled and y_i is a label denoting which *class* (+1 or -1) x_i belongs to. For example, each of our examples x_i may correspond to the expression levels for a set of genes in some particular sample. The task of classification is to create some rule or hypothesis from this *training* data so that, given un-labelled patterns we can make some estimate or inference as to their label. In other words, if an un-labelled pattern x is in some way similar to one of our training patterns x_i then it is likely that the unknown label y of x will be similar to the known label y_i of x_i. In order to do this, we need some notion of similarity in $\mathcal{X}$ - we require some function that, given two patterns in $\mathcal{X}$, x_i and x_j returns a real value that is a measure of their similarity. We shall call this function a *kernel* and denote it by

$$k(x_i, x_j). \tag{2.2}$$

A very widely used kernel function is simply the dot or scalar product - the sum of the pairwise products of the elements in x_i and x_j. In the case where the two vectors have been normalised to have length 1, this function computes the cosine of the angle between them. One of the main benefits of kernel methods is that we can perform our classification task in any space, so long as we can calculate the dot product of our patterns in that space. We do not actually require an explicit functional mapping from our input space into this new *feature* space.

However, let us introduce SVM's in their simplest form - that is, the similarity measure is the dot product in the original (input) space. Consider the two dimensional data shown in figure 2.1. We wish to find a linear separating hyperplane (in this case, simply a line) that separates the one class from the other of the form

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \qquad \mathbf{w} \in \mathcal{R}^2, b \in \mathcal{R} \tag{2.3}$$

where '.' denotes the dot product. This will results in a decision function of the form

$$f(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b) \tag{2.4}$$

If the data is linearly separable (i.e., if it is possible to separate the classes perfectly with a straight line), the following two constraints are implied

$$\mathbf{w} \cdot \mathbf{x}_i + b \geq 0 \ \ \forall \{i : y_i = +1\} \tag{2.5}$$

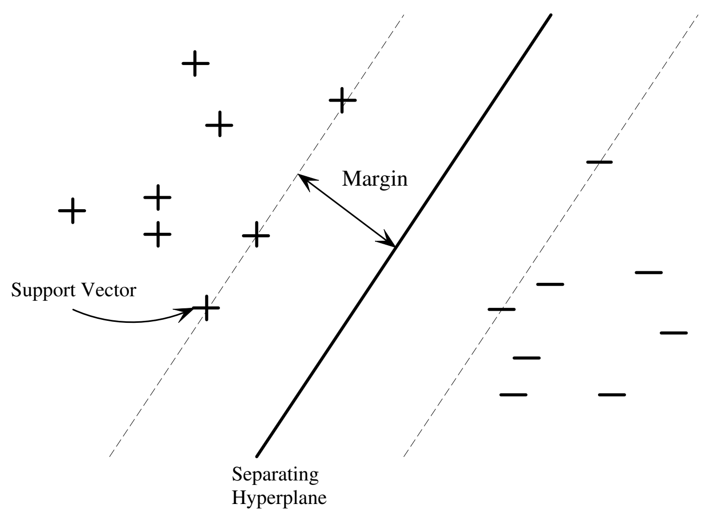


Fig. 2.1: A toy two dimensional classification problem separated by a linear hyperplane.

and

$$\mathbf{w} \cdot \mathbf{x}_i + b \leq 0 \quad \forall \{i : y_i = -1\}. \tag{2.6}$$

Due to the use of the sign function, the decision function will be invariant under any positive re-scaling of its argument. Therefore, we can re-scale $\mathbf{w}$ and b such that the closest points to the hyperplane on either side satisfy

$$|\mathbf{w} \cdot \mathbf{x}_i + b| = 1 \tag{2.7}$$

and hence can re-write our constraints as

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \quad \forall i. \tag{2.8}$$

As can be seen in 2.1, if the classes are linearly separable there must exist an infinite number of different hyperplanes that perfectly classify the training data - that is, they will all minimise the empirical risk, defined as

$$R_{emp}[f] = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} |f(\mathbf{x}_i) - y_i|. \tag{2.9}$$

However, it is unlikely that they will all perform equally well when presented un-seen data - i.e. they may not all minimise the true risk

$$R[f] = \int \frac{1}{2}|f(\mathbf{x}) - y|p(\mathbf{x}, y)d\mathbf{x}dy. \tag{2.10}$$

It is in the method of choosing one hyperplane from this infinite set where many learning algorithms differ. SVM's choose the one single hyperplane that maximises the distance between the hyperplane and the closest points on either side - the *margin*. This solution is well grounded in statistical learning theory [76] and, as we shall see, leads to a convenient learning formulation.

Considering one of the closest points from each class (say $\mathbf{x}_1$ and $\mathbf{x}_2$) in 2.1, we can algebraically calculate the margin (γ) by projecting the vector ($\mathbf{x}_1 - \mathbf{x}_2$) onto the vector normal to the hyperplane: $\mathbf{w}/||\mathbf{w}||$

$$\gamma = (\mathbf{x}_1 - \mathbf{x}_2) \cdot \frac{\mathbf{w}}{||\mathbf{w}||} = \frac{2}{||\mathbf{w}||}. \tag{2.11}$$

Hence, maximising the margin is equivalent to

$$\begin{array}{ll} \text{minimise} & \frac{1}{2}||\mathbf{w}||^2 \\ \text{subject to} & y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad \forall i \end{array} \tag{2.12}$$

This optimisation problem can be formulated as a Lagrangian where we introduce a Lagrangian multiplier α_i (such that $\alpha_i \geq 0$) for each of the m constraints

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{m} \alpha_i (y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1) \tag{2.13}$$

which must be minimised with respect to $\mathbf{w}$ and b and maximised with respect to $\boldsymbol{\alpha}$. At the solution, the derivatives of the Lagrangian with respect to $\mathbf{w}$ and b must vanish. Hence,

$$\frac{\delta L(\mathbf{w}, b, \boldsymbol{\alpha})}{\delta \mathbf{w}} = 0 = \mathbf{w} - \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \tag{2.14}$$

and

$$\frac{\delta L(\mathbf{w}, b, \boldsymbol{\alpha})}{\delta b} = 0 = \sum_{i=1}^{m} \alpha_i y_i. \tag{2.15}$$

Substituting these two expressions into our Lagrangian leaves us with a dual optimisation problem over the Lagrange multipliers α_i.

$$L(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \tag{2.16}$$

which must be maximised subject to

$$\alpha_i \geq 0, \quad \forall i, \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{2.17}$$

There are several important points arising from this optimisation formulation. Firstly, it is simply a bounded quadratic programming task and could be expressed as the following

$$L(\boldsymbol{\alpha}) = \mathbf{c}^T \boldsymbol{\alpha} + \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{H} \boldsymbol{\alpha} \tag{2.18}$$

where $\mathbf{c} = \mathbf{1}$ (i.e. a vector of ones) and $(\mathbf{H})_{ij} = -y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$. Therefore, it has a unique solution which can be easily computed using any one of a number of widely available packages. Secondly, if a solution exists, we know that the following constraints must be met

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad \forall i \tag{2.19}$$

and for the closest patterns to the separating hyperplane, this inequality must be an equality by definition. Therefore, from our primal formulation (2.13) which we must maximise with respect to α_i, patterns that are not the closest to the hyperplane must have $\alpha_i = 0$. Now, recall that our decision function is based on the sign of

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b \tag{2.20}$$

and at our solution,

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \tag{2.21}$$

and b can be calculated from

$$b = -\frac{1}{2}\left[\max_{\{i|y_i=-1\}}\left(\sum_{j=1}^{m} y_j\alpha_j(\mathbf{x}_i\cdot\mathbf{x}_j)\right) + \min_{\{i|y_i=+1\}}\left(\sum_{j=1}^{m} y_j\alpha_j(\mathbf{x}_i\cdot\mathbf{x}_j)\right)\right]. \tag{2.22}$$

Substituting 2.21 into 2.20 leaves us with the following decision function for an unlabelled pattern $\mathbf{x}$

$$f(\mathbf{x}) = \text{sign}\left[\sum_{i=1}^{m}\alpha_i y_i(\mathbf{x}_i\cdot\mathbf{x}) + b\right] \tag{2.23}$$

so patterns that are not the closest patterns on either side of the hyperplane will have no bearing whatsoever on the classification of new points. Intuitively this makes sense - our choice of separating hyperplane was based on maximising the margin which would obviously only be dependent on the closest points on each side. These closest points with $\alpha_i > 0$ are known as the *Support Vectors.* Finally, notice that in both our minimisation and decision function, the patterns only appear inside dot products. Therefore, it does not matter in which space we perform our classification as long as we are able to compute the dot product in that space. For this reason, the dot products are often replaced with the more general kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$. It is beyond the scope of this document to go into appropriate kernel methods in much detail - indeed, most applications of SVM's to gene expression data have used the most basic linear kernel (the dot product in the original, input space) as the data tends to be separable and therefore with no other prior knowledge of the problem, the simplest solution is likely to be the best. There are two important feature that a kernel function must satisfy, that is, it should be positive definite

Definiton 1 *A real $m \times m$ matrix $\mathbf{K}$ is positive definite if*
$\sum_{ij} c_i c_j K_{ij} \geq 0$
for all $c_i \in \mathbb{R}$

and symmetric - $K_{ij} = K_{ji}$.
Two popular kernel functions are radial basis functions (RBF's) with width σ

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left\{-\frac{(\mathbf{x}_i - \mathbf{x}_j)^2}{2\sigma^2}\right\} \tag{2.24}$$

and n^{th} order polynomials

$$k(\mathbf{x}_i, \mathbf{x}_j) = ((\mathbf{x}_i \cdot \mathbf{x}_j) + 1)^n \tag{2.25}$$

2.2 Extensions to the standard model

2.2.1 Soft Margins

So far, we have only considered data that are linearly separable in the feature space. Mislabelled data or data points badly corrupted by noise could cause the algorithm to fail. One way to overcome this is to slightly loosen our original constraints with the addition of positive slack variables ξ_i to allow data points to lie on the *wrong* side of the hyperplane or inside the margin band. Our original constrained minimisation 2.12 now becomes

$$\begin{array}{ll} \text{minimise} & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{m}\xi_i \\ \text{subject to} & y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \;\; \forall i \end{array} \tag{2.26}$$

which, when the derivations above are repeated leads to maximising 2.16 subject to

$$0 \leq \alpha_i \leq C \;\; \forall i, \;\; \text{and} \;\; \sum_{i=1}^{m} \alpha_i y_i = 0 \tag{2.27}$$

where C effectively controls the trade-off between maximising the margin and minimising the number of training errors. One intuitive way of thinking about this is that the higher its respective α_i value the more influence a given training point has on the final solution. The box constraint now imposed on α_i limits this effect and hence the impact of outlying points.

2.2.2 Multi-Class Classification

So far we have discussed binary classification, however, some tasks will involve multi-class prediction. A number of methods have been proposed for performing multi-class prediction with SVMs but few are demonstrably better than using a set of 'one-against-all' classifiers (this approach can be used for perceptrons too). 'One-against-all' classifiers have the drawback that several members of the set may respond to a given input. As an alternative it is possible to use a series of binary classifiers in a tree [56] (Fig. 2.2) to determine the outcome, and this method works satisfactorily if the number of classes is small.

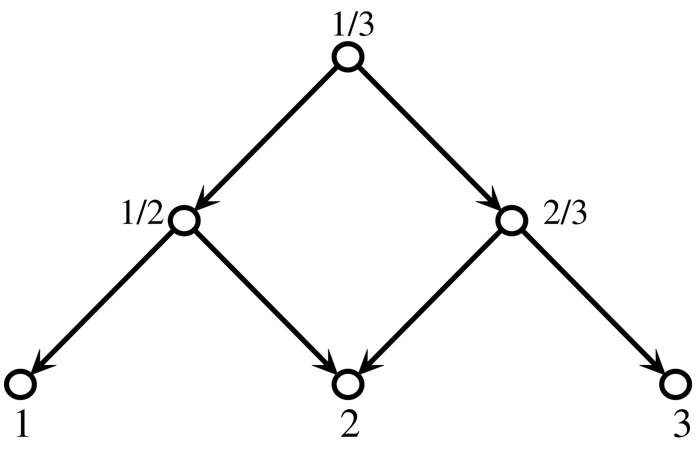


Fig. 2.2: A multi-class classification problem can be reduced to a series of binary classification tasks (represented by the top two levels in the tree).

2.2.3 Confidence Measure

If used for predicting diagnostic categories, for example, it is useful to have a confidence measure for the class assignment in addition to determining the class label. An SVM with linear kernel does have an inbuilt measure of confidence that could be exploited to provide a confidence measure for the assigned class, i.e. the distance of a new point from the separating hyperplane (Figure 2.1). A test point a large distance from the separating hyperplane should be assigned a higher degree of confidence than a point which lies close to the hyperplane.

The task of mapping this distance to probabilities has been solved in various ways. Here, we concentrate on one particular method, proposed by Platt [57]. Recall that the output of an SVM, before thresholding to ± 1, is given by

$$f(\mathbf{z}) = h(\mathbf{z}) + b \tag{2.28}$$

where:

$$h(\mathbf{z}) = \sum_i y_i \alpha_i K(\mathbf{x}_i, \mathbf{z}) \tag{2.29}$$

We use a parametric model to fit the posterior probability $P(y = 1|f)$ directly. The choice of parametric function for the posterior is the sigmoid (for more details see [57]):

$$P(y = 1|f) = \frac{1}{1 + \exp(Af + B)} \tag{2.30}$$

with the parameters A and B found from a training set (f_i, y_i). Define t_i as the target probabilities:

$$t_i = \frac{y_i + 1}{2} \tag{2.31}$$

i.e. using $y_i \in \{-1, 1\}$ we have $t_i \in \{0, 1\}$. We now minimise the log likelihood of the training data:

$$min \left[-\sum_i t_i \log(p_i) + (1 - t_i) \log(1 - p_i) \right] \tag{2.32}$$

where p_i is simply (2.30) evaluated at f_i. This is a straightforward 2-dimensional minimisation that can be solved using any one of a number of optimisation routines. Once the sigmoid has been found using this training set, we can use (2.30) to calculate the probability that a given test point belongs to each class. One question remains: how do we construct a training set to fit the sigmoid. The obvious choice would be the examples from the training set that were used to train the classifier. However, the training process biases the outputs for the support vectors to be ± 1 which is a very unlikely value for a new test point. However, for a linear SVM, the number of support vectors is usually reasonably low and so the bias should not be too severe. Therefore, it is generally acceptable to use the values from the training set to fit the posterior sigmoid. Figure 2.3 shows the training values and fitted sigmoid from a microarray dataset for ovarian cancer [66]. Note that no points are present in a large band between $+1$ and -1, due to the use of a hard margin and the data being linearly separable.

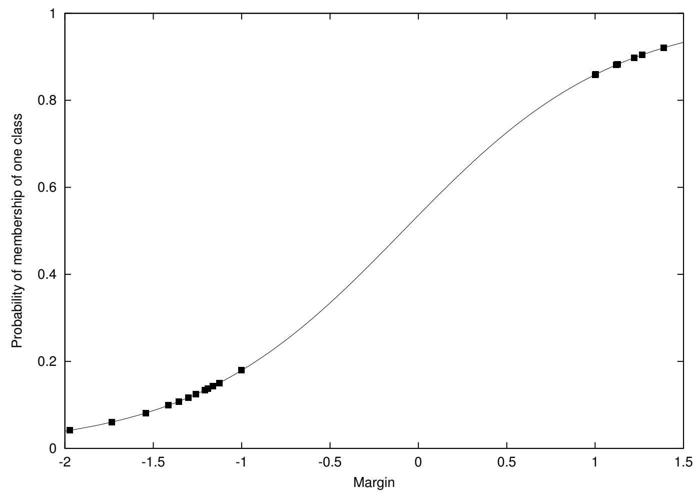


Fig. 2.3: Probability of membership of one class (y-axis) versus distance from the hyperplane. The plot shows the training points and fitted sigmoid for an ovarian cancer data set [66]. A hard margin was used which explains the absence of points in the central band.

2.2.4 Feature Selection

One characteristic of microarray data is that the number of features is usually very large and typically of the order of tens of thousands, often approximating the set of known genes in size. If a broad search is being pursued, the vast majority of these features are likely to be irrelevant for a given classification task and ideally we would like to remove them. Feature selection has two benefits. Firstly, large numbers of irrelevant features effectively inject noise into the classification task and can destroy generalisation, as we will illustrate later with an example. Secondly, from the viewpoint of interpretation, feature selection also highlights the most relevant features or genes in the data. Two general approaches may be used: *filter methods* in which features are scored individually (e.g. using statistical methods) prior to use of the classifier, and *wrapper methods* in which the algorithm uses an internal procedure to eliminate redundant features.

Filter Methods

The choice of filter method can amount to a prior assumption about the way in which the significance of individual features are ranked. Roughly speaking, filter methods can be viewed as falling into two groupings - those measures more influenced by the consistency of the difference between classes and those more influenced by magnitude differences. For example, the set of ratios (1.1,1.1,1.1,1.1) is consistently different from the set (0.9,0.9,0.9,0.9) and features in both can be separated by a simple threshold (1.0). On the other hand there is a significant difference in the means of the set of ratios (1.0,1.0,5.0,5.0) and (1.0,1.0,0.1,0.1) even though the first two members of each set are the same. As for classification algorithms there are a large number of methods which can be employed. In this section we will therefore describe three commonly used approaches: the Mann-Whitney test and the TNoM score, the Fisher and Golub scores and the t-test (which also comes in many variants). To give an impression of their performance we will also evaluate these scores on a new dataset from cancer research. Statistical scoring based on ranking will be most influenced by consistency of a difference. If all values belonging to one class are ranked higher than all members of the other class this is determined as an improbable event even if the means of both classes do not differ a great deal. For example, for binary classification, the Mann-Whitney U test provides a measure of the difference between the medians of two populations [62] based on ranking. For a particular feature, two populations are determined from the expression ratios of the two distinctly labelled sample sets. The aim of the test is to estimate the probability that the two populations were drawn from an identical distribution. If this probability is very low, the feature is consistent with the class labels and will be significant. We start by combining the samples from the two classes and ranking them in numerical order. Each sample is then ranked with a value equal to its position in the line (i.e. a rank between 1 and (n_1+n_2) where n_1 and n_2 are the number of samples in classes 1 and 2.). If expression ratios are tied, they are given the average rank. The ranks for each of the two sample sets are summed. If the sums are R_1 and R_2, we now calculate the U statistic from $U_1 = n_1 n_2 + 0.5 n_1(n_1+1) - R_1$ and $U_2 = n_1 n_2 + 0.5 n_2(n_2+1) - R_2$. Choosing the smaller of U_1 and U_2 as the test statistic U we calculate $z = (U - \mu_U)/\sigma_U$ using:

$$\mu_U = \frac{n_1 n_2}{2} \qquad \sigma_U = \sqrt{\frac{n_1 n_2 (n_1 + n_2 + 1)}{12}} \tag{2.33}$$

z is a normally distributed random variable ($\mathcal{N}(0,1)$) and hence it is straightforward to determine the probability: the lower the probability the more significant the feature for discriminating the two classes.

Designed specifically for microarray datasets the Threshold Number of Mis-classifications (TNoM) score [5] is closely related to ranking scores and will give a similar feature ranking to the Mann Whitney test, for example. We calculate the TNoM score by finding the best classification performance possible for the given feature. For the m values of the feature x_i (components of the feature vector) and label y_i we evaluate:

$$TNoM = \min_{a,b} \sum_{i=1}^{m} H\left[y_i \left(a x_i + b\right)\right] \tag{2.34}$$

thus we count an error every time $y_i \neq sign(a x_i + b)$. Since the argument inside the Heaviside step function is invariant under an arbitrary positive rescaling we can set $a = 1$ and evaluate the score using a 1-dimensional minimisation on b. If we are able to find a threshold such that all values above it belong to one class and all below belong to the other, then the TNoM score is 0. If the best threshold involves one misclassification the TNoM score is 1, etc. Given a particular class distribution we can also calculate the probability of getting a feature with any particular TNoM score. We can then compare the cumulative distribution function of the TNoM scores for a microarray dataset with the theoretical scores for a null model (i.e. purely random data) and thus determine if there is a significantly high number of low (i.e. good) scoring features compared to random occurrence and therefore if the data carries a significant information load. For example, in Figure 2.4 we give a plot of the cumulative distributions for an ovarian cancer data set [66]. Here, the curve determined from the ovarian cancer dataset lies considerably to the left of the theoretical curve expected for purely random data, suggesting a large number of discriminating features and significant information content.

The second family of feature scoring techniques has more of an emphasis on the dif-

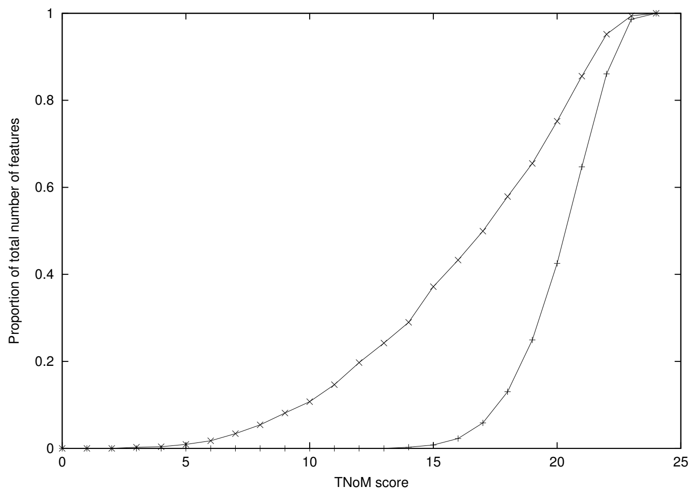


Fig. 2.4: TNoM cumulative score curves for an ovarian cancer dataset [66]. The x-axis is TNoM score and y axis is the proportion of the total number of features. The left-hand curve corresponds to actual scores and the right-hand curve to the theoretical score for a null (random) model. As the curve from the actual scores lies significantly to the left this means the data contains a significant information content.

ferences in magnitudes of the expression values between classes. Thus if we derive the means and standard deviations for the samples in each class, a good discriminating feature would have a large separation between the means and small standard deviations. If we define the means of the samples in the two classes as μ_1 and μ_2 and their standard deviations as σ_1 and σ_2, the Fisher (F) and Golub (G) scores are defined as follows

$$F = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2} \qquad\qquad G = \frac{|\mu_1 - \mu_2|}{\sigma_1 + \sigma_2}. \tag{2.35}$$

The final score we will consider is the well-known t-test for the difference between means of two populations. We calculate the means and variances (μ_1, μ_2, σ_1^2 and σ_2^2) for the two classes and then a weighted average of the two variances, with

$$s^2 = \frac{(n_1 - 1)\sigma_1^2 + (n_2 - 1)\sigma_2^2}{n_1 + n_2 - 2}. \tag{2.36}$$

and a test statistic t using

$$t = \frac{\mu_1 - \mu_2}{s\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}} \tag{2.37}$$

from which a probability measure can readily be obtained from the Student's distribution.

Wrapper Methods - Recursive Feature Elimination

Rather than a prior scoring of features we could use a procedure within the algorithm to eliminate redundant features. As an illustration we will consider one method for SVMs which removes irrelevant features during the training process. When using a linear kernel, we noted that the weight vector for an SVM can be expressed as $\mathbf{w} = \sum_{i=1}^{m} y_i \alpha_i \mathbf{x}_i$. The smallest absolute value of the weight vector will have least influence in the decision function and will therefore be the best candidates for removal. For the *Recursive Feature Elimination* (RFE) method proposed by Guyon et al [25], the SVM is trained with the current set of features and the best candidate feature for removal is identified via the weight vector. This feature is removed and the process repeated until termination. One disadvantage of this method is that the process will be slow if there are a large number of features (typically the case for modern high density microarrays), though features could be removed in batches, of course. The algorithm can be terminated if the test error is logged throughout and passes through a minimum though, for SVMs, theoretical criteria such as LOO bounds (which estimate generalisation performance) can also be used [34]. This approach is general and can be applied to the weights generated using the perceptron algorithm, for example.

Aside from RFE there are a number of other approaches where feature selection is implemented directly within the algorithm. For example, with Bayesian approaches a *Bayesian prior* can be incorporated into the design of a classifier favoring sparse solutions, i.e. there is an explicit preference for a solution with a very limited number of features (this is known as Automatic Relevance Determination (ARD), see [52]). Bayesian ARD algorithms for this purpose have been developed and work well with microarray datasets [44].

Feature Selection: A case study

To illustrate the above we will demonstrate how these techniques have been applied to a new dataset for predicting relapse versus non-relapse for a paediatric malignancy (Williams et al [80]). This represents novel work performed in collaboration with Kathy Pritchard-Jones at the Institute for Cancer Research. In this study cDNA microarrays were used with an approximately balanced dataset of 27 samples. Normalisations were implemented using a pre-existing software package [30] based on the Statistics for Microarray Analysis (SMA) R package of Speed et al [31].

A Support Vector Machine with linear kernel was used with filter methods for the feature selection. Recursive elimination of features was not used because of the large number of features (17790). Given the small size of the dataset leave-one-out (LOO) appears the best first strategy for evaluating the test error. During evaluation of the test error the 26 examples in the training set change with every leave-one-out rotation. Consequently, to derive a fair test statistic, the statistical scores were determined for each of the 27 evaluations on the test point without incorporating it into the computation of the score. In Figures 2.5, 2.6 and 2.7 we show the LOO test error (y-axis) versus number of top-ranked features (x-axis) remaining for Fisher, Mann-Whitney and t-test scoring of features. All three scores indicate prediction is poor if all the features are used. However, good prediction is achievable with a small number of features. Thus for the Fisher score the minimal LOO test error is 5/27, for Mann-Whitney 4/27 and for the t-test 1/27. For 9-fold cross-validation with 1000 random reshufflings of the order we get a $6.4 \pm 1.2\%$ precentage test error with 3 features remaining. These results indicate prediction of relapse can be achieved, though any final confirmation would await evaluation on *de novo* data and biological confirmation of the genes used. If, as here, prediction can be achieved with the expression profile of a small set of genes then the result is interesting but should be viewed with caution. If a set of genes have highly correlated expression then only one member may be needed for building a successful predictor (the rest are effectively redundant), though this gene may not be as signifi-

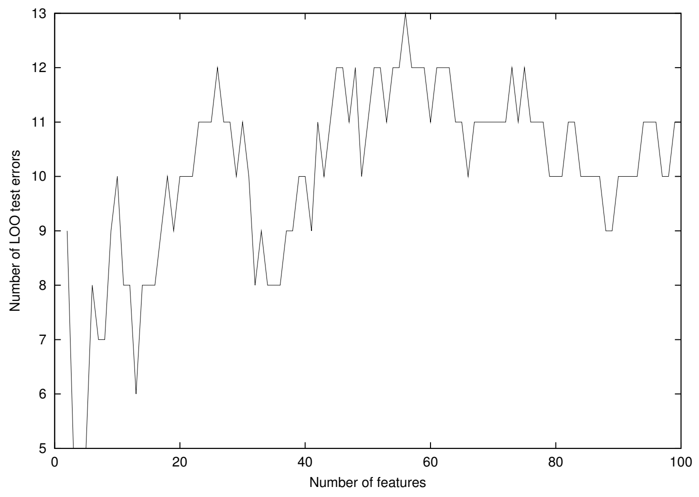


Fig. 2.5: The number of LOO test errors (y-axis) versus number of top-ranked features (x-axis) remaining with feature-ranking by the Fisher score for predicting relapse or non-relapse. For 4 or fewer features a non-zero training error appears and the test error rises from the minimum of 5/27.

cant as some of the others. If, for example, the expression pattern of a particular gene is associated with positional effects (e.g. when a biologically significant over-expressed gene is located in a region of the genome that has become duplicated in tumour cells), other genes that are not relevant to tumour outcome may be co-regulated by the same mechanism. Their apparent relevance only derives from their position in this genomic region. Consequently the significance of genes should be evaluated by other methods independently of the feature selection used by the classifier.

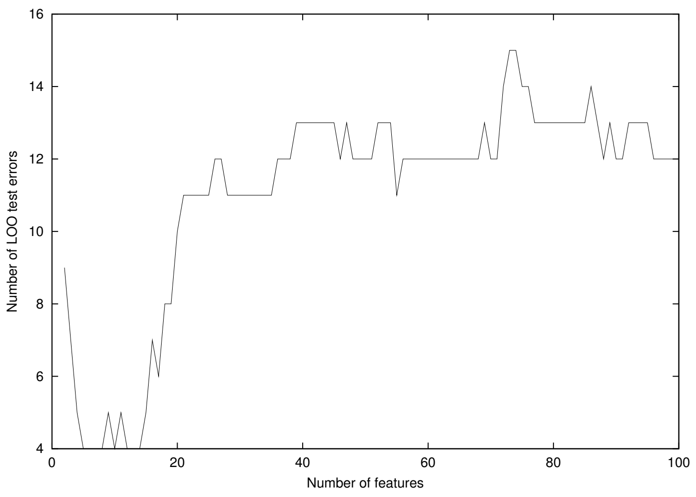


Fig. 2.6: The number of LOO test errors (y-axis) versus number of top-ranked features remaining (x-axis) with ranking of features by the Mann-Whitney score [62]. The minimum LOO test error improves on the Fisher score in Fig. 2.5 with a minimum of 4/27.

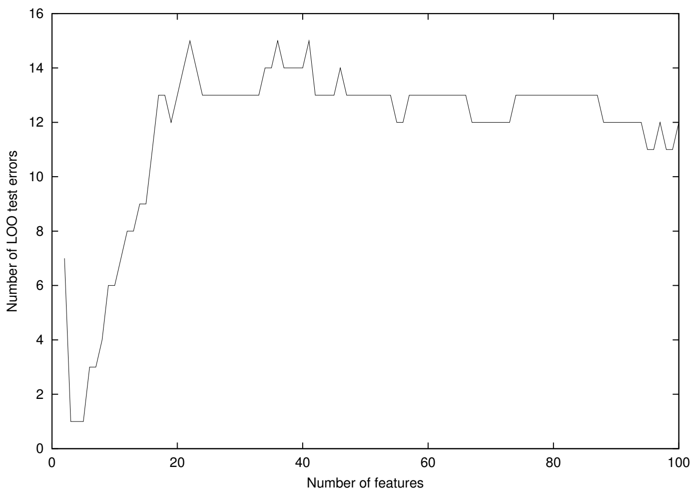


Fig. 2.7: The number of LOO test errors (y-axis) versus number of top-ranked features remaining (x-axis) with ranking of features by the t-test. The minimum LOO test error was 1 from 27 with 3-5 features remaining. With 2 features remaining a non-zero training error was recorded. The minimal test error is lower than for Fisher (Fig. 2.5) and Mann Whitney [62] (Fig. 2.6) and the curve is noticeably smoother.

Chapter 3

Probabilistic Models

3.1 Probabilistic Modelling

Kernel methods and particularly Support Vector Machines introduced in the previous chapter represent just one method for inferring information from data. An alternative to such non-parametric approaches is to try and model in some way the process that originally generated the observed data. For example, in a simple binary classification task, rather than trying to construct some kind of decision boundary it may be more sensible and provide more information to try and model in some way the two disparate data sources. Data can then be classified by evaluating which of the two sources was more likely to have produced it.

Probability theory provides a rigorous framework to perform such modelling and generally one can classify probabilistic learning methods into one of two categories - *Bayesian* and *frequentist*. To illustrate the difference between these two approaches, consider the challenge of trying to predict the outcome of tossing a coin of unknown properties. The frequentist's viewpoint would be interested in using observations to calculate a single quantity - the probability of Heads. This would be achieved by simply looking at the proportion of heads in our observations. This single probability could then be used directly to predict the outcome of a future coin toss. The Bayesian on the other hand, rather than considering a single value for the required probability would encode belief in the value through a probability distribution. Observations would be used to update this distribution and predictions would be based on its expected value. More precisely, let θ correspond to the probability of interest, i.e. $p(head)$ and let $\mathcal{X} = x_1 \dots x_n$ be our observations ($x_i = 1$ if the outcome of the i^{th} toss was a head and 0 otherwise). The frequentist approximation to θ would be calculated as

$$\theta = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{3.1}$$

whilst the Bayesian would first specify a *prior* distribution over θ and then update this, using Bayes rule to give a *posterior* distribution

$$p(\theta|\mathcal{X}) = \frac{p(\mathcal{X}|\theta)p(\theta)}{p(\mathcal{X})} = \frac{p(\mathcal{X}|\theta)p(\theta)}{\int_\theta p(\mathcal{X}|\theta)p(\theta)d\theta} \tag{3.2}$$

where $p(\theta)$ is the *prior* distribution over the values of θ and $p(\mathcal{X}|\theta)$ is known as the *likelihood* and can be thought of as how likely it would be to observe the data that we have, given our hypothesis (θ). To predict the value of an unknown observation, $\hat{x}$, an expectation is taken with respect to this posterior distribution

$$p(\hat{x}|\mathcal{X}) = \int_\theta p(\hat{x}|\theta)p(\theta|\mathcal{X})d\theta. \tag{3.3}$$

Notice that we can perform our posterior update iteratively - for example, having seen n observations, we could use the current posterior as the prior for the next observation

$$p(\theta|x_1, \dots, x_n, x_{n+1}) = \frac{p(x_{n+1}|\theta)p(\theta|x_1, \dots, x_n)}{\int_\theta p(x_{n+1}|\theta)p(\theta|x_1, \dots, x_n)d\theta}. \tag{3.4}$$

Figure 3.1 shows how the posterior is iteratively updated from a uniform prior for three observations (head, head, tail). One of the benefits of the Bayesian approach is how it avoids over-fitting the model to the training data through the use of the prior distribution. For example, after the first two observations (both heads) in the example above, the Bayesian prediction of an observation being heads would be 0.75 whilst the frequentist approach would assign a probability of 1.

Unfortunately, in most useful examples, the integrals in 3.2 and 3.3 are analytically intractable and have to be approximated via numerical methods - for example, sam-

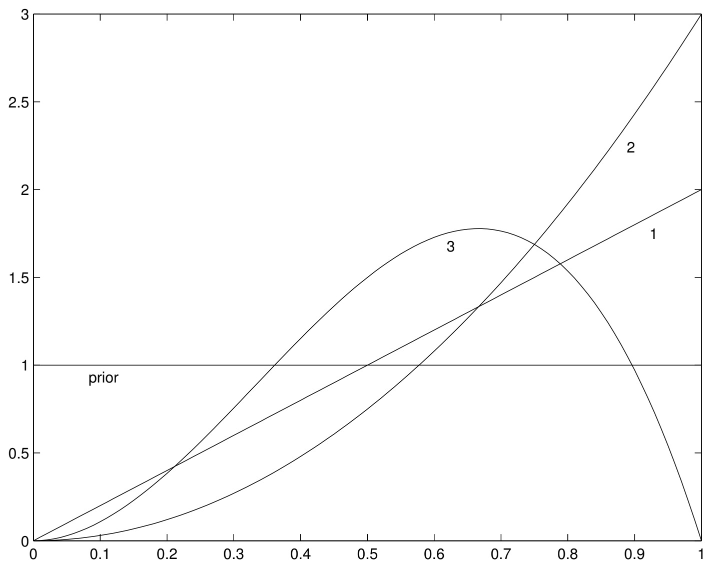


Fig. 3.1: Evolution of the posterior probability for the coin tossing problem as observations are received. In this case, the observations are head, head, tail.

pling from the appropriate distribution. In large problems, these methods also become infeasible and various approximations are made. Two common methods are maximum likelihood (ML) and maximum a posteriori (MAP). Maximum likelihood finds the single parameter that maximises the likelihood term in 3.2 whilst the MAP solution is the single parameter value that gives the maximum of the posterior distribution $p(\theta|\mathcal{X})$, equivalent to maximising the numerator of 3.2 as $p(\mathcal{X})$ is not parameter dependent.

3.2 Graphical Models

Graphical models provide a means of visualising probabilistic models. Figure 3.2 shows

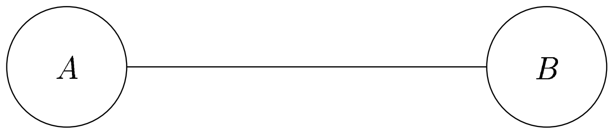


Fig. 3.2: Graphical representation of a simple probabilistic model

a very simple graphical model corresponding to the distribution $P(A, B)$ of the two random variables A and B. Following Heckerman [27] (albeit with a slightly simplified model) A could represent the binary variable *start* denoting whether or not a car will start. In this case, B might represent whether or not there is fuel present in the tank. Clearly, any useful probabilistic model will be considerably more complicated than this and it may seem daunting to consider dealing with joint distributions over large numbers of variables. However, it is likely that by exploiting conditional independencies between variables we can greatly simplify most models. Graphical models provide a means of visualising these conditional independencies. Figure 3.3 shows how

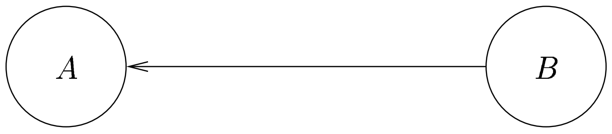


Fig. 3.3: Graphical representation of a conditional probabilistic model

dependencies between variables are represented. The flow of information is represented by the direction of the arrow so, in this example, our belief in A is dependent on our knowledge of B. In our trivial car example, this is equivalent to the assumption that the probability of the car starting is dependent on whether or not there is any fuel in the tank. This particular dependency defines the following decomposition of the joint probability $P(A, B)$

$$P(A, B) = P(A|B)P(B). \tag{3.5}$$

Consider the slightly more complicated model shown in figure 3.4. This corresponds to the following decomposition of the joint distribution

$$P(A, B, C, D, E) = P(E|D, C)P(C)P(D|A, B)P(A)P(B). \tag{3.6}$$

We say that variable E is conditionally independent on A and B as, although there is obviously a dependency between them, it is a dependency through the variable D. Hence, conditioned on a value of D, E can be considered independent of A and B.

As well as random variables, any real model will also consist of observable variables. These are shown in our graphical representation as shaded circles. For an example, see figure 3.5. Notice also the *plate* in this diagram. The plate is a helpful way of

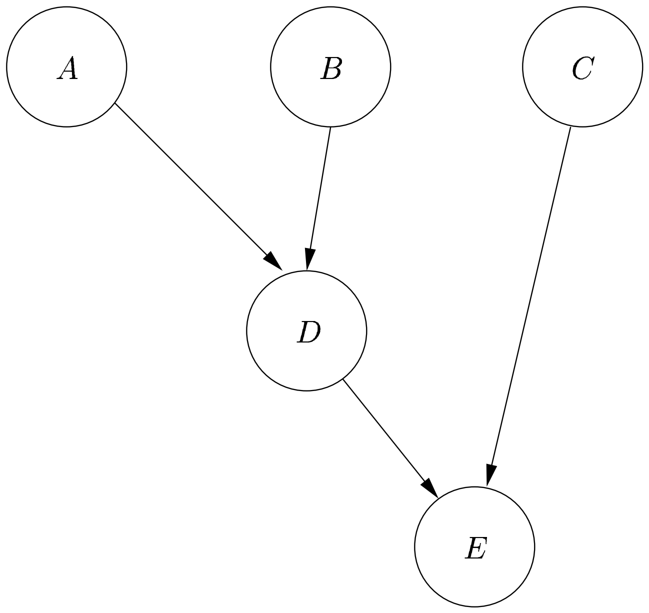


Fig. 3.4: Graphical representation of a conditional probabilistic model

describing repeated sampling. Figure 3.6 shows a graphical model without plates and the corresponding plates diagram. The plate explicitly states the assumption that the N variables $A_1, \ldots, A_N$ are conditionally independent and identically distributed given B. For a more detailed introduction to graphical models, see [36]. In figure 3.5, N data points have been observed and the model being used assumes that they were all drawn independently from the distribution defined by the node A - for example, the observed values may all be from $\mathbb{R}^1$ and A may define a one dimensional normal distribution. A possible inference task would be to try and infer the parameters of this normal distribution from the observed data. In the next section we will discuss one of the possible methods for performing such inference.

3.3 Variational Inference

In this section we briefly introduce the use of variational methods to learn parameters in graphical models. For a more detailed introduction, the reader is referred to [35]. As discussed previously, any probabilistic graphical model we might be interested in is likely to consist of observable variables, hidden or latent variables and parameters.

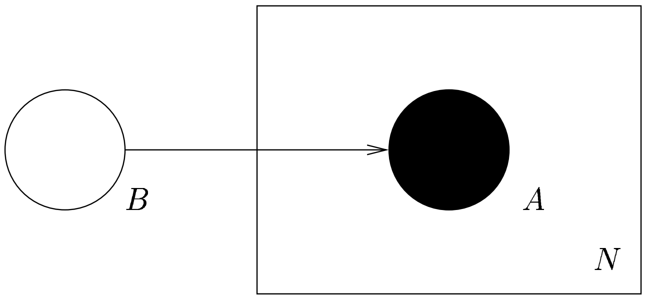


Fig. 3.5: Graphical representation of a conditional probability model with observed node and repeated sampling.

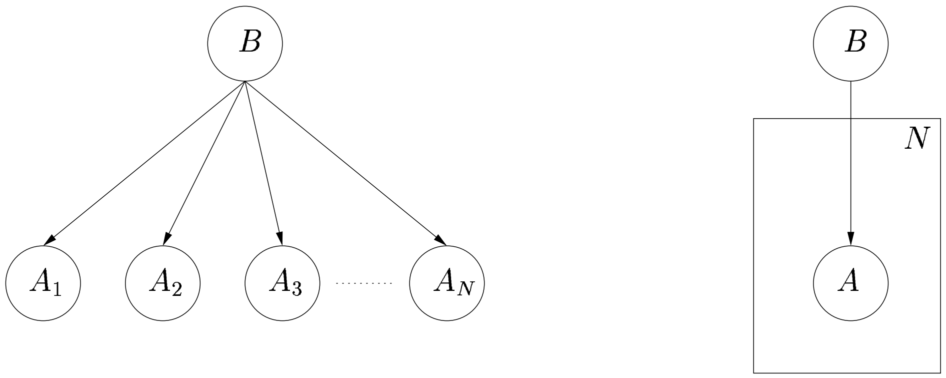


Fig. 3.6: The concept of plates to represent repeated sampling in graphical models. The model shown here on the right is a shorthand way of depicting the model on the left.

Given a particular model structure, we are interested in using observed data to infer suitable values for the parameters in question. In some cases, we will be interested in a fully Bayesian approach whilst in others we may, due to reasons of complexity or suitability be satisfied with single valued estimates of our parameter values. Variational methods can be used to do both. The main feature of such techniques is the introduction of variational distributions over latent variables, the use of which enables strict lower and upper bounds to be placed on distributions of interest. Ideally these bounds would be much easier to optimise than the distributions themselves, leading to accurate, tractable, approximations of the true distributions.

One method that we shall use later in this thesis is using variational methods to find maximum likelihood parameter values. Recall that the maximum likelihood solution is that set of parameter values that maximises the quantity $p(\mathcal{X}|\theta)$ where $\mathcal{X}$ is some set of observed data and θ represents the parameters in the model. For simplicity, we will often consider the natural logarithm of this quantity and we shall treat it as a function of the parameters, $\mathcal{L}(\theta)$. Now, given a graphical model with observed variables $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, latent variables $\mathbf{y}$ and parameters θ, this likelihood is given by the following

$$\mathcal{L} = \log P(\mathcal{X}|\theta) = \sum_{i=1}^{n} \log P(\mathbf{x}_i|\theta) = \sum_{i=1}^{n} \log \int P(\mathbf{x}_i|\mathbf{y}, \theta) P(\mathbf{y}) d\mathbf{y} \tag{3.7}$$

where the integral would be replaced by a sum for discrete latent variables. We have also used the assumption that the data are independent and identically distributed (iid). We wish to maximise this quantity with respect to θ, however this is far from straightforward due to the potential coupling of the parameters induced by the logarithm of the integral (or sum). However, by introducing *any* distribution over the latent variables, we can lower bound this quantity as follows

$$\begin{aligned} \sum_{i=1}^{n} \log \int P(\mathbf{x}_i, \mathbf{y}|\theta) d\mathbf{y} &= \sum_{i=1}^{n} \log \int Q_{\mathbf{y}}(\mathbf{y}) \frac{P(\mathbf{x}_i, \mathbf{y}|\theta)}{Q_{\mathbf{y}}(\mathbf{y})} \\ &\geq \sum_{i=1}^{n} \int Q_{\mathbf{y}}(\mathbf{y}) \log \frac{P(\mathbf{x}_i, \mathbf{y}|\theta)}{Q_{\mathbf{y}}(\mathbf{y})} \\ &= \mathcal{L}'(Q, \theta) \end{aligned} \tag{3.8}$$

where we have made use of that fact that log is a concave function and used Jensen's inequality

Definiton 2 *Jensen's inequality. For any concave function f,*
$f(\boldsymbol{E}_{p(z)}[z]) \geq \boldsymbol{E}_{p(z)}[f(z)]$
where the expectation $\boldsymbol{E}_{p(z)}[f(z)] = \int f(z)p(z)dz$.

We now wish to maximise this bound with respect to both Q and θ. This may seem more complicated than simply maximising it with respect to θ but the fact that the log has moved inside the summation has considerably simplified matters. We can also take advantage of the following observation. Setting $Q_{\mathbf{y}}(\mathbf{y}) = P(\mathbf{y}|\mathbf{x}_i, \theta)$, leads to the following

$$\begin{aligned}\mathcal{L}'(Q,\theta) &= \sum_{i=1}^{n} \int P(\mathbf{y}|\mathbf{x}_i,\theta) \log \tfrac{P(\mathbf{x}_i,\mathbf{y}|\theta)}{P(\mathbf{y}|\mathbf{x}_i,\theta)} d\mathbf{y} \\ &= \sum_{i=1}^{n} \int P(\mathbf{y}|\mathbf{x}_i,\theta) \log P(\mathbf{x}_i|\theta) d\mathbf{y} \\ &= \mathcal{L}(\theta)\end{aligned} \tag{3.9}$$

i.e., our inequality has become an equality. Hence, we can employ the following iterative procedure

$$Q_{\mathbf{y}}^{t+1} \leftarrow \arg\max_{Q_{\mathbf{y}}} \quad \mathcal{L}'(Q,\theta^t) \tag{3.10}$$

$$\theta^{t+1} \leftarrow \arg\max_{\theta} \quad \mathcal{L}'(Q^{t+1},\theta). \tag{3.11}$$

Since after each Q update, our inequality is in fact an equality, i.e. $\mathcal{L}' = \mathcal{L}$ and given that the Q update doesn't change θ, we can guarantee that these iterations cannot possibly lead to a decrease in $\mathcal{L}$. This iterative update scheme is a form of the Expectation-Maximisation (EM) algorithm [11, 7] where the maximisation with respect to Q is the *E-step* and maximisation with respect to θ is the *M-step*. It is well known that this guarantees at least convergence to a local extrema.

In many cases, it may not be possible to evaluate the posteriors over the latent variables for the Q update. Notice however, that maximisation of $\mathcal{L}'$ with respect to Q is equivalent to minimising

$$\int Q_{\mathbf{y}}(\mathbf{y}) \log \frac{Q_{\mathbf{y}}(\mathbf{y})}{P(\mathbf{y}|\mathbf{x}_i,\theta)} d\mathbf{y}, \tag{3.12}$$

that is, the Kullback-Leibler (KL) divergence between Q and the required posterior. Although it may not be possible to analytically solve this minimisation, given a sensible choice of variational distribution (and one which factorises over the latent variables) it should be possible to differentiate with respect to the parameters of Q and iterate the resulting fixed point equations.

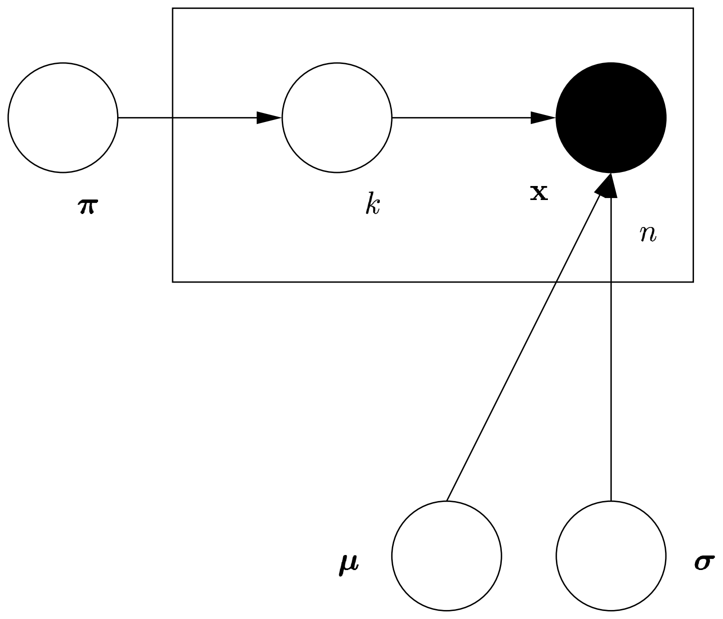


Fig. 3.7: Graphical representation of mixture model

3.4 Example - Gaussian Mixtures

To illustrate the methods introduced in this chapter, we will look at the example of inferring the parameters of a simple Gaussian mixture. The task is as follows, given a set of d-dimensional data vectors $\mathcal{X} = \{\mathbf{x}_i, \dots, \mathbf{x}_n\}$ we wish to find the density $p(\mathbf{x})$ from which they were produced where we assume that the density function is a mixture of $\mathcal{K}$ multi-variate Gaussians, that is

$$p(\mathbf{x}) = \sum_{k=1}^{\mathcal{K}} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \tag{3.13}$$

To simplify the analysis, we will make the naive-Bayes assumption. That is, there is no dependency between the d dimensions and hence the covariance matrices have only diagonal entries. We will denote the vector of these diagonal values as $\boldsymbol{\sigma}_k$.

Figure 3.7 shows the corresponding graphical model. Our $\mathbf{x}_i$ variables are observed and the k's are hidden. The model describes the following generative process

- For each $i = 1, \dots, n$

– Sample a component k from $\boldsymbol{\pi}$

– Sample $\mathbf{x}_i$ from $\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k)$.

A Bayesian might argue that a better model would include a Dirichlet prior distribution on $\boldsymbol{\pi}$ and also suitable priors on the $\boldsymbol{\mu}_k$ and $\boldsymbol{\sigma}_k$ parameters. However, it is unnecessary in this illustrative example.

The likelihood of our training data $\mathcal{X}$ is given as follows

$$\mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\pi}) = \sum_{i=1}^{n} \log \sum_{k=1}^{\mathcal{K}} \pi_k \prod_{j=1}^{d} p(x_{ij}|\mu_{kj}, \sigma_{kj}). \tag{3.14}$$

As discussed previously, the log outside the sum makes straight maximisation of this quantity with respect to the various parameters infeasible. For this reason, we introduce the training sample specific set of multinomial variational distributions $\mathbf{Q}$ subject to the following two conditions

$$p(\mathbf{Q}) = \prod_{i=1}^{n} p(Q_i) \tag{3.15}$$

i.e. they are factorisable over the training samples and

$$\sum_{k=1}^{\mathcal{K}} Q_{ik} = 1 \tag{3.16}$$

they sum to 1 over the $\mathcal{K}$ mixture components. Hence, we can view Q_{ik} as the posterior probability that $\mathbf{x}_i$ was produced by mixture component k. We can add these new variables to our likelihood and lower bound it as follows

$$\begin{aligned} \mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\pi}) &= \sum_{i=1}^{n} \log \sum_{k=1}^{\mathcal{K}} \tfrac{Q_{ik}\pi_k}{Q_{ik}} \prod_{j=1}^{d} p(x_{ij}|\mu_{kj}, \sigma_{kj}) \\ &\geq \sum_{i=1}^{n} \sum_{k=1}^{\mathcal{K}} Q_{ik} \log \left\{ \tfrac{\pi_k}{Q_{ik}} \prod_{j=1}^{d} p(x_{ij}|\mu_{kj}, \sigma_{kj}) \right\} \\ &= \mathcal{L}'(\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\pi}, \mathbf{Q}). \end{aligned} \tag{3.17}$$

We can maximise this lower bound by iterating the fixed point equations obtained by taking partial derivatives with respect to the various parameters. The obtained updates are given by (details in appendix B)

$$Q_{ik} = \frac{\pi_k \prod_{j=1}^{d} \mathcal{N}(x_{ij}|\mu_{kj}, \sigma_{kj})}{\sum_{k'=1}^{\mathcal{K}} \pi_{k'} \left[\prod_{j=1}^{d} \mathcal{N}(x_{ij}|\mu_{k'j}, \sigma_{k'j}) \right]} \tag{3.18}$$

$$\pi_k = \frac{1}{n} \sum_{i=1}^{n} Q_{ik} \tag{3.19}$$

$$\mu_{kj} = \frac{\sum_{i=1}^{n} Q_{ik} x_{ij}}{\sum_{i=1}^{n} Q_{ik}} \tag{3.20}$$

$$\sigma_{kj}^2 = \frac{\sum_{i=1}^{n} Q_{ik} (x_{ij} - \mu_{kj})^2}{\sum_{i=1}^{n} Q_{ik}}. \tag{3.21}$$

As an example, consider the data shown in figure 3.8. This data was drawn from the mixture model shown in figure 3.9. The model was initialised as follows. The means were drawn from mean zero isotropic Gaussians with variance of 1, the variances were set to 1 and the mixture coefficients, π_k were set to $1/\mathcal{K}$ where $\mathcal{K}$ in this case was set to 4. The above iterations were then used until the algorithm converged (approximately 25 iterations). The learned probability density can be seen in figure 3.10. In this example, for a given training set, it is possible to exactly evaluate the true likelihood and lower bound at every iteration. The evolution of the likelihood can be seen in figure 3.11. The solid (red) line corresponds to the true likelihood and the dashed (black) line to the lower bound. As required, both curves always increase and the lower bound is always below the true likelihood.

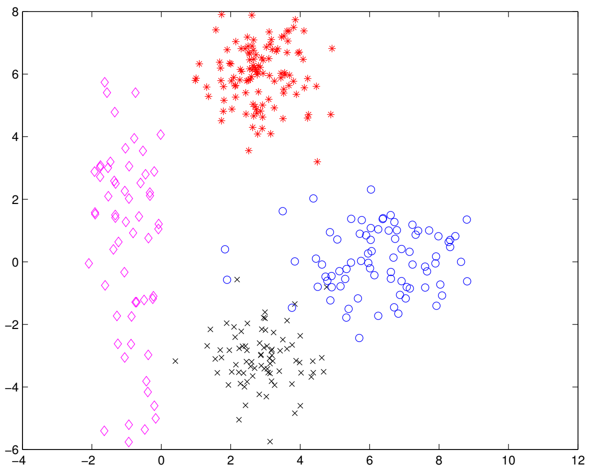


Fig. 3.8: Example data used to train mixture model.

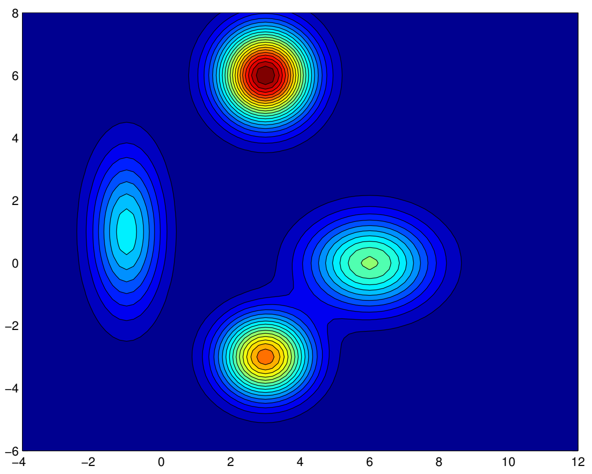


Fig. 3.9: Actual probability density for mixture model example.

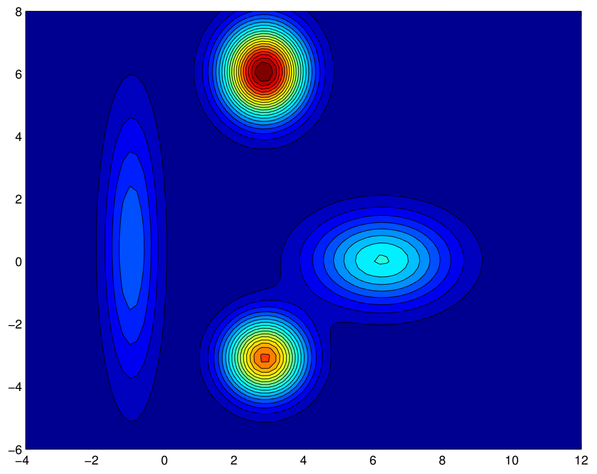


Fig. 3.10: Learned probability density for mixture model example.

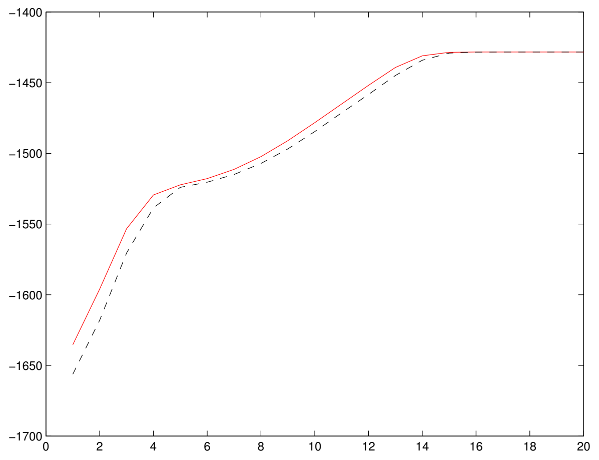


Fig. 3.11: Evolution of likelihood (solid, red line) and likelihood lower bound (dashed, black line) versus the number of training iterations for the Gaussian mixture example.

Chapter 4

Latent Process Decomposition

4.1 Introduction

In this chapter, we will introduce a new method named Latent Process Decomposition (LPD) - a generative probabilistic model for the analysis of microarray data. We will demonstrate the power of LPD on several different microarray datasets. This work is currently under submission [64] and was presented as a poster at RECOMB 2004 [63].

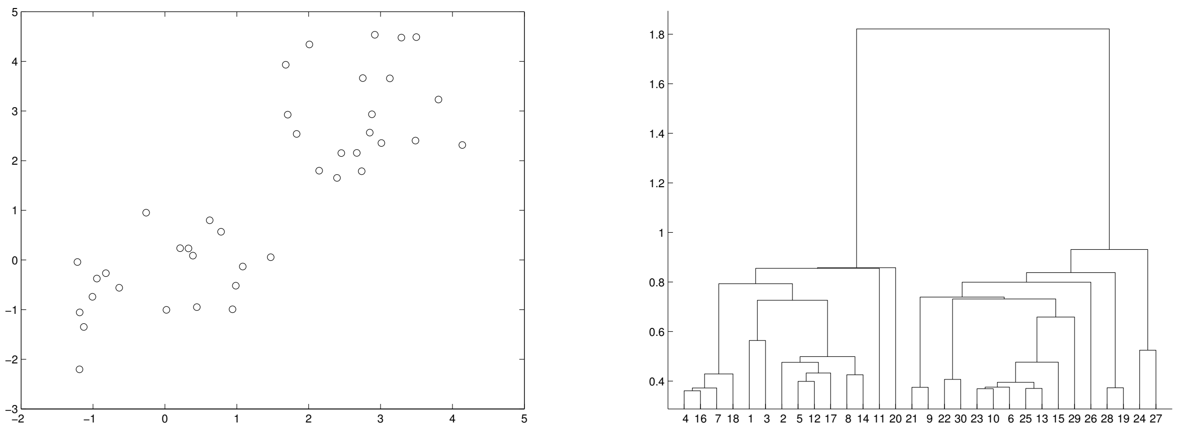
4.2 Motivation

One of the main aims of microarray data analysis is clustering similar genes and/or samples. There are many reasons for doing this. Finding clusters of similar expression in the same tumour type can lead to the discovery of important sub-classes within tumours. For example, Alizadeh et al [4] analysed lymphoma tissue samples and found evidence for two previously unrecognised sub-types of the disease. Similarly, Bhattacharjee et al [6] identified sub-types of lung cancer and Sorlie et al [70] discovered sub-types in breast cancer. These sub-classes may correspond to new, previously un-seen diseases or correlate with survival statistics or reaction to treatment. Hence, discovering these classes and the genes that are differentially expressed across them can provide excellent pointers for future research directions. On the other hand, clustering together similar genes also has potential benefits. Many known genes still have unknown function and by clustering together similar genes it is possible to infer function by association, i.e. by the function of other genes expressing similarly. Also, genes expressing similarly over a range of experimental conditions are likely to be regulated by the same parent genes and such groups of genes may be useful in inferring regulatory networks.

Probably the most popular technique for performing such cluster analysis is the use of hierarchical clustering, for example, see [16]. This approach recursively joins the most similar data points until only one point remains and the resulting connections are illustrated using a dendrogram as can be seen in figure 4.1(b). The points are indexed along the x-axis whilst the y-axis gives the measure of similarity, the higher the value, the less similar the data. Looking at this dendrogram, it would seem reasonable to make the assumption that 2 clusters exist in the data as we have two distinct sets of points with small intra-cluster distance and large inter-cluster distance. Effectively, what we have done to draw this conclusion is horizontally cut the dendrogram at a particular height leaving us with a number of disjoint point sets. Therefore, we could make the cut at any height and end up with any number of clusters between 1 and the number of data points. In this example, it is reasonably obvious from the dendrogram that a cut should be made giving two clusters, but with real data this will not be so easy. This slightly subjective method for determining the number of clusters in a dataset is one of the weaknesses of methods such as hierarchical clustering.

Another weakness of standard cluster analysis techniques is the fact that points are generally assumed to belong to only one cluster. In the case of clustering genes this is likely to be a poor assumption. Often genes belong to many functional classes and in a time-series experiment, the effects of these different functions may be visible at different stages. Therefore, restricting genes to only one class may make the clustering more straightforward but ignores the intricate nature of the underlying system. When clustering samples, it is also unreasonable to assume one cluster per sample. Take, for example, the task of clustering different tumour samples. The tumours are dynamic entities, they evolve with time and defining hard boundaries between different tumour types or stages is unrealistic.

One final criticism of standard cluster analysis techniques is their lack of transparency. In our simple example above, our data is just two dimensional and hence we can learn



(a) Data (b) Dendrogram

Fig. 4.1: Example dendrogram and data used to create it

a lot from it by simply visualising it. From 4.1(a), it is obvious that there are two clusters and also that both features can discriminate reasonably well between them. In a typical microarray experiment, we may have a few thousand features and hence visualisation (at least in the original space) is not possible. Also, many of the features will be noisy and not participate in our clustering. The information regarding which features are causing the observed clustering cannot be directly extracted from a dendrogram. Often this means that the user has to resort to ad hoc techniques such as aligning together clusters of samples and genes by eye.

Some attempts have previously been made to overcome these shortcomings. For example, Martoglio et al [47] used a variational independent components analysis approach to analyse ovarian cancer data. Rather than assigning each gene to one gene cluster, genes were allowed to participate in many different expression profiles with varying degrees. MacLachlan et al [48] introduced a mixture of factor analysers approach to clustering (in the sample space). However, due to the much larger number of features than samples, quite extreme pre-processing was required to reduce the dimensionality. Lazzeroni and Owen's Plaid model [42] is one of the best known attempts at a more flexible clustering method. This model essentially clusters both genes and samples simultaneously and each sample/gene can belong to any number of clusters. The microarray data is assumed to be made up from a number of separate layers, where genes/samples can either belong to a layer or not. Therefore, each layer is essentially a cluster of both genes *and* samples. Segal et al [67] suggested some weaknesses with this approach and proposed a model consisting of a number of cellular processes where the measured expression value for a gene in a particular array is given as the sum of the activity level in this array of each process identified with this gene. This model has one particularly appealing property. Genes can belong to any number of clusters, including zero - i.e., it is possible to simply not classify a gene as belonging to any of the discovered processes. This model is generally considered to be the state of the art in this area and as such we will use it later as a comparison with LPD.

4.3 Latent Dirichlet Allocation

The latent process decomposition model described in the next section is inspired by the Latent Dirichlet Allocation (LDA) model proposed by Blei et al [8]. LDA was introduced as a novel model for modelling text documents. The model consists of $\mathcal{K}$ latent topics, each of which is defined by a multinomial distribution over the possible words in the vocabulary. Each document p of $\mathcal{P}$ is assumed to have been produced from a number of these different topics by the following process. Firstly, draw the multinomial distribution $\boldsymbol{\theta}$ from the the Dirichlet $\mathcal{D}(\boldsymbol{\alpha})$. This multinomial is the distribution over topics that will be used to generate this document. For each word w_{np} in this document, draw a topic k from $\boldsymbol{\theta}$ with probability θ_k and then sample the word from the kth multinomial, β_k. Using the graphical notation defined in 3.2, the LDA model is shown in figure 4.2, where $\mathcal{P}$ is the number of documents in the corpus and $\mathcal{N}$ is the number of words in each document (note that this $\mathcal{N}$ can vary across different documents).

Microarray data is continuous and so the original LDA model is unsuitable. However, the one crucial feature that we wish to keep from the LDA formulation is this repeated sampling for each feature in each data example. This means that the data examples

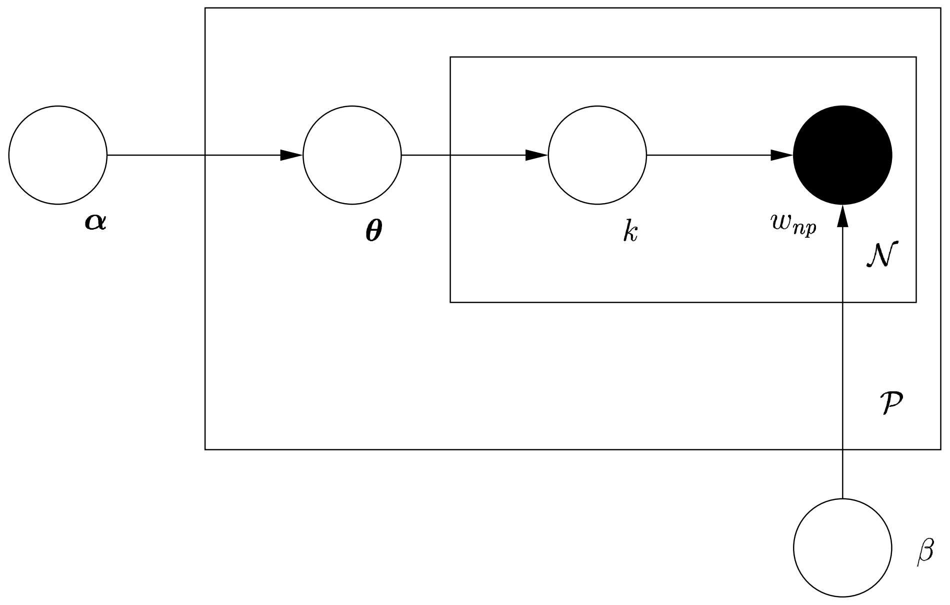


Fig. 4.2: Latent Dirichlet Allocation plates diagram

can truly belong to more than one topic (i.e. cluster). In our application, this type of re-sampling could, for example, correspond to genes performing different functions as time progresses in time series experiments.

Therefore, we need to define a new *topic* that suits continuous microarray data. Observations of the data (after it has been logged) suggests it is distributed in a roughly Gaussian manner. The Gaussian distribution is also useful as analytically it tends to be easier to handle than many other distributions - for example, it is possible to get closed form solutions for its maximum likelihood parameters - something not possible with say a gamma distribution. We therefore define our new *topic* (which we shall call a *process*) as a set of one dimensional Gaussians, one for each array (or gene, if we are performing our analysis in sample space). As with the LDA model, this means that genes/samples can belong to more than one process (i.e. cluster). The probabilistic nature of the model enables comparison between models (through likelihood values) allowing us to evaluate the most probable number of processes $\mathcal{K}$. Finally, the nature of the processes means that they can easily be compared to discover genes differentially expressed across different samples. In the following sections, we will describe the model in more detail, show how the parameter values can be learned and discuss the performance of the algorithm on several microarray datasets.

4.4 Model Description

Consider the following generative procedure for generating the expression values for a gene. Firstly, sample $\boldsymbol{\theta}$ from the Dirichlet $\mathcal{D}(\boldsymbol{\alpha})$. Then for each array $a = 1, \ldots, \mathcal{A}$ sample an element k with probability θ_k. For this array, a, sample an expression value from $\mathcal{N}(\mu_{ak}, \sigma_{ak})$. This generative process is depicted graphically in figure 4.3. As can be seen from the figure, the only major difference between LPD and LDA is in the process definitions which in this case are the Gaussian means and variances and for LDA were the multinomials. It should also be noted that we are being slightly liberal with our graphical representation. Strictly speaking, the model parameters $\boldsymbol{\sigma}$, $\boldsymbol{\alpha}$ and $\boldsymbol{\mu}$ are random variables and we should define prior distributions over their values (for example, the process mean vectors may be considered as $\mathcal{K}$ samples from an $\mathcal{A}$ dimensional Gaussian).

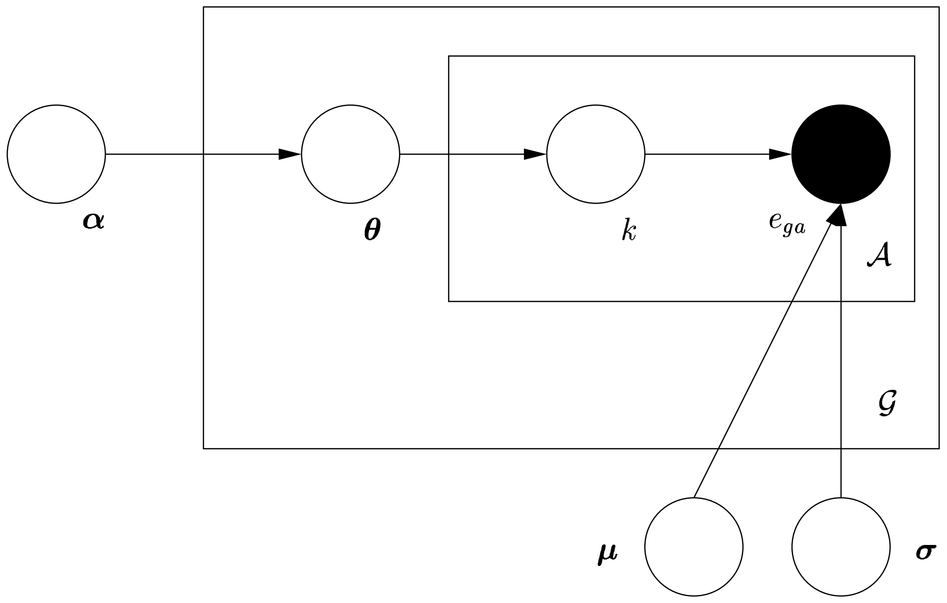


Fig. 4.3: Latent Process Decomposition plates diagram

We have discussed the generative procedure in gene space. That is, we sample from the Dirichlet for each gene and then sample $\mathcal{A}$ expression values. Therefore, this model is comparable to clustering together similar genes based on their expression profiles over the different arrays. We can also switch the model around by sampling from the Dirichlet for each array and then sampling $\mathcal{G}$ expression values. This is comparable to clustering together cell samples based on their expression profiles over the genes. Both methods have potential uses and later we will present results for both.

4.5 Learning

4.5.1 Uniform Priors - Maximum Likelihood

Unlike a more standard machine learning task where we might wish to create a model that can be used to predict some quantity or event given some input data, we are more interested here in analysing and extracting information from the available data. Therefore, the normal concern of over-fitting is not really appropriate and we can be satisfied with a straightforward maximum likelihood or MAP approximation to parameter values. Recall that for a set of parameters, $\mathcal{H}$ and data $\mathcal{X}$, Bayes rule tells us that

$$P(\mathcal{H}|\mathcal{X}) \propto P(\mathcal{X}|\mathcal{H})P(\mathcal{H}) \tag{4.1}$$

where $P(\mathcal{X}|\mathcal{H})$ is known as the *likelihood* and $P(\mathcal{H})$ the *prior* on our parameters $\mathcal{H}$. We are interested in finding the set of parameters $\mathcal{H}$ that maximises $P(\mathcal{H}|\mathcal{X})$ (the MAP solution). In the case of a uniform (and hence uninformative) prior, this is the maximum likelihood solution. We will begin by deriving the maximum likelihood and then extend this to the MAP solution. The expression we are interested in maximising is

$$\log p(\mathcal{G}|\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\alpha}) \tag{4.2}$$

(note, that as a shorthand, we have used $\mathcal{G}$ to represent the dataset consisting of the expression for all of the genes *and* the total number of genes in summations etc - it should be obvious from the context which applies) which is factorised over the individual genes as follows (note, in all derivations, we will assume the analysis is being done in gene space - the conversion to the sample space essentially involves swapping the a and g indices)

$$\log p(\mathcal{G}|\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\alpha}) = \sum_{g=1}^{\mathcal{G}} \log p(g|\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\alpha}). \tag{4.3}$$

Marginalising over the latent variable $\boldsymbol{\theta}$ allows us to expand this expression as follows

$$\log p(\mathcal{G}|\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\alpha}) = \sum_{g=1}^{\mathcal{G}} \log \int_{\Theta} p(g|\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\theta}) p(\boldsymbol{\theta}|\boldsymbol{\alpha}) d\boldsymbol{\theta}. \tag{4.4}$$

Using Jensen's inequality as described in section 3.3, if we introduce a set of gene specific variational distributions $\boldsymbol{\gamma}_g$, we can lower bound this likelihood as follows

$$\begin{aligned}
\log p(\mathcal{G}|\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\alpha}) &= \sum_{g=1}^{\mathcal{G}} \log \int_{\Theta} p(g|\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\theta}) p(\boldsymbol{\theta}|\boldsymbol{\alpha}) d\boldsymbol{\theta} && (4.5)\\
&= \sum_{g=1}^{\mathcal{G}} \log \int_{\Theta} p(g|\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\theta}) \frac{p(\boldsymbol{\theta}|\boldsymbol{\alpha}) p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)}{p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)} d\boldsymbol{\theta} && (4.6)\\
&\geq \sum_{g=1}^{\mathcal{G}} \int_{\Theta} p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) \log \left\{ p(g|\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\theta}) \frac{p(\boldsymbol{\theta}|\boldsymbol{\alpha})}{p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)} \right\} d\boldsymbol{\theta} && (4.7)\\
&= \mathcal{L}'(\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\alpha},\boldsymbol{\gamma}). && (4.8)
\end{aligned}$$

We also have the following conditional independence over the arrays

$$p(g|\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\theta}) = \prod_{a=1}^{\mathcal{A}} p(e_{ga}|\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\theta}) \tag{4.9}$$

where e_{ga} is the observed expression level of gene g in array a. Substituting this into our expression for $\mathcal{L}'$ gives

$$\mathcal{L}'(\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\alpha},\boldsymbol{\gamma}) = \sum_{g=1}^{\mathcal{G}} \int_{\Theta} p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) \log \left\{ \prod_{a=1}^{\mathcal{A}} p(e_{ga}|\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\theta}) \frac{p(\boldsymbol{\theta}|\boldsymbol{\alpha})}{p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)} \right\} d\boldsymbol{\theta} \tag{4.10}$$

which can be marginalised over our second latent variable k as follows

$$\mathcal{L}'(\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\alpha},\boldsymbol{\gamma}) = \sum_{g=1}^{\mathcal{G}} \int_{\Theta} p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) \log \left\{ \prod_{a=1}^{\mathcal{A}} \left[\sum_{k=1}^{\mathcal{K}} p(e_{ga}|\mu_{ak},\sigma_{ak})\theta_k \right] \frac{p(\boldsymbol{\theta}|\boldsymbol{\alpha})}{p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)} \right\} d\boldsymbol{\theta}. \quad (4.11)$$

Introducing a second set of variational distributions, Q_{agk} subject to $\sum_k Q_{agk} = 1$ we can further bound this expression to move the log inside the summation

$$\begin{aligned}
\mathcal{L}'(\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\alpha},\boldsymbol{\gamma}) &= \sum_{g=1}^{\mathcal{G}} \int_{\Theta} p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) \log \left\{ \prod_{a=1}^{\mathcal{A}} \left[\sum_{k=1}^{\mathcal{K}} p(e_{ga}|\mu_{ak},\sigma_{ak}) \frac{\theta_k Q_{agk}}{Q_{agk}} \right] \frac{p(\boldsymbol{\theta}|\boldsymbol{\alpha})}{p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)} \right\} d\boldsymbol{\theta} \\
&\geq \sum_{g=1}^{\mathcal{G}} \int_{\Theta} \sum_{a=1}^{\mathcal{A}} \sum_{k=1}^{\mathcal{K}} p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) Q_{agk} \log \left\{ p(e_{ga}|\mu_{ak},\sigma_{ak}) \frac{\theta_k}{Q_{agk}} \frac{p(\boldsymbol{\theta}|\boldsymbol{\alpha})}{p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)} \right\} d\boldsymbol{\theta} \\
&= \mathcal{L}''(\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\alpha},\boldsymbol{\gamma},\mathbf{Q}).
\end{aligned}$$

These variational multinomials may be thought of as the posterior probability of the expression value for gene g in array a having been produced by process k.

Expanding the log term and removing unnecessary summations and integrals, we can write this bound out fully as

$$\begin{aligned}
\sum_{g=1}^{\mathcal{G}} \log p(g|\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\alpha}) \geq\; & \sum_{g=1}^{\mathcal{G}} \int_{\Theta} p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) \log p(\boldsymbol{\theta}|\boldsymbol{\alpha}) d\boldsymbol{\theta} \\
& - \sum_{g=1}^{\mathcal{G}} \int_{\Theta} p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) \log p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) d\boldsymbol{\theta} \\
& + \sum_{g=1}^{\mathcal{G}} \sum_{a=1}^{\mathcal{A}} \sum_{k=1}^{\mathcal{K}} Q_{agk} \log p(e_{ga}|\mu_{ak},\sigma_{ak}) \\
& + \sum_{g=1}^{\mathcal{G}} \sum_{a=1}^{\mathcal{A}} \sum_{k=1}^{\mathcal{K}} \int_{\Theta} Q_{agk} p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) \log \theta_k d\boldsymbol{\theta} \\
& - \sum_{g=1}^{\mathcal{G}} \sum_{a=1}^{\mathcal{A}} \sum_{k=1}^{\mathcal{K}} Q_{agk} \log Q_{agk}. \qquad (4.12)
\end{aligned}$$

As described in section 3.3, we can take partial derivatives of this bound with respect to our model and variational parameters to give update equations guaranteed to increase the likelihood. Derivation details are given in appendix C and they result in the following set of updates for our variational and model parameters

$$Q_{agk} = \frac{p(e_{ga}|\mu_{ak},\sigma_{ak})\exp\{\mathbf{E}_{p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)}[\log\theta_k]\}}{\sum_{k=1}^{\mathcal{K}} p(e_{ga}|\mu_{ak'},\sigma_{ak'})\exp\{\mathbf{E}_{p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)}[\log\theta_{k'}]\}} \tag{4.13}$$

$$\gamma_{gk} = \alpha_k + \sum_{a=1}^{\mathcal{A}} Q_{agk} \tag{4.14}$$

$$\mu_{ak} = \frac{\sum_{g=1}^{\mathcal{G}} Q_{agk} e_{ga}}{\sum_{g=1}^{\mathcal{G}} Q_{agk}} \tag{4.15}$$

$$\sigma_{ak}^2 = \frac{\sum_{g=1}^{\mathcal{G}} Q_{agk}(e_{ga}-\mu_{ak})^2}{\sum_{g=1}^{\mathcal{G}} Q_{agk}} \tag{4.16}$$

where

$$\begin{aligned}\mathbf{E}_{p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)}[\log\theta_k] &= \int_{\Theta} p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)\log\theta_k d\boldsymbol{\theta} \\ &= \Psi(\gamma_{gk}) - \Psi(\sum_j \gamma_{gj}).\end{aligned}$$

The update for $\boldsymbol{\alpha}$ does not have a convenient closed form solution and so second order methods are required. Details of this procedure are given in appendix C.2.3.

4.5.2 Informative Priors - MAP

In the previous section, we derived a series of update equations, guaranteed to find at least a local maximum of the likelihood. We will now extend this result to the MAP solution by incorporating informative priors. For example, a suitable prior on our means may be a zero mean Gaussian. This reflects our belief that most genes will be uninformative and will have expression values around zero. For the variance, we may wish to define a prior that helps us to avoid over-fitting. Over-fitting may occur when Gaussians contract onto a single data point causing poor generalisation in addition to numerical instability. With sensible choices of prior, extension of our previous derivation to the MAP solution is straightforward. Our combined likelihood and prior expression is (assuming a uniform prior on $\boldsymbol{\alpha}$)

$$p(\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\alpha} | \mathcal{G}) \propto p(\mathcal{G} | \boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\alpha}) p(\boldsymbol{\mu}) p(\boldsymbol{\sigma}). \tag{4.17}$$

Taking logarithms of the right hand side, we can formulate our problem as

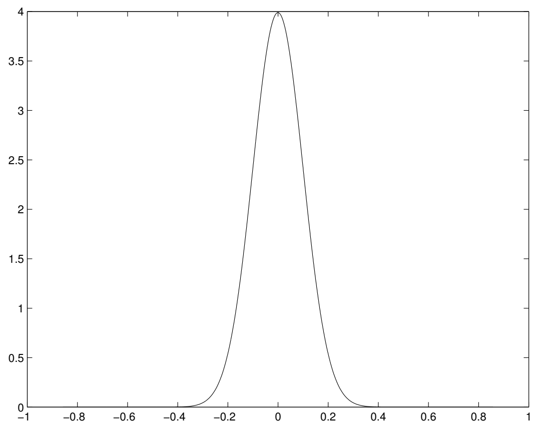
$$\boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\mu} = \arg \max_{\boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\mu}} \quad \log p(\mathcal{G} | \boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\alpha}) + \log p(\boldsymbol{\mu}) + \log p(\boldsymbol{\sigma}). \tag{4.18}$$

Therefore, we can simply append these two additional terms onto our previous bound. Noting that they are functions only of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ (and any associated hyper-parameters), we can conclude that they will only change the updates for μ_{ak} and σ_{ak}. Let us assume the following priors

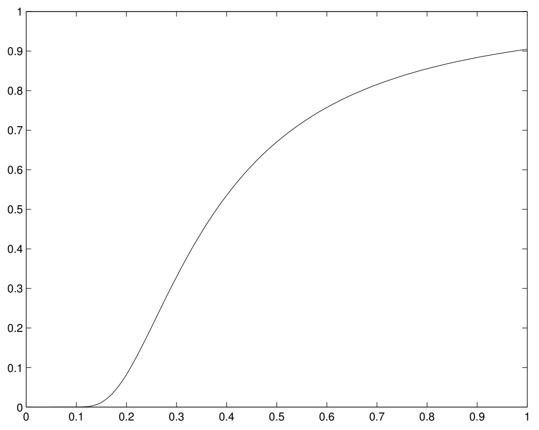
$$p(\mu_{ak}) \propto \mathcal{N}(0, \sigma_\mu) \tag{4.19}$$
$$p(\sigma_{ak}^2) \propto \exp\left\{ -\frac{s}{\sigma_{ak}^2} \right\} \tag{4.20}$$

i.e. a zero-mean Gaussian for the means and an improper prior (i.e. it doesn't integrate to 1 over its domain) for σ_{ak}^2 that gives zero weight to $\sigma_{ak}^2 = 0$ and asymptotically approaches 1 as σ_{ak} approaches infinity. Plots of these priors can be seen in figure 4.4 where $\sigma_\mu = 0.1$ and $s = 0.1$. Adding these to our original bound, we now have to differentiate the following expression to obtain updates for μ_{ak} and σ_{ak}

$$\sum_{g=1}^{\mathcal{G}} Q_{agk} \log p(e_{ga} | \mu_{ak}, \sigma_{ak}) + \log p(\mu_{ak}) + \log p(\sigma_{ak}^2). \tag{4.21}$$



(a) Prior for μ_{ak}



(b) Prior for σ^2_{ak}

Fig. 4.4: Example priors for LPD μ_{ak} and σ_{ak} parameters

Performing the necessary differentiations, we are left with the following new updates

$$\mu_{ak} = \frac{\sigma^2_\mu \sum_{g=1}^{\mathcal{G}} Q_{agk} e_{ga}}{\sigma^2_{ak} + \sigma^2_\mu \sum_{g=1}^{\mathcal{G}} Q_{akg}} \tag{4.22}$$

$$\sigma^2_{ak} = \frac{\sum_{g=1}^{\mathcal{G}} Q_{agk}(e_{ga} - \mu_{ak})^2 + 2s}{\sum_{g=1}^{\mathcal{G}} Q_{agk}}. \tag{4.23}$$

These expressions clearly show the effect of the priors. In 4.22, we see that we now have a dependency on both σ^2_{ak} and σ^2_μ. In the limit of $\sigma^2_\mu \to \infty$, we effectively have a uniform prior and we recover exactly our original update. In the opposite limit of $\sigma^2_\mu \to 0$, we find that $\mu_{ak} \to 0$ as we would expect. In 4.23, the prior is effectively putting a lower bound on the value of σ^2_{ak}. As $s \to 0$ the prior approaches a uniform distribution over all positive reals and we recover our original σ^2_{ak} update.

4.6 Likelihood Computation

To compare performance between models and to assess the convergence of a given model, it is important to be able to compute the likelihood $p(\mathcal{G}|\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\alpha})$. In order to do this, we need to marginalise over our latent variables k and $\boldsymbol{\theta}$ i.e., we need to calculate

$$\log p(\mathcal{G}|\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\alpha}) = \sum_{g=1}^{\mathcal{G}} \log \int_{\Theta} \left[\prod_{a=1}^{\mathcal{A}} \sum_{k=1}^{\mathcal{K}} p(e_{ga}|\mu_{ak}, \sigma_{ak})\theta_k \right] p(\boldsymbol{\theta}|\boldsymbol{\alpha}) d\boldsymbol{\theta}. \tag{4.24}$$

Unfortunately, the calculation of the integral is not analytically tractable and so we are forced to resort to an approximation technique. Many methods exist for approximating expectations such as this. If sampling from the respective distribution is straightforward, we can simply take a large number of samples and approximate our expectation with the mean value using these samples. If sampling is not straightforward, methods such as importance sampling and rejection sampling can be used to approximate the expectation. Markov-chain Monte-Carlo approaches (for example, the Metropolis-Hastings algorithm) can also be used to generate samples that can then be used to empirically approximate the expectation.

In this case, we require samples from a Dirichlet, $p(\boldsymbol{\theta}|\boldsymbol{\alpha})$. It is possible to sample from such a distribution by sampling from k independent gamma distributions with parameters α_k and then normalising the result. Hence we can use the first method described above.

Let θ_k^n denote the kth component of the nth sample of $\boldsymbol{\theta}$ from $p(\boldsymbol{\theta}|\boldsymbol{\alpha})$. If we take N samples in total, we can approximate our likelihood as follows

$$\mathcal{L} \sim \sum_{g=1}^{\mathcal{G}} \log \frac{1}{N} \sum_{n=1}^{\mathcal{N}} \left[\prod_{a=1}^{\mathcal{A}} \sum_{k=1}^{\mathcal{K}} p(e_{ga}|\mu_{ak}, \sigma_{ak})\theta_k^n \right]. \tag{4.25}$$

In our investigations we found that of the order of 1000 samples gave a very stable likelihood value.

4.7 Algorithm and Implementation Details

4.7.1 Parameter Initialisation

From 4.13 we can see that in order to update Q_{agk} we need values of μ_{ak}, σ_{ak} and γ_{gk}. The former two quantities, we initialise to the array means and standard deviations respectively. The γ_{gk} values were initialised to positive random numbers. With the means and standard deviations uniform over the k processes, if the γ_{gk} values were also uniform, the algorithm would be immediately caught in a local maximum and no learning would take place.

4.7.2 Numerical Stability

There are several situations which could cause the algorithm to lose numerical stability. Firstly, the problem of over-fitting. If a Gaussian contracts onto a single point, its standard deviation will shrink to zero and its p.d.f. will approach ∞. To get around this, we apply a lower bound to the value of σ^2_{ak} (generally, this will not be a problem if a MAP solution is used and a suitable prior has been placed on σ^2_{ak}). The choice of the value of this bound is reasonably important as too low a value and numerical problems will persist and too high a value will over-constrain the model and prevent it from fitting to the data. Secondly, in the update for Q_{agk} (4.13) we occasionally find that we begin to reach the limits of the machines precision and both the numerator and the denominator approach zero. To get around this, we add a small constant (10^{-200}) to each value in $\mathbf{Q}$ before normalisation (i.e., before performing the division in 4.13). Finally, when calculating the likelihood, we have to compute a product over the arrays a. For a large number of arrays, this soon becomes smaller than the machine precision. However, we are ultimately interested in the logarithm of this product and so we can overcome this problem with a standard numerical technique, described in appendix F.

4.7.3 Decomposing over genes or arrays?

All of the derivations thus far have concerned decompositions over genes. That is, we are interested in clustering together genes. This type of analysis is sensible where we have data from some form time series experiment (for example, the Spellman yeast data [71] that we look at below). However, in the cancer examples discussed below, we are interested in decomposing over arrays - i.e. being able to distinguish between cancer sub-types and identifying genes that are differentially expressed across these types. It is trivial to switch between these two different forms of analysis. Practically, we simply transpose our data matrix. Mathematically, we swap the gene and array indexes.

4.7.4 Algorithm Pseudo code

```
Data: Read the data $\mathbf{e}$
$\boldsymbol{\alpha} \longleftarrow 1$
Initialise $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$
$\gamma \longleftarrow rand()$
while $iterations \leq maxIterations$ do
    Update: $\boldsymbol{Q}$ as in equation (4.13)
    Update: $\boldsymbol{\gamma}$ as in equation (4.14)
    Update: $\boldsymbol{\mu}$ as in equation (4.15 or 4.22)
    Update: $\boldsymbol{\sigma}^2$ as in equation (4.16 or 4.23)
    if $\sigma^2 < tol_1$ then
        $\sigma^2 = tol_1$
    end
    while $|\alpha_{new} - \alpha_{old}| > tol_2$ do
        $\alpha_{new} \longleftarrow \alpha_{old} - H(\alpha_{old})^{-1} g(\alpha_{old})$
        where $g$ and $H$ are given in appendix C.2.3
    end
    Determine the likelihood (see below)
end
```

Algorithm 1: Latent Process Decomposition algorithm. We used a tolerance $tol_1 = 10^{-5}$ to avoid Gaussians collapsing onto a single point, and $tol_2 = 10^{-10}$ for the $\boldsymbol{\alpha}$ updating.

```
Given: e, $\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\alpha}$
Take $N$ samples $\theta^n$ from the Dirichlet distribution $p(\boldsymbol{\theta}|\boldsymbol{\alpha})$
Determine the likelihood from equation 4.25
```

Algorithm 2: Determining the likelihood for LPD

4.8 Experimental Results

We will illustrate the performance of LPD on two forms of microarray dataset. The first are cancer datasets and we are interested in the ability of LPD to distinguish between various tissue classes. The second dataset types are time series datasets for yeast (*Saccharomyces cerevisiae*) and we are interested in finding processes that correspond to known biological processes.

4.8.1 Prostate

Our first cancer dataset comes from a study of prostate cancer by Dhanasakera et al [13] and consists of 54 samples with 9984 features (we used their complete dataset (i.e., with samples from all sources), corresponding to fig 8 in the supplementary information of [13]). The data consists of 14 samples for benign prostatic hyperplasia (BPH) - a non cancerous swelling of the prostate, 3 normal adjacent prostate (NAP), 1 normal adjacent tumour (NAT), 14 localised prostate cancer (PCA), 1 prostatitis (PRO) (inflammation of the prostate) and 20 metastatic tumours (MET). The ability to be able to distinguish between local and metastatic tumours is an important one and therefore discovery of genes that are expressed differently across them is crucial. The inclusion of BPH samples in this dataset is also worth mentioning. It is a difficult task to distinguish between (non-cancerous) BPH and prostate cancer (PCA) but it is obviously crucial in the treatment required. Using this dataset, we will illustrate the difference in performance between uniform and informative priors and compare LPD to hierarchical clustering.

In figure 4.5 we can see how the held-out likelihood (in all experiments, these values are the average of 10-fold cross validation and bars show standard error) changes as

$\mathcal{K}$, the number of processes is increased, when we train with the maximum likelihood technique. We see that the likelihood peaks at $\mathcal{K} = 4$ processes. Figure 4.6 shows a typical decomposition of the prostate data into four processes. Each column represents an array and each row a process. The bars correspond to the normalised $\boldsymbol{\gamma}$ values and represent the *responsibility* of a process to that particular array. These can be thought of as analogous to cluster memberships. Remember that we are decomposing over arrays and not genes and so each column in figure 4.6 corresponds to an array. From figure 4.5 we can see that there is some distinction between the metastatic samples (process 4) and the rest (processes 1, 2 and 3), however it is far from clear and hence would not be terribly useful if the labels were not known! Figure 4.7 shows the result of clustering this data with a dendrogram, using average linkage and and the euclidean distance measure. From the dendrogram, we can see that although some local structure has been identified (i.e, a lot of adjacent samples are of the same type) globally, the separation between the known classes is poor.

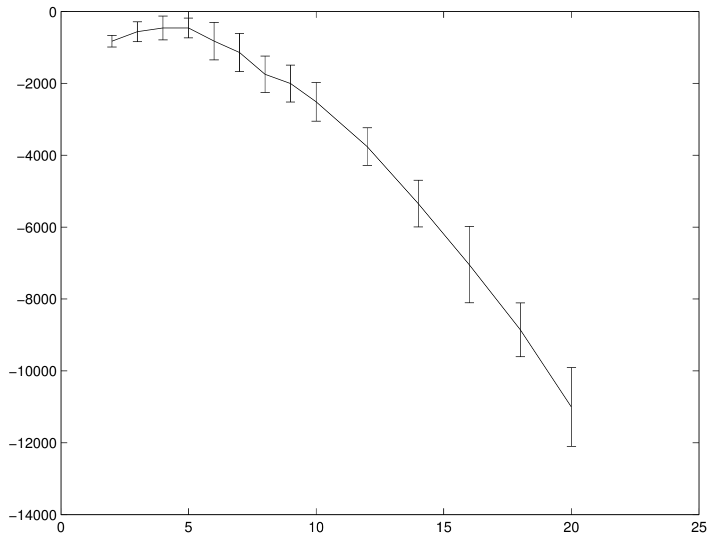


Fig. 4.5: Held-out likelihood (x-axis) versus the $\mathcal{K}$ for the prostate dataset with no parameter priors.

This poor performance may be due to the fact that we have used all 9,984 genes in the original dataset. It is fair to assume that even though the condition of the tissue

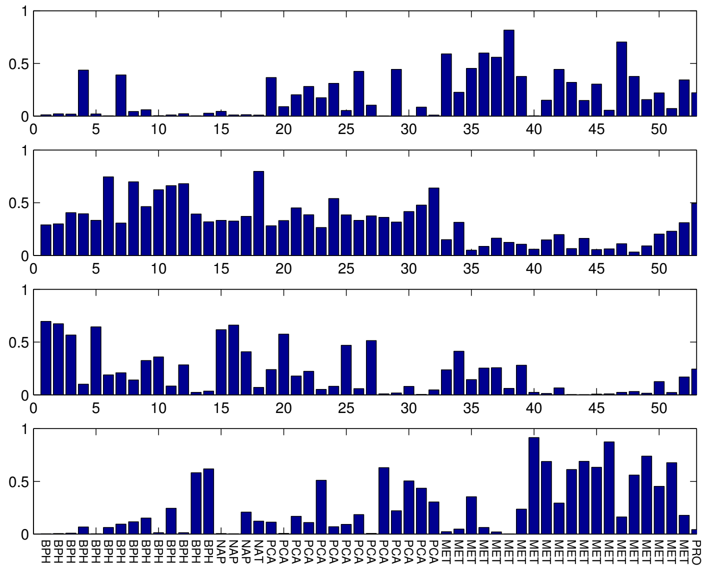


Fig. 4.6: $\mathcal{K} = 4$ process decomposition of the prostate dataset. Each column corresponds to a sample whilst each row corresponds to a process. Bars are $\mathbf{E}_{p(\boldsymbol{\theta}|\boldsymbol{\gamma}_a)}[\theta_k]$ - and reflect the *responsibility* of that process to that particular array.

from which the samples were taken is quite variable across the sample types, the actual number of genes that will be differentially expressed is only going to be a small fraction. Therefore, it is likely that the interesting differences are being swamped by random (noise induced) differences in the remainder of the dataset. The most simple way of overcoming this problem is to rank the genes based on their variance across samples and use just those with the highest variance. This may seem a rather naive approach but remember that we cannot use our prior class knowledge and the only real belief we have is that genes varying only slightly across samples are unlikely to be interesting. Inclusion of genes with only small variations may well be detrimental to analysis as variations are just as likely to be due to noise as they are actual biological effects. The cross validation plot using just the 500 most varying genes can be seen in figure 4.8. We can see from this plot that there appears to be a peak at $\mathcal{K} = 4$ processes and a decomposition into $\mathcal{K} = 4$ processes can be seen in figure 4.9. The LPD has now managed to distinguish well between the four major classes. We can see from 4.9 that the BPH samples are concentrated in process 1 along with the prostatitis, the local tumours (PCA) in process 2 and the metastatic samples in processes 3 and 4. Figure

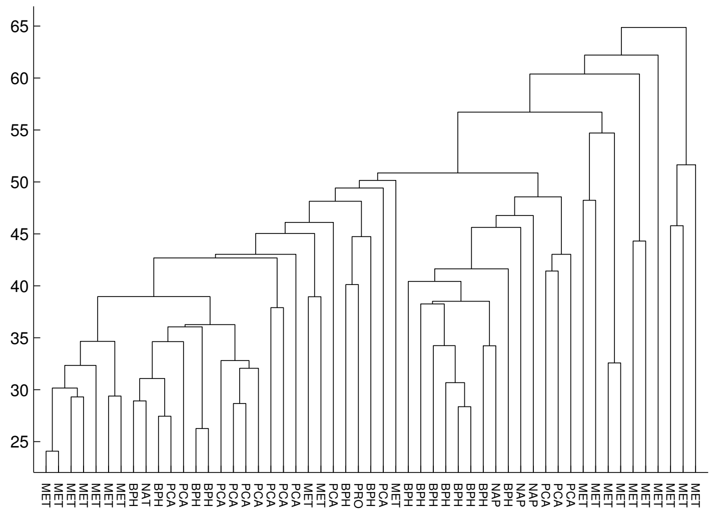


Fig. 4.7: Result of hierarchical clustering on all prostate data, using average linkage and euclidean distance.

4.10 shows the result of performing hierarchical clustering on this reduced data set (again with average linkage and using euclidean distance). Although the dendrogram manages to split metastatic from the rest successfully, the difference between PCA and BPH is still very unclear and it is very difficult to infer from this dendrogram just how many clusters are present.

The other advantage of LPD over standard clustering techniques is the ease at which information can be extracted from the learnt model. In this prostate example, we may be interested in finding out which genes are most differentially expressed between the local cancer (PCA) and the metastatic samples. The following score can be used to rank genes under this criteria

$$Z = \frac{|\mu_{gi} - \mu_{gj}|}{\sqrt{\sigma_{gi}^2 + \sigma_{gj}^2}} \tag{4.26}$$

where i and j correspond to the two processes being compared. Genes ranking highly will be those with large separation between their means and small standard deviations - corresponding to genes whose values fall into two disjoint clusters. A list of the top 10 ranking genes based on this score can be seen in table 4.1. Plots of the mixture densities defined by the latent processes and the actual data values for three of these genes can

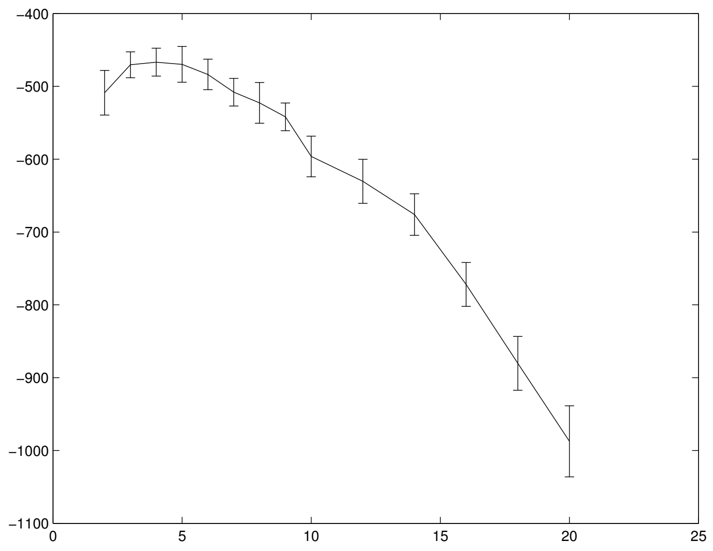


Fig. 4.8: Held-out likelihood (x-axis) versus the $\mathcal{K}$ for the top 500 genes in the prostate dataset with no parameter priors.

be seen in figure 4.11. These three plots show that the extracted genes appear to be able to distinguish well between local cancer and metastasis and importantly, have been highlighted as significant without the use of the class labels. Investigations into these genes (Dr. Jeremy Clark, private correspondence) show that they are interesting candidates for future examination. For example, CTGF is a gene that is induced by growth-factors or certain oncogenes. Its respective protein binds to integrins and modulates the invasive behaviour of certain human cancer cells [9]. Similarly, it has been reported [18] that tumour angiogenesis [1] and growth are critically dependent on the activity of EGR1.

[1]the creation of blood vessels

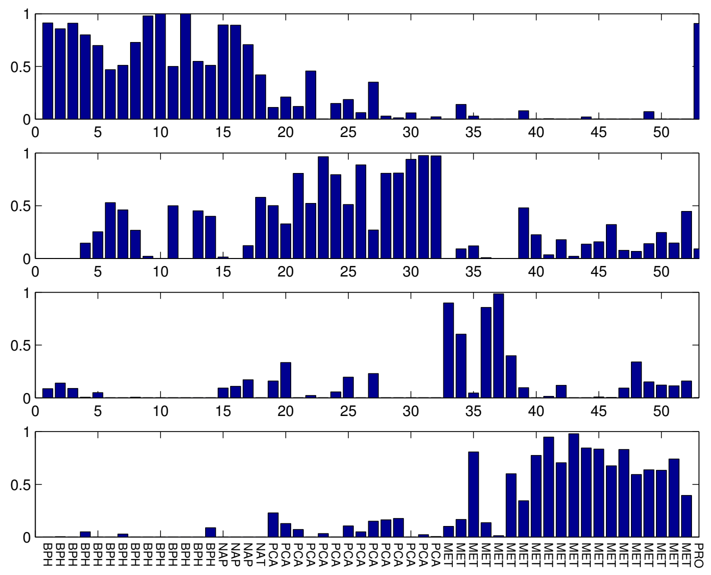


Fig. 4.9: Normalised γ_{ak} values for the prostate dataset. Using the 500 most varying genes and no parameter priors.

Rank	Name	Z
1	translocating chain-associating membrane protein (TRAM)	4.243369
2	early growth response 1 (EGR1)	4.131861
3	connective tissue growth factor (CTGF)	3.579336
4	osteoglycin (osteoinductive factor, mimecan) (OGN)	3.339221
5	golgi membrane protein (GP73)	3.202468
6	butyrate response factor 1 (EGF-response factor 1) (BRF1)	3.074948
7	ribosomal protein L5 (RPL5)	2.985735
8	dopachrome delta-isomerase, tyrosine-related protein 2 (DCT)	2.929469
9	pericentriolar material 1 (PCM1)	2.877109
10	sprouty (Drosophila) homolog 4	2.755093

Tab. 4.1: Top 10 genes separating localised prostate from metastatic prostate cancer. Genes are ranked by Z.

It is obvious from 4.8 that as we increase the number of processes, the model begins to over-fit quite dramatically to the training data. This is to be expected with the maximum likelihood solution as we are allowing the model too much freedom by not

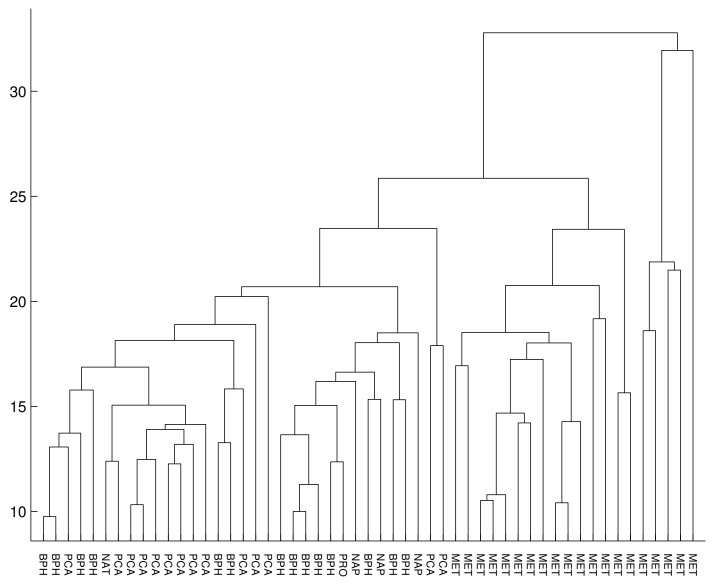
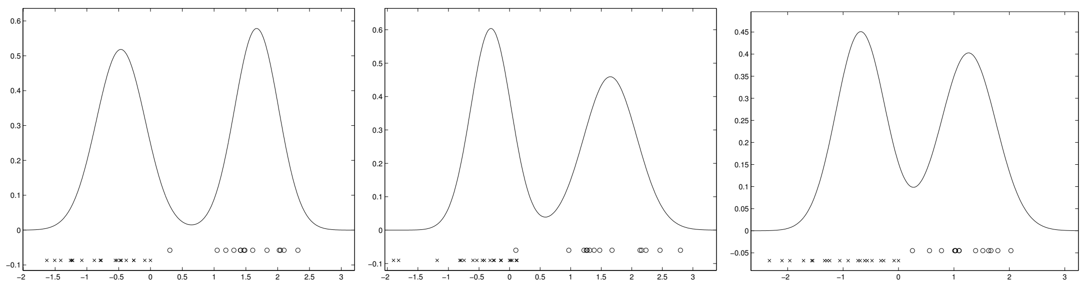


Fig. 4.10: Result of hierarchical clustering on top 500 prostate genes, using average linkage and euclidean distance.

penalising complexity. To overcome this, we can incorporate the priors discussed in section 4.5.2. Using values of $\sigma_\mu^2 = 0.1$ and $s = 0.01$, we get the likelihood versus $\mathcal{K}$ plot given in figure 4.12. We see from this plot that rather than fall away as the number of processes increases, the likelihood remains reasonably constant. Figure 4.13 provides an explanation for this behaviour. This is a decomposition of the prostate data (top 500 genes) into $\mathcal{K} = 10$ processes with no parameter priors. Given that the estimated optimum number of processes is 4, we might expect the model to severely over-fit as indeed it does. However, if we introduce priors with $\sigma_\mu^2 = 0.1$ and $s = 0.5$ we obtain the decomposition seen in figure 4.14. We observe that the decomposition is over just four processes, with the remaining six empty. This is potentially a very useful property as, given a suitable value for this prior we no longer need to decide on an exact number of processes. Also notice that the decomposition is much cleaner as a result of incorporating the prior. This suggests that the *crossover* between the processes in 4.9 is due to noise rather than interesting sub-structure. Also, note that the metastatic samples are falling across two processes. This split is due to the samples coming from two separate sources.



(a) Early growth response (EGR1) (b) Connective tissue growth factor (CTGF) (c) Dopachrome delta-isomerase, tyrosine-related protein 2 (DCT)

Fig. 4.11: Inferred density plots for 3 genes separating localised prostate cancer (PCA, process 2) from the metastatic samples (MET and M, process 4) in Figure 4.9. Actual data values are shown below the mixtures, separated by class with circles representing values from localised prostate cancer and crosses representing values from metastatic prostate cancer samples.

Although it appears that we can avoid our cross-validation over the number of processes if we have the correct prior parameters, this still leaves us with the choice of the prior parameters. Investigations show that the mean parameter has negligible effect and can be set at something in the order of 0.1. This leaves s, the variance parameter. A cross validation over s for the top 500 genes in the prostate data can be seen in figure 4.15. We notice a peak at $s = 0.1$. We cannot be too confident in this due to the large error bars but it does reasonably suggest a peak somewhere between 0.05 and 0.5. It may appear that we have gained very little in moving from the cross-validation over processes to that over s but arguably this is not the case for two reasons. Firstly, we are dealing with a small number of samples and hence by removing some samples for cross-validation, we could easily be removing a substantial portion of one particular class, suggesting a bias towards under-estimating the number of classes present. The variance parameter on the other hand is a measure of noise that should be class independent and hence should be the same regardless of whether the full data set is being used or not. Also, the number of processes present will be dependent on the particular

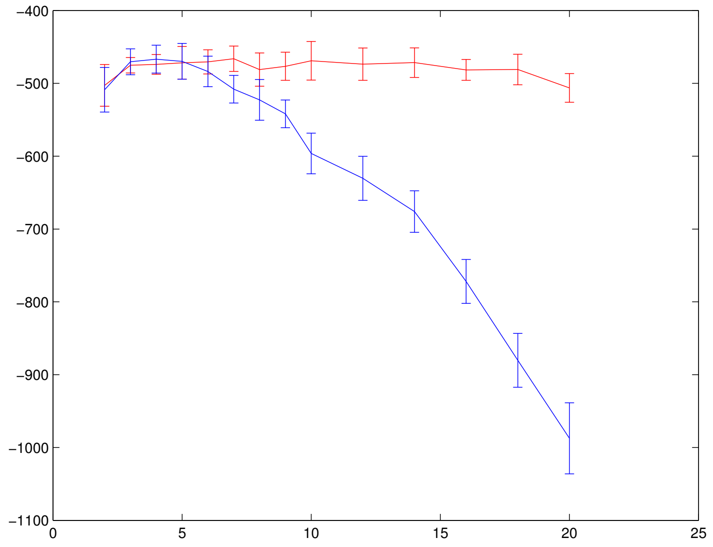


Fig. 4.12: Held-out likelihood (x-axis) versus the $\mathcal{K}$ for the top 500 genes in the prostate dataset with parameter priors as given in section 4.5.2 (red) and without (blue).

data set and could vary greatly depending on the subject of the investigation. On the other hand, the noise present is a result of the experimental procedure and we may expect the optimum value of s to vary very little across data sets. Indeed, in the following sections, we shall see that this is the case.

Finally, we have used likelihood as our measure of model performance. One issue that we have not yet explored is how do these likelihood values compare with those for different (possibly simpler) models. The most obvious model to compare with is a standard Gaussian mixture. The main difference between LPD and the Gaussian mixture is the repeated sampling for each feature in LPD allowing multiple processes to interact to produce one sample. Due to the large number of features involved, it is computationally infeasible to use a Gaussian mixture with full covariance matrices. We shall constrain the matrices to be diagonal (equivalent to Gaussians aligned with the coordinate axis). This Naive Bayes Gaussian mixture is identical to the model derived in appendix B. Figure 4.16 shows the likelihood for the top 500 prostate genes using maximum likelihood (i.e. no priors) for LPD and Naive Bayes. We see that Naive-Bayes begins over-fitting immediately, suggesting only 2 clusters in the data. LPD rises above

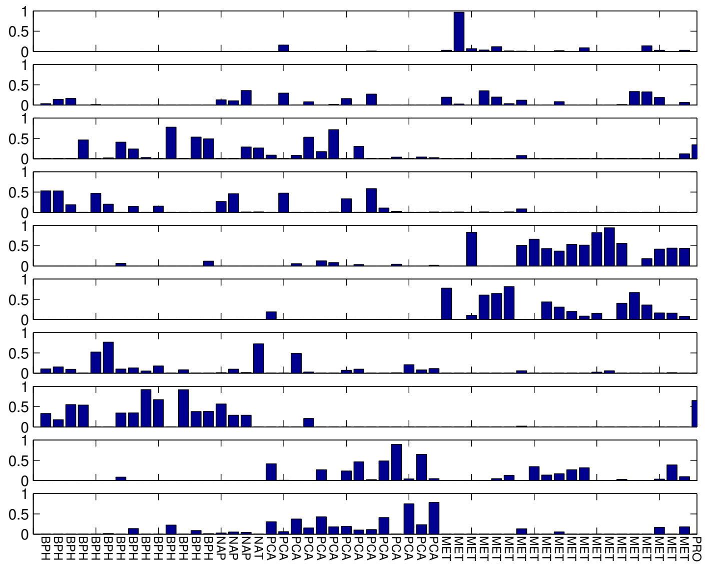


Fig. 4.13: $\mathcal{K} = 10$ process decomposition of prostate data, using top 500 genes without parameter priors.

Naive-Bayes to a peak at $\mathcal{K} = 4$ processes before dropping towards the Naive Bayes curve as $\mathcal{K}$ increases. Although the difference in likelihood is not huge, it is significant (with respect to standard error) and the suggestion of only 2 clusters by Naive Bayes is misleading. Note that we have used the maximum likelihood approach to training LPD here to give the fairest comparison with Naive Bayes which is also trained by maximum likelihood. It is also worth noting the relative sizes of the error bars. For Naive Bayes, it is possible to calculate the likelihood *exactly* whereas with LPD we must take samples from the Dirichlet. The error bars for the two methods are certainly of the same order, suggesting that most variation is due to the cross-validation procedure and allowing us to be confident in our LPD likelihood approximation method.

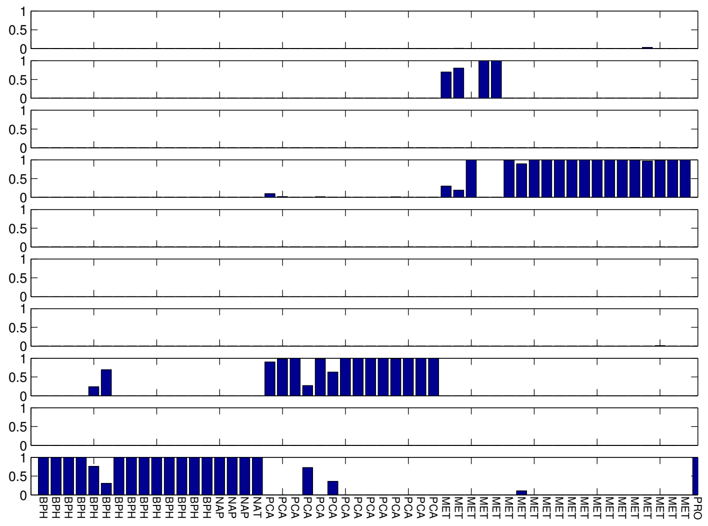


Fig. 4.14: $\mathcal{K} = 10$ process decomposition of prostate data, using top 500 genes with parameter priors. $\sigma_\mu^2 = 0.1$ and $s = 0.5$.

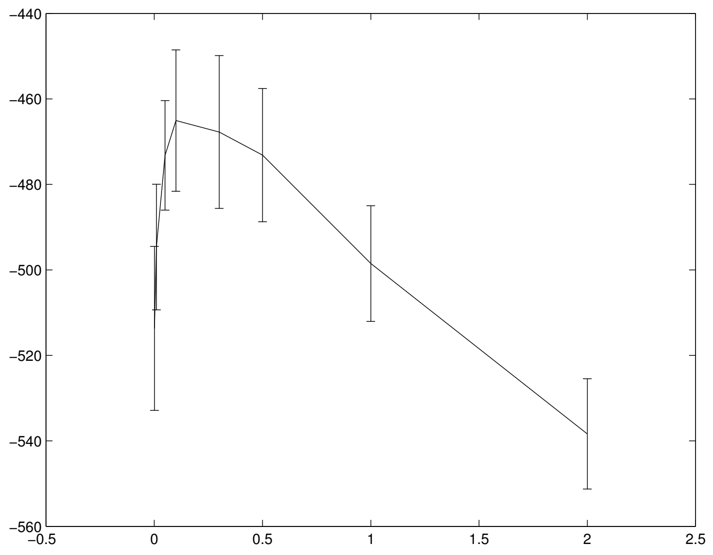


Fig. 4.15: Held-out likelihood as a function of s (the variance prior parameter) for the top 500 genes in the prostate data.

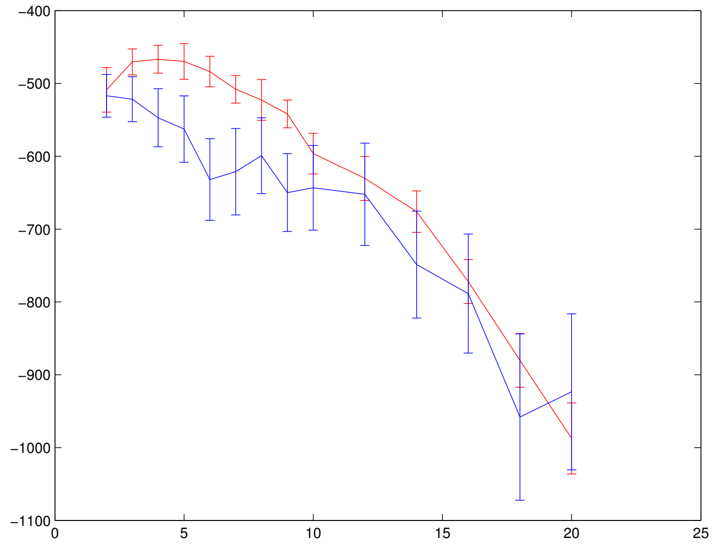


Fig. 4.16: Held-out likelihood for maximum likelihood LPD (red) and Naive-Bayes (blue) for various value of $\mathcal{K}$.

4.8.2 Lung Cancer

The lung cancer dataset [21] consists of 73 gene expression profiles from normal and tumour samples with the tumours labelled as squamous, large cell, small cell and adenocarcinoma. 10-fold cross validation was performed for values of $\mathcal{K}$ between 2 and 20 with and without parameter priors and the results are shown in figure 4.17. As with the prostate, we notice that the introduction of the priors avoids over-fitting and causes the likelihood to plateau. In [21] the authors identified 7 clusters in the data - with the adenocarcinoma samples falling into three separate clusters with strong correlation with clinical outcomes. For their ordering (which we will follow) samples 1-19 belong to adenocarcinoma cluster 1, samples 20-26 belong to adenocarcinoma cluster 2, samples 27-32 are normal tissue samples, samples 33-43 are adenocarcinoma cluster 3, samples 44-60 are squamous cell carcinomas, samples 61-67 are small cell carcinomas and samples 68-73 are from large cell tumours.

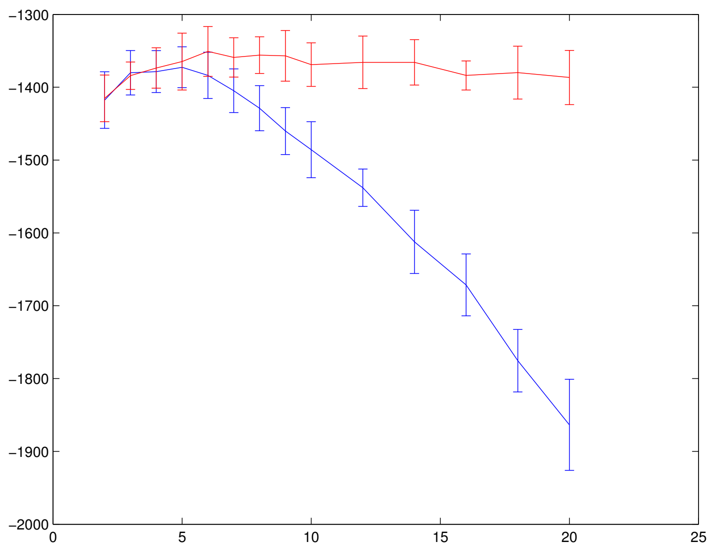


Fig. 4.17: Hold-out likelihood (y-axis) versus $\mathcal{K}$ (x-axis) for the lung cancer dataset with (red) and without (blue) parameter priors.

As in the previous section, the plateau of the likelihood when using priors suggests that we are using more processes than exist in the data and additional processes are being left empty. Cross-validating over s results in figure 4.18 and we can see a peak at somewhere between 0.3 and 0.5. This corresponds reasonably closely to the peak found for the prostate data in the previous section.

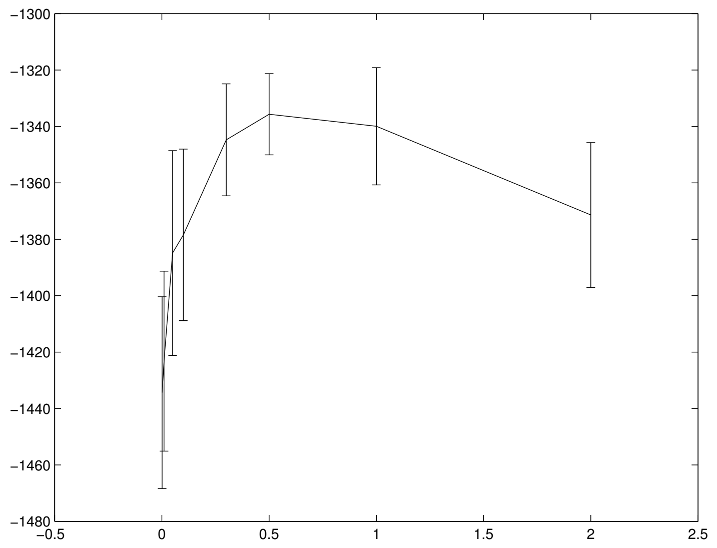


Fig. 4.18: Held-out likelihood as a function of s for lung data.

The results presented in the previous section suggest that the use of the prior is certainly worthwhile. From the prior (red) curve in figure 4.17, $\mathcal{K} = 10$ seems reasonable. Using the algorithm to decompose the data into 10 processes we observe the normalised γ_a values seen in figure 4.19. The samples are in the order in which they are presented in the original paper [21].

First we observe that the clustering by dendrogram is accurately reproduced by latent process decomposition. One notable difference is that the last two entries for the small cell carcinoma grouping are placed with the adenocarcinoma cluster 2 by latent process decomposition with high values of probability (close to one). For the original clustering by dendrogram these two samples are two adenocarcinomas inaccurately placed by the dendrogram in a cluster with small cell carcinomas. This difference is therefore in favour of latent process decomposition and it is also important : the treatment regime for small cell tumours is distinct from that for non-small cell lung cancers (i.e. adenocarcinomas, squamous and large cell tumours) with a pronounced initial susceptibility to chemotherapy. For latent process decomposition we find that the adenocarcinoma cluster 1 is decomposable into processes 1 and 2 and the squamous cell carcinomas are decomposable into processes 5 and 6, both of which have little commonality with any other tumour types in the dataset. This strongly suggests further taxonomic decomposition into subtypes. However, this possibility should be treated with caution: decomposition into sub-processes could also mirror the different stages of a particular subtype, for example.

In [21] genes were extracted from the data in a supervised approach that distinguished well between the different adenocarcinoma clusters. We can perform the same task directly from the latent process model without employing knowledge of class labels as we will now illustrate. Suppose that we are interested in the difference between process 1 in figure 4.19 and process 4 (these 2 processes have been chosen as they correspond to clusters defined in [21] and hence we can compare with their interpretation - the same method could obviously be applied to any pair of processes.). The Z score introduced in the previous section gives a measure of the difference between two normal distributions. Table 4.2 shows the top ten genes ranked by their difference across processes 1 and 4 (adenocarcinoma 1 and adenocarcinoma 3). Of these top genes, numbers 1, 2 and 10 were explicitly mentioned in [21] as being significant. Importantly, and unlike in [21], these features have been extracted with no prior knowledge of the class memberships[2].

Finally, as with the prostate analysis, we should use the held-out likelihood to compare the LPD method with Naive-Bayes. Again, we will use the maximum likelihood method to train both models to ensure a fair comparison. The results can be seen in figure 4.20. As with prostate, we see that although the difference between the two methods is small, LPD tends to over-fit less quickly as $\mathcal{K}$ increases.

[2]For a more complete list and comparisons between other processes see `http://www.enm.bris.ac.uk/lpd`

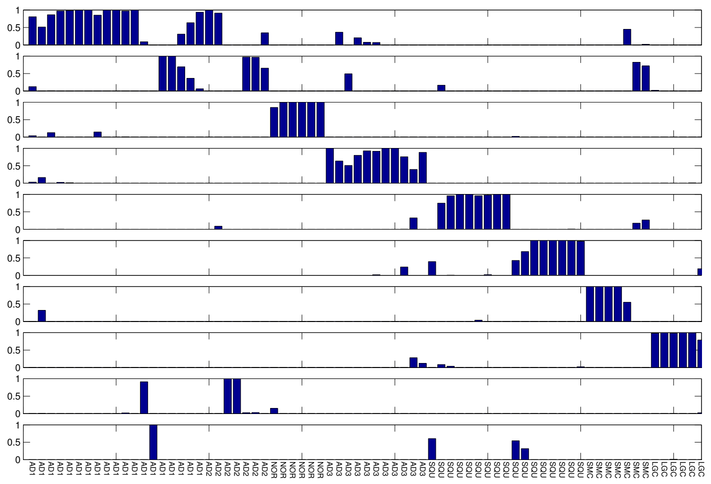


Fig. 4.19: Normalised γ_c values for 10 process decomposition of the lung cancer data. Each column corresponds to one sample and each row corresponds to one component of $\boldsymbol{\theta}$.

Rank	**Name**	Z
1	ESTs Hs.11607	2.5462
2	Solute carrier family 7, member 5	2.1824
3	Transmembrane, prostate androgen induced RNA	1.8716
4	Trophinin associated protein (tastin)	1.8706
5	Fructose-1,6-bisphosphatase 1	1.7742
6	Epididymal secretory protein (19.5kD)	1.7589
7	Phosphofructokinase, platelet	1.7040
8	ESTs Hs.76728 H00660	1.6998
9	Solute carrier family 2, member 1	1.6893
10	Protein tyrosine kinase 7	1.6774

Tab. 4.2: Top lung genes, ranked by Z.

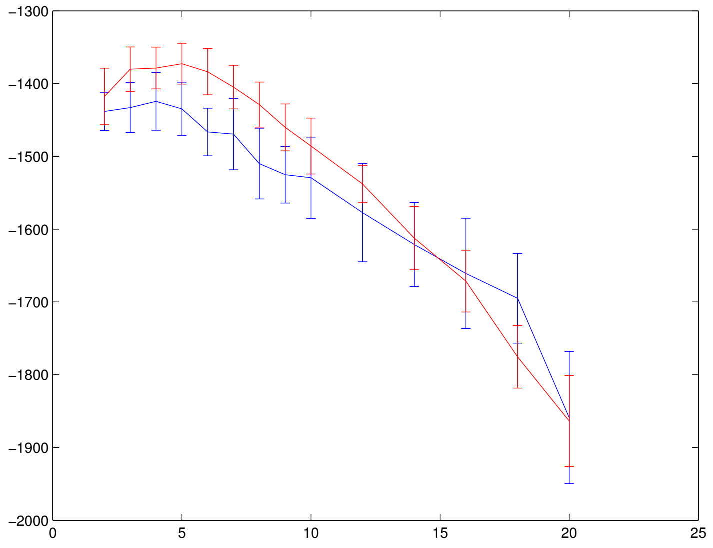


Fig. 4.20: Held-out likelihood for maximum likelihood LPD (red) and Naive-Bayes (blue) for various value of $\mathcal{K}$.

4.8.3 Lymphoma

Finally we considered the Lymphoma dataset of Alizadeh et al [4] which consists of 96 samples of normal and malignant lymphocytes. In this study the authors used samples from diffuse large B-cell lymphoma (DLBCL), the most common form of non-Hodgkin's lymphoma in addition to including data for follicular lymphoma (FL), chronic lymphocytic leukemia (CLL) and other cell types. In Figure 4.21 we show the hold-out likelihood with 10-fold cross validation with and without priors on the parameters. Using $\mathcal{K} = 10$ we can cross-validate over different values of the variance prior parameter s, the results of which can be seen in figure 4.22.

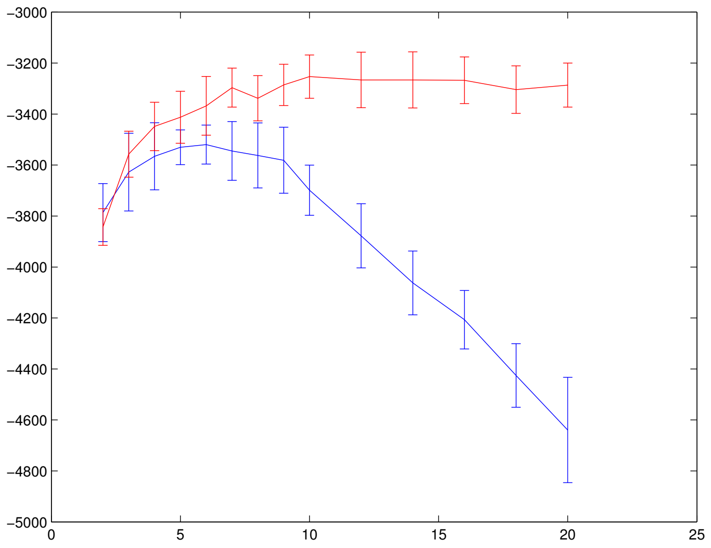


Fig. 4.21: Cross-validation over the number of latent processes without priors (blue) and with priors (red).

As with the lung results in the previous section, when a prior is used, the cross-validation procedure across the number of processes does not give a defined peak. It tends to plateau and as such, gives a minimum number of processes to use. In this case, this minimum is around ten processes. A decomposition into $\mathcal{K} = 10$ processes (using the value of s that gave the maximum in figure 4.22) can be seen in figure 4.23. The samples are presented in the same order as [4] with samples 1 to 16 largely consist of DLBCL with samples 1 and 2 consisting of two DLBCL cell lines (OCI Ly3 and Ly10)

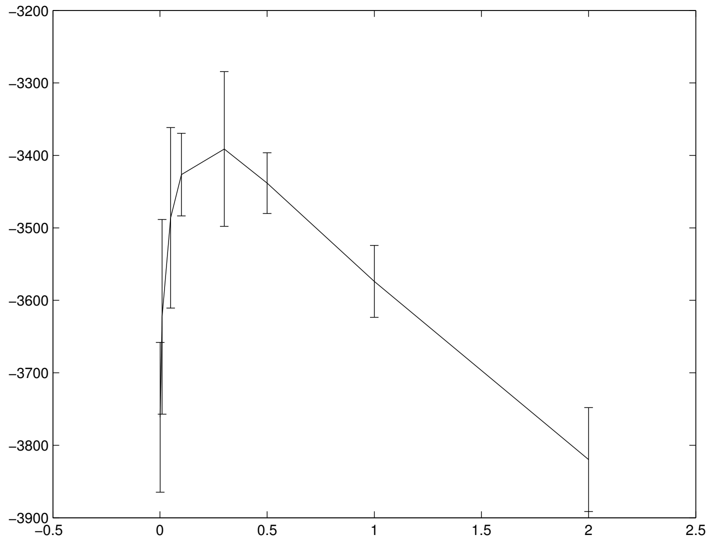


Fig. 4.22: Hold-out likelihood for lymphoma data versus s, the variance prior parameter.

while 10 and 11 are tonsil germinal cell samples. Samples, 17 to 47 consist of DLBCL, samples 48 to 57, 58 to 63 and 65 to 70 consist of blood B and other samples, sample 64 is the OCI Ly1 cell line, sample 71 is a single DLBCL sample placed in with this general cluster of blood B and other tissues, samples 72 - 80 are follicular lymphoma (FL), samples 85 to 95 are CLL and 96 is a lone DLBCL sample. The decomposition agrees well with the structure obtained using a dendrogram [4]. The breaking up of the DLCL samples into several processes suggests some sub-structure, something which is backed up by discussion in [4]. It is also worth looking at some of the areas where LPD and dendrograms differ. For example, sample 71 (DLCL) is now partly grouped with the other DLCL samples and not the FL samples as it was with the dendrogram. Also, process 6 appears to bring together the OCI samples, something which the dendrogram failed to do.

Finally, as with the previous examples, in figure 4.24 we compare the held out likelihood for LPD with that for the Naive-Bayes model. As in the previous examples, we see that LPD does not over-fit as quickly and suggests a greater number of processes than the naive-Bayes model. It also gives a higher maximum value, suggesting it is better suited to this particular data.

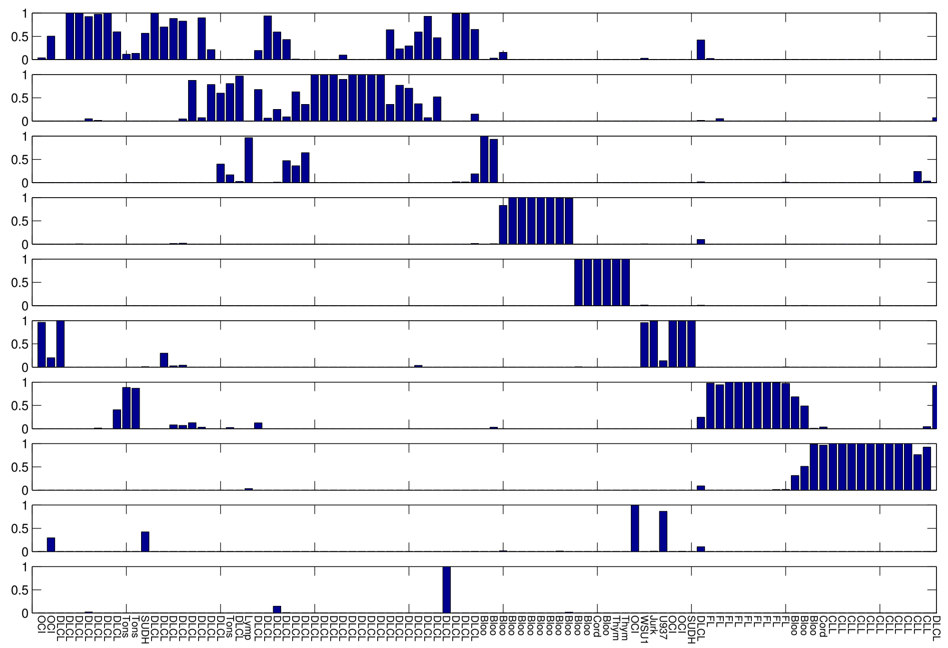


Fig. 4.23: $\mathcal{K} = 10$ process decomposition of lymphoma data.

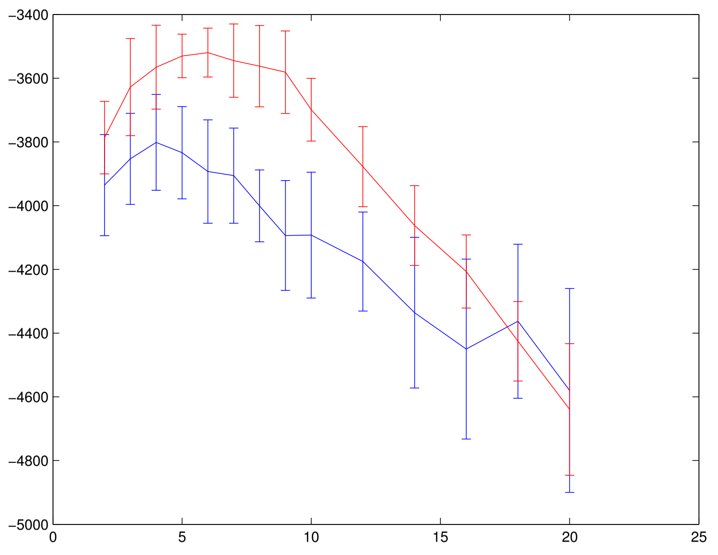


Fig. 4.24: Held-out likelihood as a function of $\mathcal{K}$ for maximum likelihood LPD and the naive-Bayes Gaussian mixture.

4.8.4 Yeast Cell Cycle

The first yeast data set we shall use is the yeast cell cycle data set of Spellman et al [71]. The aim of this study was to investigate which yeast genes showed periodic expression changes during the cell cycle. Microarrays were used that covered the whole of the yeast genome - some 6200 genes. Note that for this dataset, we are performing the decomposition in gene space.

Figure 4.25 shows how the held-out likelihood varies with $\mathcal{K}$ for the maximum likelihood LPD, Naive-Bayes clustering (as described in appendix B) and the model proposed by Segal et al [67]. We can see that LPD gives significantly higher likelihood than either of the other two methods. It is worth noting that the cross-validation on the Segal model does not go over the same range as for the other two models. Each gene/process combination in the Segal model is described by a single binary variable - i.e. 1 if that gene belongs to that process, 0 otherwise. Therefore, assigning genes to processes requires a search through all $2^{\mathcal{K}}$ combinations; something that rapidly becomes infeasible as the number of processes is increased. Various heuristics could be used to arrive at an approximation, however, they could not possibly improve the solution and as such it was not necessary to use them to increase the number of processes above 10. Figure 4.25 shows that for this dataset between 5 and 10 processes give the highest predictive likelihood. From the well-characterized biology of the experiment we expect one process corresponding to each of the known cell cycle phases (G1, S, G2, M, MG1, as defined in the original paper [71]) plus a small number of "classifying" processes distinguishing between the four different synchronisations used in the experiment.

Figure 4.26 shows the ten processes defined by LPD. It can easily be seen that processes 1 and 3 represent "classifying" processes that characterise alpha factor/elutriation synchronisation and cdc15 synchronisation respectively. Processes 2, 4, 6, and 8 correspond to cycling processes and each can be uniquely mapped to one of the major cell cycle phases by reference to the biological information given in [71] (highlighted in fig. 4.26). The fifth cell cycle phase, G2, which is very short in exponentially growing yeast cultures, is not recovered as a distinct process but can easily be obtained by the linear

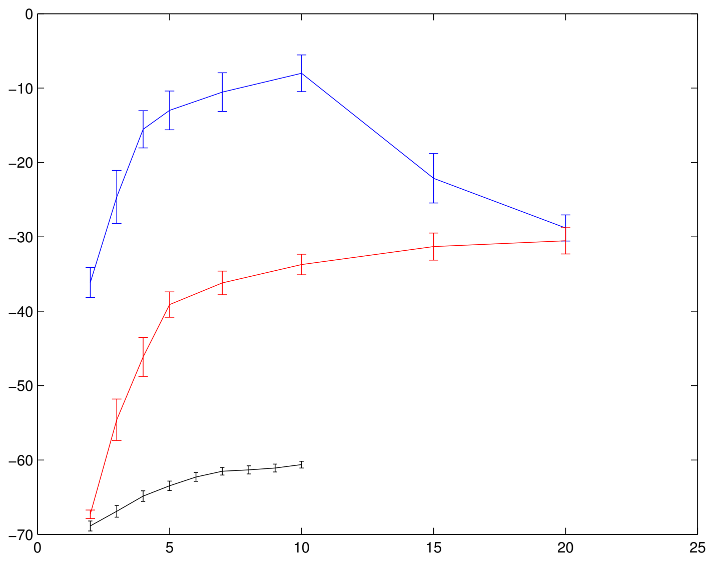


Fig. 4.25: Cross validation on yeast cell cycle data for LPD (top curve), Naive Bayes (middle curve) and the Segal model (bottom curve).

combination of the adjacent S and M phases (not shown). The remaining processes (5, 7, 9, and 10) seem to be dominated by noise and experimental artifacts (particularly in process 7). This emphasises that the identification of the most parsimonious number of latent processes is an extremely useful feature of the LPD approach. This would be even more important in experiments without previous knowledge of the underlying process structure, where the identification of the really informative processes would not be as easy as in the present test case.

From a biological point of view it is particularly noteworthy that the cycling processes were identified without any prior reference to the time-series structure of the experiment. The original publication used Fourier transformation to determine the expression peaks of cyclically expressed genes and Pearson correlation with known cell cycle-regulated genes to assign them to cell cycle phases. In the case of LPD, the same can be achieved by reference to the gene-specific variational parameter γ_{gk}. This is shown in figure 4.27, where the expression of a classical G1 phase marker gene (cyclin CLN2) is indeed represented most strongly by process 2, which according to fig. 4.26 corresponds to peak expression in G1.

Figure 4.28 shows a more complex example of the power of this approach. The histone genes, which code for the main protein component of chromosomes, are known to be most strongly expressed in S phase, when new chromosomes need to be assembled. Indeed, fig. 4.28 shows that expression of one representative histone (H1) is dominated by the S phase-specific process 8 (compare fig. 4.26). However, there are also significant contributions by processes 2 and 6 (G1 and M phase) and the measured expression profile (solid line in fig. 4.28, lower panel) shows clearly that this specific histone has a much broader expression range that extends well beyond S phase. This is not entirely unexpected as the production of histone proteins should well precede the replication of DNA that defines S phase.

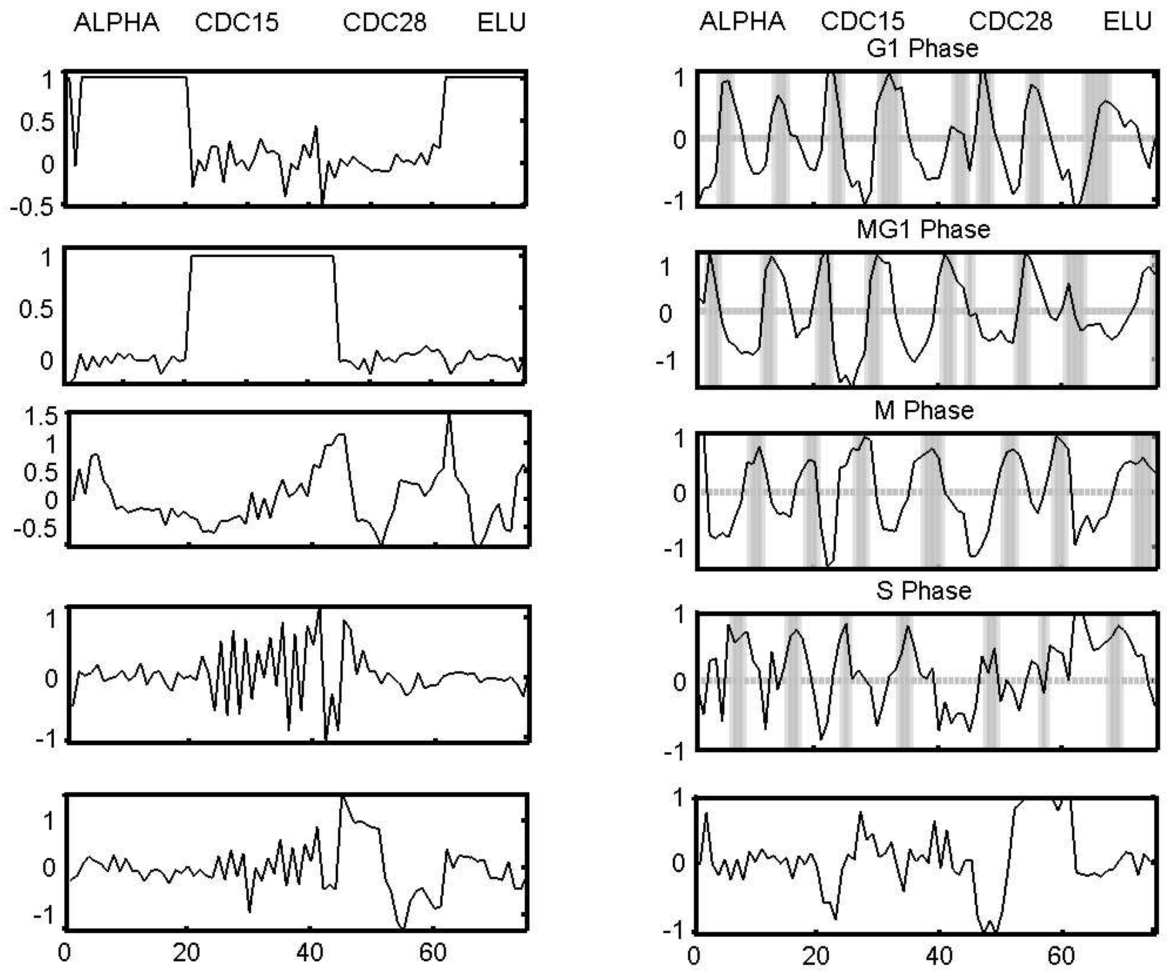


Fig. 4.26: The ten processes identified by LPD of the yeast cell cycle. The cycling processes can be seen to be peaking in the various phases of the cell cycle as highlighted on subplots 2, 4, 6 and 8. The four different synchronization methods are indicated at the top.

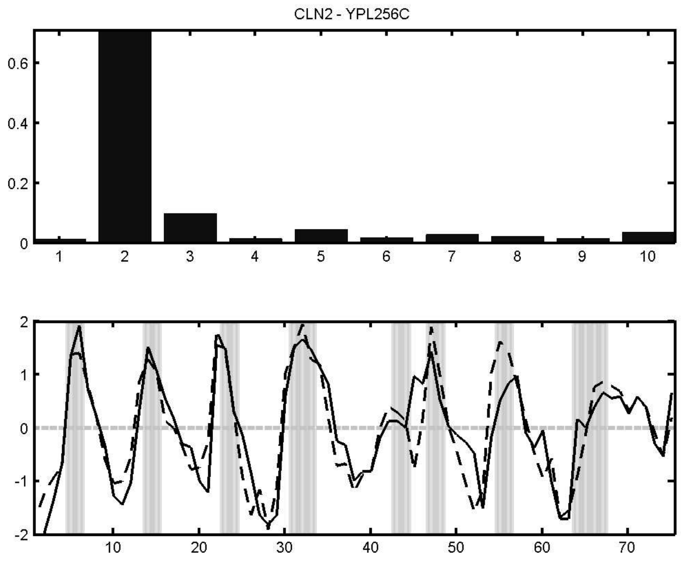


Fig. 4.27: The expression levels of the G1-specific cyclin CLN2 YPL256C peaks in G1 phase (lower panel, solid line). The dotted line is the sum of the mean for each process weighted by the value of the gene specific variational parameter for each process (top bar chart). In this case it can be seen that the second process profile (G1 phase) is indeed dominant for this gene.

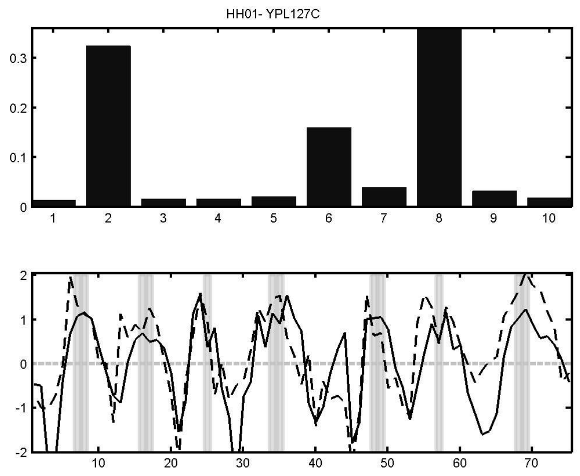


Fig. 4.28: The expression levels of the histone H1 gene (YPL127C) peaks in S phase (lower panel, solid line). The dotted line is the sum of the mean for each process weighted by the value of the gene specific variational parameter for each process (top bar chart). In this case it can be seen that a combination of the second, sixth and eighth process profiles is required for this histone. This combination corresponds to the G1, M and S phase processes identified by LPD (figure 4.26).

4.8.5 Yeast Environmental Stress

For this example we have used the yeast dataset of Gasch et al. [22]. In this study cDNA microarrays were used to determine temporal alterations of yeast gene expression levels in response to a variety of environmental changes such as heat shock, hydrogen peroxide, amino acid starvation, etc. 142 samples were analysed with whole-genome microarrays. Figure 4.29 shows that the highest predictive likelihood is obtained using

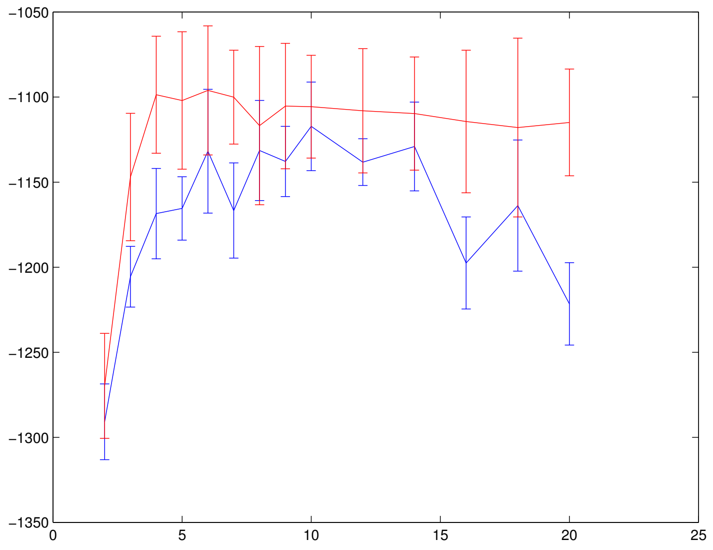


Fig. 4.29: Cross validation on yeast shock data for LPD (red) and Naive Bayes (blue).

between 10 and 15 processes. Fig. 4.30 illustrates the processes for $\mathcal{K} = 15$. It can be seen that this number of processes is probably too large, as several of them appear to be very noisy and partially redundant (e.g. processes 4, 5, 6, and 11). This again highlights the importance of being able to identify the most parsimonious number of processes by LPD. However, it is also obvious that even with the large number of processes a number of them clearly dominate certain types of experiment. E.g., process 9 is strongly "induced" during the entry into stationary phase. It can also be seen that the same process contributes significantly to the late stages of nitrogen starvation as would be expected based on the biology of the experiment: Stationary phase is characterised by the depletion of resources including nitrogen.

In fact, many experimental conditions in this dataset are best described by a combination of processes, while several processes contribute to a variety of experimental conditions. Again this is expected biologically: All the experimental treatments in this study involve stressful conditions. Many of the stresses will affect a number of general processes (such as starvation or protein denaturation). On the other hand, the intense stress will affect each sample in various aspects of its biology. Such an observation would not be easily possible based on standard clustering results. Furthermore, just as in the cancer studies, the probabilistic representation provided by LPD could directly be used to identify the specific genes that dominate each of the distinct types of stress response.

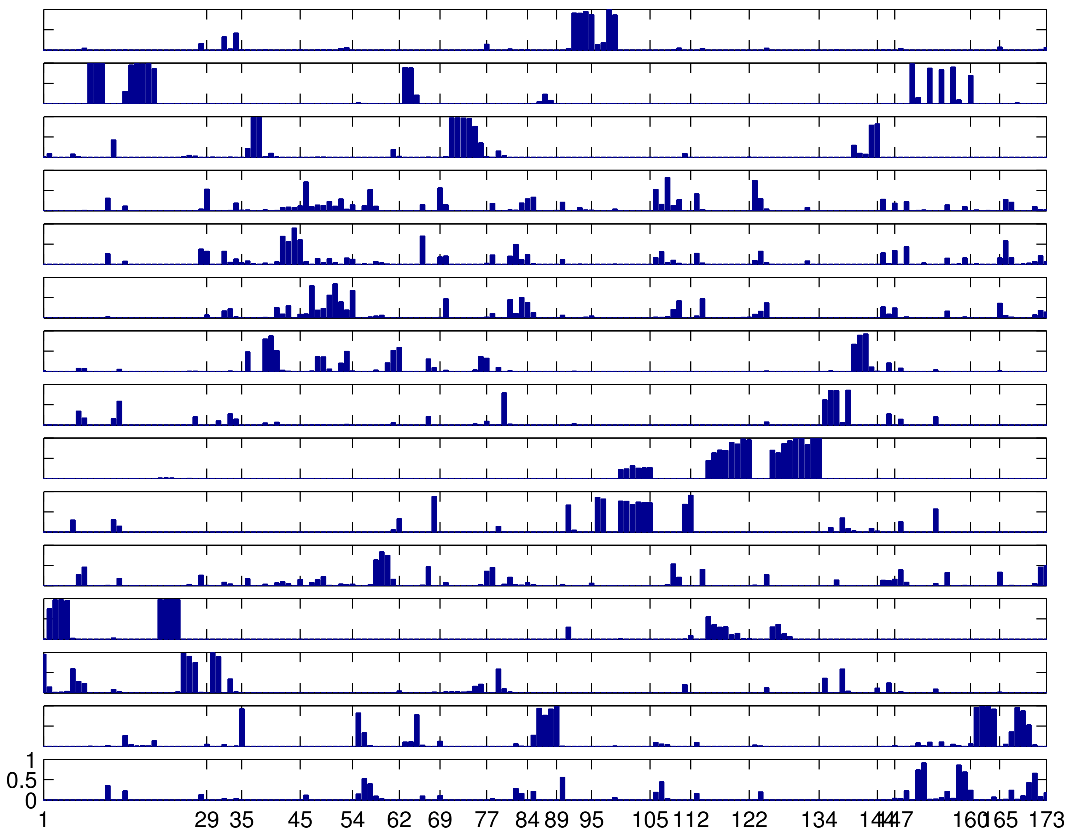


Fig. 4.30: The variational parameters for a fifteen-process LPD of yeast environmental stress. The ticks on the horizontal axis correspond to the following experimental conditions in order; heat shock, sorbitol, H202, menadione, DTT, DTT-2, diamide, sorbitol-2, osmotic stress, amino acid starvation, nitrogen depletion, diauxic shift, stationary phase,stationary phase-2,dby,msn2, carbon source, cold stress, cold stress-2.

4.9 LPD formulation for CESH

4.9.1 CESH - a brief introduction

Microarrays allow us to measure the expression levels of thousands of genes simultaneously and as we have discussed previously, this has been shown to be very useful in classifying tumours and predicting clinical behaviour. However, there are drawbacks to the microarray method. Firstly, the data collection process is both time consuming and expensive. Although this has not stopped its extensive and succesful use in research, it currently renders it impractical as a diagnosis tool. Secondly, as we describe in this thesis, the analysis and interpretation of thousands of noisy data points is far from trivial.

CESH (Comparative Expressed Sequence Hybridisation) [45] has been developed as a rapid expression profiling technique. Rather than measuring the expression of particular genes, expression is measured directly from the chromosomes, identifying regions on the chromosome that correspond to differential gene expression. As with microarray experiments, we require a control sample as well as our sample of interest and these are labelled with different dyes and hybridised to normal chromosomes. The ratio of dye intensities is then used as a measure of expression. If a particular region has significantly high expression, it is labelled $+1$, significantly low -1 and everything else is labelled 0. The measurement process is significantly faster than that for microarray data and the dimensionality of the data produced is roughly a factor of 10 less than standard microarray experiments. CESH also has the added advantage that only very small quantities of RNA are required and so the technique is particularly useful for small samples.

This descrete form of data is obviously not suited to the LPD formulation that we have described thus far. However, in our full expanded bound on the log likelihood (equation 4.12) we can see that $p(e_{ga}|\mu_{gk}, \sigma_{gk})$ only appears in one term and hence only has an effect on the updates for its own parameters and the update for Q_{akg}. Additionally, the Q_{akg} update (equation 4.13) makes no explicit assumptions about the form of

$p(e_{ga}|\mu_{gk},\sigma_{gk})$. Therefore, we can plug-in a new parametric form and derive new fixed point iterations.

The CESH data is discrete and has three possible states - normal, over-expression and under-expression. Therefore, the most obvious choice of distribution is a multinomial across these three states. If we define a 3 element binary vector $\mathbf{v}_{ga}$ to denote the state of region g on array a as $[1,0,0]$ if the region is under-expressing, $[0,1,0]$ if it is normal and $[0,0,1]$ for over-expressing then we have the following form for our probabilities

$$p(\mathbf{v}_{ga}|\mathbf{p}_{gk}) = \prod_{i=1}^{3}(p^i_{gk})^{v^i_{ga}} \tag{4.27}$$

where we have used the superscript to denote particular elements of a vector. Our new parameters, p^i_{gk} represent the probability that region g is in state i in process k.

Following an identical procedure to the gaussian LPD described previously (decomposing over arrays and not genes), our updates for these parameters are obtained by differentiating the following expression

$$\sum_{g=1}^{\mathcal{G}}\sum_{a=1}^{\mathcal{A}}\sum_{k=1}^{\mathcal{K}} Q_{agk}\log p(\mathbf{v}_{ga}|\mathbf{p}_{gk}) - \lambda\left(\sum_{i=1}^{3} p^i_{gk} - 1\right) \tag{4.28}$$

where the Lagrangian multiplier λ has been introduced to ensure that the summation constraint over the multinomial probabilities is satisfied. Differentiating this with respect to p^i_{gk} and re-arranging results in the following update

$$p^i_{gk} = \frac{\sum\limits_{a=1}^{\mathcal{A}} Q_{agk}v^i_{ga}}{\sum\limits_{a=1}^{\mathcal{A}} Q_{agk}}. \tag{4.29}$$

The only other change in the updates is in the equation for Q_{agk}. Trivially, this becomes

$$Q_{agk} = \frac{p(\mathbf{v}_{ga}|\mathbf{p}_{gk}) \exp\{\mathbf{E}_{p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)}[\log \theta_k]\}}{\sum_{k'=1}^{\mathcal{K}} p(\mathbf{v}_{ga}|\mathbf{p}_{gk'}) \exp\{\mathbf{E}_{p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)}[\log \theta_{k'}]\}} \tag{4.30}$$

and similarly, the likelihood approximation becomes

$$\mathcal{L} \sim \sum_{g=1}^{\mathcal{G}} \log \frac{1}{N} \sum_{n=1}^{\mathcal{N}} \left[\prod_{a=1}^{\mathcal{A}} \sum_{k=1}^{\mathcal{K}} p(\mathbf{v}_{ga}|\mathbf{p}_{gk}^{i}) \theta_k^n \right]. \tag{4.31}$$

Everything else remains unchanged.

Figure 4.31 shows how the held-out likelihood varies as $\mathcal{K}$ is increased from 2 to 10 for soft tissue sarcoma samples. Unlike the curves for the microarray data, we do not observe a single well-defined peak, rather the likelihood is reasonably flat until $\mathcal{K} = 4$ processes and then the begins to drop suggesting the model is over-fitting. A $\mathcal{K} = 4$

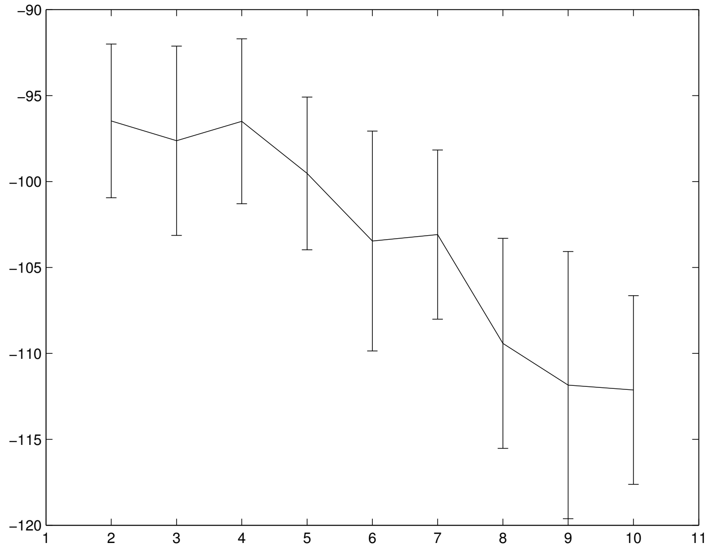


Fig. 4.31: Held-out likelihood versus $\mathcal{K}$, the number of processes for the CESH data.

process decomposition of the CESH data can be seen in figure 4.32. We can see that some of the class structure present in the data has been uncovered but the results

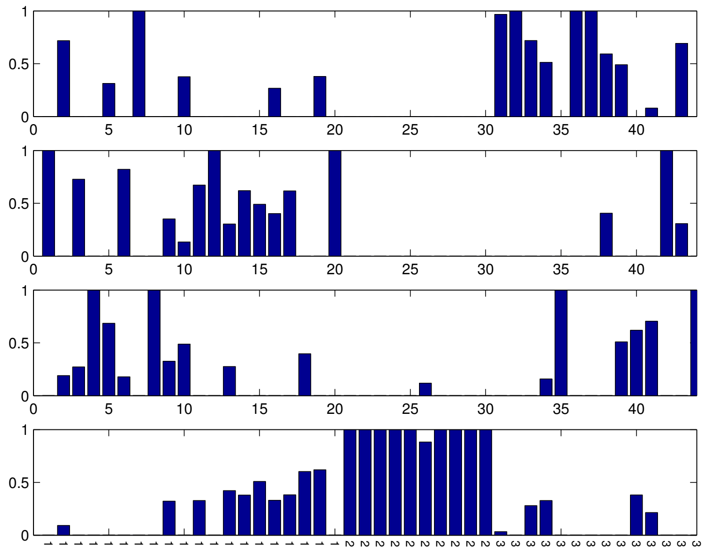


Fig. 4.32: $\mathcal{K} = 4$ process decomposition of the CESH data. 1 corresponds to Leiomyosarcoma samples, 2 to Liposarcoma and 3 to Malignant Fibrous Histiocytoma.

are far from convincing. Unfortunately, due to the present lack of CESH data, it is impossible to say whether or not this poor performance is due to the choice of model or volume of data and when more data becomes available it will be an interesting research direction. As with the gaussian LPD model, it may be beneficial to place priors on the model parameters. In this case, given that the parameters are multinomials, the priors would be Dirichlets. Again, these enhancements to the model will be interesting to persue when more data becomes available. The details of the changes to the update equations required to incorporate such priors are given in appendix E.

4.10 Conclusions

In this chapter, we have presented a new framework (LPD) for the analysis of microarray data. We have argued that this new method overcomes the limitations of standard clustering techniques - for example, the restriction of data points to single structures, the inability to objectively assess the number of clusters in the data and the lack of transparency. Using datasets from various forms of cancer and the yeast (*Saccharomyces Cerevisiae*) we have demonstrated how, in an unsupervised manner, LPD is able to successfully extract biologically relevant genes and processes. LPD is not restricted to microarray data. The manner in which the assumed probabilistic form decouples from the rest of the expressions allows any particular form to be used, so long as it results in tractable updates. This is demonstrated with the description of a model that could be used for CESH data. Finally, there are several ways in which the basic LPD model could be enhanced. One possible extension is the use of a mixture of Dirichlet priors rather than an individual one. Details of such a model are given in appendix D. Experimentation showed that such a model didn't offer any real improvement over the original model. However, as more data becomes available it may be worth returning to.

Chapter 5

Estimating Sample Size Requirements

5.1 Introduction

In recent years, the application of machine learning algorithms to DNA microarray data to perform cellular classification for various tumour types has been shown to have both statistical and clinical significance. For example, investigations into leukemia [23], lymphoma [68], brain cancer [58], lung cancer [6] and the classification of multiple primary tumours [60, 59, 81] have shown how classification and clustering algorithms have been able to distinguish between different classes of samples and even discover previously unknown disease subtypes. Given a particular DNA microarray dataset and having performed this initial classification, it is reasonable to question how much improvement in performance could be obtained if more data were available - given a current training set size p and generalisation error ϵ, how big a training set $\bar{p}$ do we need to obtain $\bar{\epsilon}$. In other words, is it worth embarking on a larger study including more samples to improve classification *or* make the results obtained more rigorous. The ability to answer such a question potentially offers the experimentalist huge savings in both time and money. This work aims to address the following questions

1. For a given number of samples, how significant is the performance of the classifier?

2. If the results are significant, can we predict the performance of the classifier when presented with additional samples and will this performance improve?

3. With new alternatives to microarrays becoming available, is it possible to compare them with microarrays to discover which is more efficient?

In this chapter, we present the following novel work (published in [51]). Firstly, we develop a simple methodology for assessing the significance of a particular classification result. An inverse power law model is then constructed to model the error-rate as a function of the training sample size. This model can then be used to extrapolate the test error for larger sample sizes. Power calculations [3] are a standard way of estimating sample-size requirements, however, there are two reasons why they are not applicable here. Firstly, they rely on the assumption that the samples are independently drawn from a Gaussian distribution. Clearly, this is not applicable here. Secondly, they address the problem of determining the level of confidence in an empirical error estimate whereas we are interested in how the error rate may decrease given more data.

It is important not to confuse this with the problem of calculating the confidence in a particular error rate as described in [3] and [24] which compute bounds or estimates of the deviation of the test error from its expected value as a function of the number of samples. There have also been other studies more specific to microarrays which analyse variations due to factors that can be addressed by experimental design or by the use of replication [73, 37, 38]. Also, [43] looked at how replication and sample size can increase the power of ANOVA models. However, none of these examples are particularly useful in determining the future performance of a classifier when presented with more samples. We will introduce a method that will enable us to predict dependence of generalisation on sample size and test this method on eight different cancer data sets representing a large spectrum of classification tasks. These range the from the relatively straightforward task of discriminating different tumour types to the more challenging task of predicting the outcome of treatment. We shall see that this difference in classification difficulty will be reflected in the number of training samples required for low error rates.

In the following section, we will motivate our learning model from a statistical mechanics perspective. In section 3 we will introduce our significance test for error rates. In section 4 we present results obtained from the eight benchmark data sets whilst in section 5 we use our methodology to compare the classification performance of microarrays with that of a new experimental technique - Comparative Expressed Sequence Hybridisation (CE SH). In the final section we present our conclusions.

5.2 A statistical mechanics motivation

The probability approximately correct (PAC) formulation to learning theory enables bounds to be derived on the number of training samples required to obtain generalisation error less than some arbitrarily small value ϵ with probability greater than some arbitrarily large value δ. However these bounds tend to be very general and, as they are derived in a worst case sense, tend to be far too pessimistic to be of any practical use. For example, following Mitchell [50], we can derive the following bound. We wish to determine the minimum number of training patterns p that, with probability greater than δ, give a generalisation error less than ϵ. We make the following assumptions

- We are trying to learn a hypothesis c from some set $\mathcal{C}$ given independent training samples drawn from some stationary distribution $\mathcal{D}$.
- Our choice of learning algorithm defines some finite hypothesis space $\mathcal{H}$.
- Our learning algorithm is consistent - it is able to fit the training data perfectly.

Now, we define the version space as the subspace of all hypothesis in $\mathcal{H}$ that are consistent with the training data. All consistent hypothesis that have error less than ϵ must lie inside this version space. Therefore, we can restate our problem as that of finding the number of training examples required such that there are no hypothesis in the version space with error greater than ϵ. A version space that contains no hypothesis with error greater than ϵ is known as ϵ-exhausted. Haussler [26] proved that the probability that the version space is *not* ϵ-exhausted is less than or equal to

$$|\mathcal{H}| \exp\{-\epsilon p\}. \tag{5.1}$$

We will now sketch the proof of this theorem, for a more detailed description, the reader is referred to [50].

The version space is *not* ϵ-exhausted if there is at least one hypothesis h with error greater than ϵ that is consistent with all p training examples. The probability of a hypothesis h that has error greater than ϵ being consistent with a random training example is at most $(1-\epsilon)$. Therefore the probability that it is consistent with p of them is bounded above by $(1-\epsilon)^p$. Now, let us assume that k hypothesis exist with error greater than ϵ. Obviously, $k \leq |\mathcal{H}|$ and therefore the probability of the version space not being ϵ-exhausted is upper bounded by $|\mathcal{H}|(1-\epsilon)^p$. Using the standard inequality that if $0 \leq \epsilon \leq 1$, $(1-\epsilon) \leq \exp\{-\epsilon\}$, the probability that the version space is not ϵ-exhausted is less than or equal to $|\mathcal{H}| \exp\{-\epsilon p\}$.

Now, we would like to bound this quantity so that the probability of failure (i.e. version space *not* ϵ-exhausted) is below some pre-determined quantity δ.

$$|\mathcal{H}| \exp\{-\epsilon p\} \leq \delta. \tag{5.2}$$

Solving this for p gives us

$$p \geq \frac{1}{\epsilon}(\ln |\mathcal{H}| + \ln(1/\delta)). \tag{5.3}$$

Such bounds tend to be very pessimistic due to the $|\mathcal{H}|$ term as realistically $k << |\mathcal{H}|$. Therefore, although theoretically, such bounds are very useful, in a practical sense they are likely to be far too vague.

The alternative approach is taken from statistical mechanics. Statistical mechanics is a field of theoretical physics used to model complex systems with large numbers of variables composed of many interacting entities. In particular, the *thermodynamic limit* of statistical mechanics enables us to simulate performance for small sample sizes relative to the number of tunable parameters. This approach has been successfully applied to a number of neural network models and learning algorithms (for example [14, 55, 17]) and we will briefly describe the overall methodology with respect to the linear perceptron as introduced above.

Consider the following formulation of the learning problem. We have a teacher **T** and a student **S** with the same parametric form. The goal of any learning (specifically for classification) algorithm is the following: given a set of training examples and corresponding labels from the teacher, train the student such that it gives optimal performance (i.e. agrees with the teacher) on further, unseen examples. For illustration, consider the linear perceptron (we will assume that the bias term has been included in the weights). We are interested in learning a set of weights $\mathbf{w} \in \mathbb{R}^N$ (where N is the dimensionality of our input space plus the bias) which will provide the following decision function

$$y_i = \text{sign}(\mathbf{w} \cdot \mathbf{x}_i). \tag{5.4}$$

As this set of weights is the only tunable parameter in both our teacher and our student, for convenience, we will denote the teacher's weights by **T** and the student's by **w**. We are interested in obtaining a functional mapping between the number of training patterns and the generalisation error of the student.

The argument inside the sign function in 5.4 is invariant under any positive re-scaling of the weight vector, i.e. $\mathbf{w} \to \lambda\mathbf{w}$ for any $\lambda > 0$. Hence, we can implicitly fix a scale by, for example, selecting $\mathbf{w}^2 = N$, where N is the input dimensionality. Theoretically, given a set of training patterns $\mathcal{X}$, the easiest way to set the weights of our student, **w** is to simply pick a student at random from the version space (the space of all students consistent with the teacher over the training patterns) - Gibbs learning. When a new training pattern is presented, some of the weight vectors inside this version space may become inconsistent with the teacher and the size of the version space will decrease. As this space decreases, the *average* student will overlap more and more with the teacher and hence the generalisation error will drop. Therefore, if we can obtain an expression for the size of the version space after say p patterns, we can obtain an approximation for the generalisation error.

For a particular weight vector, **w**, teacher **T** and data set $\mathcal{X} = \mathbf{x}_1, \ldots, \mathbf{x}_p$ the following indicator function returns 1 iff the output of the student perceptron is consistent with that of the teacher perceptron for all data points in the set

$$I(\mathbf{w}) = \prod_{i=1}^{p} \theta \left[\left(\frac{1}{\sqrt{N}} \mathbf{T}\mathbf{x}_\mathbf{i} \right) \left(\frac{1}{\sqrt{N}} \mathbf{w}\mathbf{x}_\mathbf{i} \right) \right] \tag{5.5}$$

where $\theta(z) = 1$ if $z > 0$ and 0 otherwise. Therefore, the volume of the version space V_p for a particular set of p patterns and a particular teacher is

$$V_p = \int_{\mathbf{W}} \prod_{i=1}^{p} \theta \left[\left(\frac{1}{\sqrt{N}} \mathbf{T}\mathbf{x}_i \right) \left(\frac{1}{\sqrt{N}} \mathbf{w}\mathbf{x}_\mathbf{i} \right) \right] \delta(\mathbf{w}^2 - N) d\mathbf{w} \tag{5.6}$$

where $\delta(z) = 1$ if $z = 0$ and 0 otherwise.

Taking the expectation of this quantity with respect to the training patterns and possible teachers, and then taking the limit $N \to \infty$ we obtain the so called *annealed approximation*. In this limit it can be shown that the version space is dominated by weight vectors with one particular value of generalisation error (see Engel [17] p. 16 for more details). Therefore we have an approximation for the generalisation error as a function of sample size, p. However, this approximation is not particularly accurate. By taking the expectation of V_p with respect to the training samples and possible teachers, we have calculated the mean of a product of many random variables. Such products generally have distributions with very long tails such that the mean and most probable value (mode) do not coincide - i.e., they are not *self-averaging*. This problem is overcome by finding the expected value of $\ln V_p$ and then exponentiating the result. Taking the logarithm leaves us with a sum over random variables, something that *is* generally self-averaging.

Calculation of $\mathbf{E}[\ln V_p]$ is far from trivial and relies on the so-called *replica* trick. This allows us to transform our problem from one of calculating $\mathbf{E}[\ln V_p]$ to one of calculating

$$\lim_{n \to 0} \frac{\mathbf{E}[V_p^n] - 1}{n}. \tag{5.7}$$

Further details of the use of the replica trick to perform this calculation are beyond the scope of this thesis and the reader is referred to [17] for details.

The method described above for Gibbs learning can be extended to other perceptron learning algorithms. In the large β limit ($\beta = p/N$), Opper et al [55] found that the generalisation error of an optimal perceptron scales as $0.5p^{-1}$ (as the value of N is fixed, we can assume the functional dependence on the number of patterns rather than β). Similarly, the Bayes optimal classifier (corresponding to the student at the centre of mass of the version space) scales as $0.44p^{-1}$ ([54]). For the Adaline rule, we get a scaling of $0.24p^{-1/2}$ and Hebb scales as $0.40p^{-1/2}$ (see [77] for a good review). Hence it appears that the dependence on p is approximately $ap^{-\alpha}$ (where α approaches 1 for the more efficient rules) and this looks like a good basis for our functional form. However, there are still issues that need to be addressed:

- These solutions are for the perceptron and not our algorithm of interest - the Support Vector Machine.
- These relationships are in the large β limit so they hold when the number of patterns is much greater than the number of features - clearly not the case with microarray data.
- They assume a perfect teacher - i.e., the training samples presented are noise free.

The first issue is easily overcome. Dietrich et al [15] argue that a linear support vector machine has the same generalisation error dependence as the linear perceptron. We can rather subjectively overcome the second issue as follows. In the large β limit, the generalisation performance can be extracted explicitly, as we have seen. In the small limit, we have to resort to numerical techniques to extract a model of the behaviour. When this is done, we find that the best fit of our chosen functional form gives us a very good fit. Figure 5.1 shows the numerical solutions for a range of β values and also the best fit of our functional form. As can be seen from the figure, the two curves coincide and the fit is very good over a large range of β values and not just in the large β limit. We can also overcome the final issue. The effect of adding white noise to the training examples suggests that we will never be able to obtain perfect performance. This is equivalent to adding a constant term to our function that determines the best error rate achievable.

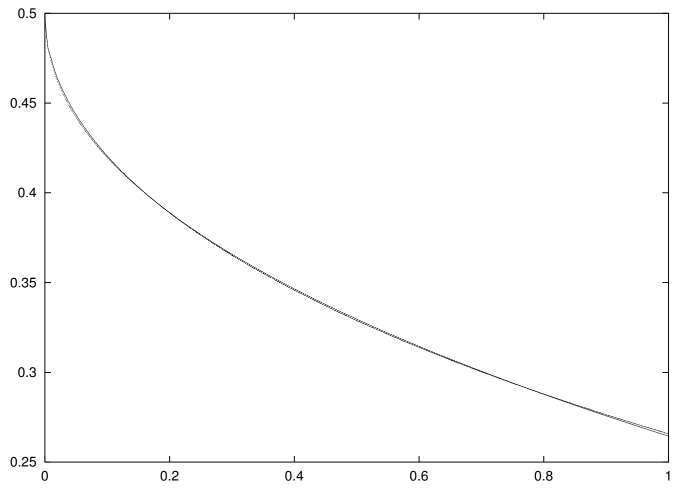


Fig. 5.1: Theoretical solution for various values of β and best fit curve of the form $ap^{-\alpha} + b$.

To summarise, the arguments presented above, lead us to the following functional form relating sample size and generalisation error

$$e(p) = ap^{-\alpha} + b \tag{5.8}$$

where $e(p)$ is the expected error rate after presentation of p examples, a is the *learning rate*, α is the *decay rate* and our noise term b is the *Bayes error* - the minimum error rate achievable. These constants will change for datasets of different difficulties and levels of noise and so we will proceed by using empirical error rates from each dataset for various sample sizes to fit the parameters of the model.

5.3 Methodology

In order to fit our error model to particular datasets, we need to build classifiers of various sample sizes to evaluate the required empirical errors. This is achieved through sub-sampling of our complete dataset of size P into a training set of size p and a test set of size $P - p$. This is repeated T_1 times for each sample size p. The average error across these T_1 realisations is used as our error estimate and the 25th and 75th percentiles are used to define an error envelope. This average is an un-biased estimate of the true error

- see appendix A for details. The 25th and 75th percentiles are used in place of the variance because the variance tends to be optimistically biased - see appendix A for details.

Before we can fit our model to these empirical errors, we need to assess whether or not they are statistically significant. This is particularly important for small sample sizes; without enough samples, the classifier will not be able to learn any structure and is effectively random. We can evaluate this significance by comparing the measured error with the error obtained by random classifiers realised by training with the same input vectors but permuted labels. For each of our T_1 realisations of size p, we perform T_2 permutations of the labels to give T_2 random errors. For each size p we can now construct the following empirical distribution for the random error

$$P_p^{ran}(x) = \frac{1}{T_1 T_2} \sum_{i=1}^{T_1} \sum_{j=1}^{T_2} \theta(x - e_{p,i,j}) \tag{5.9}$$

where $\theta(z) = 1$ if $z \geq 1$ and 0 otherwise. The significance of the classifier is then given by $P_p^{ran}(\bar{e}_p)$, which is the percentage of random classifiers with error rate smaller than $\bar{e}_p$, where $\bar{e}_p$ is the average error of the actual classifier. It is informative to illustrate this with a small fictitious example. A dataset was created with $P = 200$ and $N = 100$ features. The first two features were separable Gaussians whilst the remaining 98 were noise. With $T_1 = 1$ and $T_2 = 100$, the actual error and random error distributions were calculated for $p = 5$ and $p = 50$. The random distributions can be seen in figures 5.2 and 5.3 respectively. The actual errors for the two values of p were 0.4205 and 0.1467 corresponding to significances of 0.2 and 0. Throughout this investigation we will use a significance level of 0.05 as our cutoff - that is, we will assume a classifier is learning something if its performance is better than that of a random classifier at least 95% of the time. In this example, we can conclude that the classifier becomes significant for p between 5 and 20.

Once we have significant errors for a range of p values, we can fit our mean and 25th and 75th percentile curves. Recall that our chosen functional form is

$$e(p) = ap^{-\alpha} + b. \tag{5.10}$$

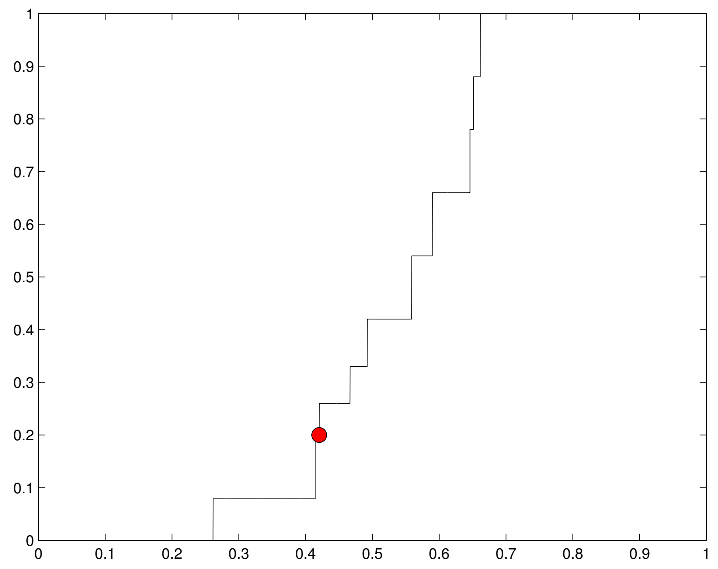


Fig. 5.2: Empirical distribution of random errors for classifiers using just 5 training points (out of 200) from the toy dataset. The x-axis gives the error rate and the y-axis gives the probability that the error is less than x. The red circle gives the error rate without permuting the labels

Fitting the parameters of this from a training set of $\{(p_1, \bar{e}_{p_1}), \dots, (p_r, \bar{e}_{p_r})\}$ pairs is found by minimising

$$\min_{\alpha,a,b} \sum_{i=1}^{r} (a p_i^{-\alpha} + b - \bar{e}_{p_i})^2 \tag{5.11}$$

subject to

$$\alpha, a, b \geq 0. \tag{5.12}$$

This optimisation is not straightforward as there are many minima. However, the problem becomes convex when b is fixed. Therefore, with b fixed, we can take logs and solve the equivalent problem

$$\min_{\alpha,a,b} \sum_{i=1}^{r} (\log(a) - \alpha \log(p_i) + \log(b - \bar{e}_{p_i}))^2 \tag{5.13}$$

subject to the same constraints. To find the full solution we simply solve this problem for various values of b and then perform a line search.

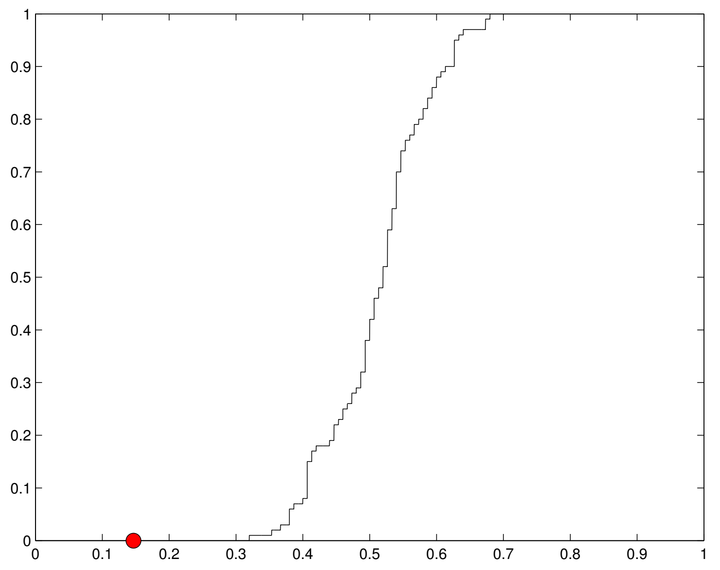


Fig. 5.3: Empirical distribution of random errors for classifiers using 50 training points (out of 200) from the toy dataset. The x-axis gives the error rate and the y-axis gives the probability that the error is less than x. The red circle gives the error rate without permuting the labels

Therefore, our complete methodology for a two-class problem is:

1. Sub-sampling and significance testing

   (a) Sub-sampling

      i. For a dataset D, with P_{c_1} and P_{c_2} samples from classes 1 and 2 respectively, select 10 training set sizes $p_1, \ldots, p_j, \ldots, p_{10}$ over the interval $[10, P-10]$.

      ii. For each training set size, randomly sample p_j samples from class 1 and class 2 subject to the requirement that $\frac{p_{c_1}}{p_{c_2}} \approx \frac{P_{c_1}}{P_{c_2}}$. Repeat this T_1 times. Call the i^{th} train test split for size p, S_{pi}.

      iii. Train and test a classifier on each split dataset S_{pi} and call the corresponding error e_{pi}.

   (b) Permutation test

i. For each of the S_{pi} train test splits, randomly permute the training labels T_2 times. Call these new datasets S_{pij}^{ran}.

ii. Train a classifier on each of these datasets and evaluate its error. Call this error e_{pij}^{ran}.

(c) Significance Calculation

i. For each training set size p, construct the empirical distribution defined by 5.9.

ii. From this distribution function, evaluate the smallest value of p for which a significance level of s is achieved (typically $s = 0.05$).

2. Learning Curve Generation

(a) For each p above the minimum p calculated above, calculate the mean error $\bar{e}_p$ and the 25th and 75th quantiles.

(b) Using the set $\{(p_1, \bar{e}_{p_1}), \ldots, (p_r, \bar{e}_{p_r})\}$ of sample sizes and average errors, fit the learning curve by

(c) In the same manner, using the set of sample sizes and quantiles, fit the curves corresponding to the 25th and 75th quantiles.

5.4 Application to Cancer Classification Problems

To illustrate the procedure described in the previous section, it has been applied to eight different cancer classification problems. These problems fall into two broad categories - discrimination between tumour morphologies (e.g, tumour versus normal or between different diseases) and predicting treatment outcomes. The former is generally considered an easier classification task. As such, we find that statistical significance is reached for small sample sizes and therefore even the smaller of these datasets are sufficiently large to enable construction of a learning curve. The latter problems tend to be much harder and require presentation of many more samples to reach statistical significance. Sadly, the number of samples in the dataset is often barely larger than the number required to reach significance and therefore the number of points that can be used to fit the curve is very small. In these cases, it can be useful to relax the methodology slightly and fit the curve with points that are not statistically significant. Results obtained in this manner should be treated with caution although may still be useful in an exploratory context. Although it didn't happen for any of the eight datasets here, it is possible that statistical significance will never be reached with the samples available. This may be due to the difficulty of the problem and with more samples, significance could be reached. However, it may also be that the measurements being taken do not convey the information required to perform the classification. Either way, fitting a learning curve is irrelevant as it is not clear whether any learning is in fact taking place. Indeed, it is not clear whether any learning could *ever* take place. In the next few sections, we describe and interpret the results obtained for each of our eight datasets. Table 5.2 summarises the learning curve parameters obtained for each dataset and table 5.3 comments on the results obtained.

Dataset	**25th Quartile**	**Mean**	**75th Quartile**
Cancer vs. normal	4.7%	7.3%	8.9%
Leukemia, AML vs. ALL	0.6%	2.5%	3.4%
Colon, Cancer vs. normal	4.8%	9.0%	12.4%
Ovarian, Cancer vs. normal	0.03%	1.2%	1.8%
Lymphoma, BLBC vs. Follicular	0%	0.9%	1.5%
Brain, outcome	14.5%	16.1%	23.8%
Lymphoma, outcome	11.2%	17%	24.3%
Breast, outcome	24.3%	32.3%	40.4%

Tab. 5.1: Extrapolated error rates at 400 samples

Dataset		a	α	b
Cancer vs. normal	25th quantile	1.89	-0.63	0.003
	Mean	1.42	-0.52	0.0010
	75th quantile	1.17	-0.43	0.000
Leukemia, AML vs. ALL	25th quantile	39.8	-2.3	0.0006
	Mean	0.771	-0.65	0.009
	75th quantile	0.884	-0.6	0.010
Colon, Cancer vs. normal	25th quantile	0.439	-0.37	0.00
	Mean	0.480	-0.28	0.00
	75th quantile	0.500	-0.23	0.00
Ovarian, Cancer vs. normal	25th quantile	10.7	-1.76	0.00
	Mean	0.736	-0.69	0.00
	75th quantile	0.995	-0.67	0.00
Lymphoma, BLBC vs. Follicular	25th quantile	8.99	-2.48	0.00
	Mean	0.57	-0.70	0.001
	75th quantile	0.671	-0.64	0.00
Brain, outcome	25th quantile	1.06	-0.37	0.004
	Mean	1.12	-0.33	0.006
	75th quantile	0.980	-0.24	0.00
Lymphoma, outcome	25th quantile	1.23	-0.41	0.008
	Mean	0.943	-0.30	0.01
	75th quantile	0.872	-0.21	0.00
Breast, outcome	25th quantile	0.532	-0.14	0.01
	Mean	0.485	-0.07	0.01
	75th quantile	0.429	-0.01	0.00

Tab. 5.2: Summary of learning curve parameters

Dataset	**Size**	**Size for stat. sig.**	**Was curve fitted with sig. data**	**Maximum size used in fit**	**Actual error**	**Extrapolated error**	**Difference**	**Comments**
Cancer vs. normal	280	15	yes	210	7.0%	8.6%	2.6%	A low error rate is achieved and exatrapolation is reasonably accurate. Suggests error rates < 0.07 are achievable with > 400 samples.
Leukemia, AML vs. ALL	73	10	yes	35	0.0%	5.6%	5.6%	A low error rate is achievable, extrapolation is quite pessimistic.
Colon, Cancer vs. normal	62	10	yes	50	16.3%	15.2%	1.1%	Error rate of 0.16 is achieved, dataset is large enough for accurate extrapolation. Lower error rates < 0.09 may be achievable with > 400 samples.
Ovarian, Cancer vs. normal	54	10	yes	40	5.6%	4.8%	0.8%	Low error rate is achieved, extrapolation is accurate. It may be possible to get ~ 0 error for larger dataset sizes.
Lymphoma, BLBC vs. Follicular	53	5	yes	40	4%	4.7%	0.7%	Low error rate is achieved and extrapolation is accurate. Additional samples may reduce error to ~ 0.
Brain, outcome	60	45	no	40	22%	29.6%	7.6%	Dataset is too small for problem difficulty. Learning curve is inaccurate but suggests error rates < 0.14 may be possible with > 400 samples.
Lymphoma, outcome	58	50	no	40	23%	29.5%	6.5%	Dataset is too small for problem difficulty. Learning curve is inaccurate but suggests possible error rates < 0.17 with > 400 samples.
Breast, outcome	78	65	no	70	30%	36.3%	6.3%	Dataset is too small for problem difficulty. Learning curve is inaccurate but suggests error rates < 0.32 with > 400 samples.

Tab. 5.3: Summary of results and comments

5.4.1 Tumour versus Normal tissue

The first dataset is ([59]) consists of 16,063 expression measurements for 280 samples. 180 of these samples are from different primary tumours (for example, prostate, lung etc) and 100 healthy samples from the same tissues. A Support Vector Machine (SVM) (see chapter 2 for details and references) with a linear (i.e. dot product) kernel was used. We used all of the 16,063 features in the numerical experiments. Using our significance test, we find that statistical significance is reached at 15 training samples.

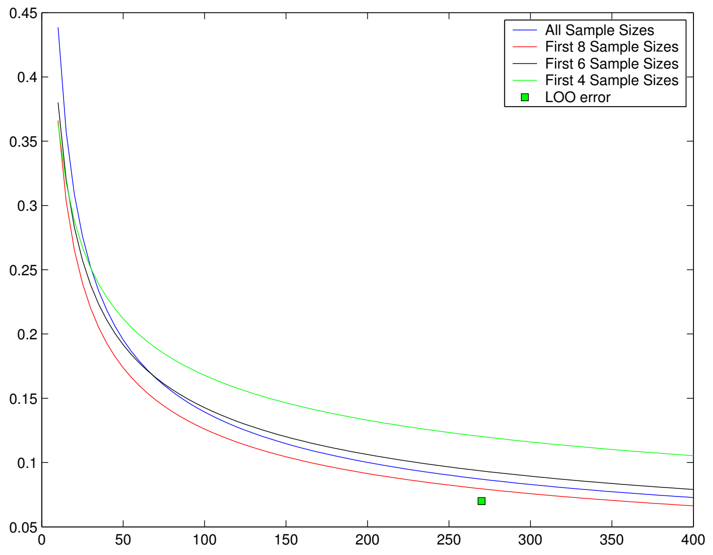


Fig. 5.4: Learning curves for the normal versus tumour data as different numbers of training samples are used.

We are now able to create our learning model for this dataset. Error rates were estimated for 9 different values of n ($n = 30, 40, 50, 60, 80, 90, 130, 170, 210$) as well as a leave-one-out error computed using all of the 280 samples.

Different numbers of these sizes were then used to construct learning curves to investigate how the learning curve estimates improve as the size of the training set increases. Four curves were created using a) the error rates for all training set sizes, b) the first 8 training set sizes, c) the first 6 training set sizes and d) just the first 4 sizes. The

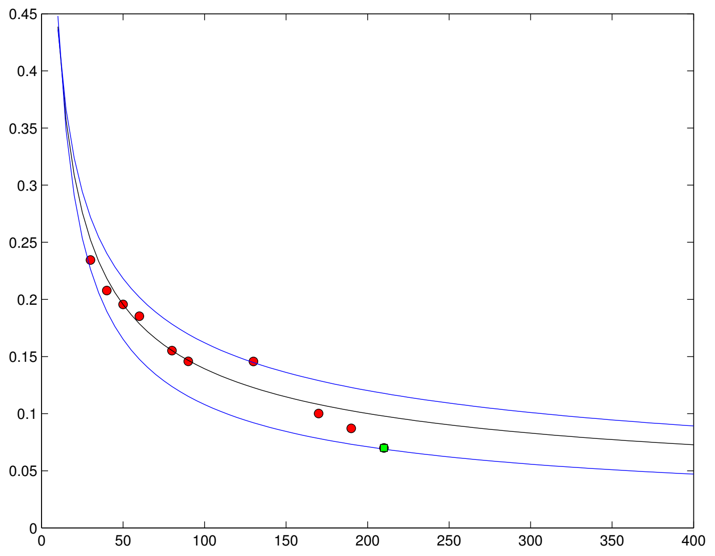


Fig. 5.5: Learning curve and empirical data for tumour versus normal data. Learning curve has been learnt with all possible sample sizes.

resulting curves are shown in figure 5.4 along with the leave one out error with all 280 samples. As expected, as more and more training set sizes are used to fit the curve, the model improved - i.e. the predicted error rate at 279 training samples approaches the leave one out error. When all sizes are used, the actual leave-one-out error is only approximately 2% less than the the predicted error. A plot of the learning curve obtained when all sample sizes are used, and the empirical data that was used to fit the curve can be seen in figure 5.5.

When all training set sizes are used to construct the curve, the fitted error rate expression is given by

$$e(p) = 1.42p^{-0.52} + 0.0098. \tag{5.14}$$

For the 25th and 75th quantiles, the respective curves are given by

$$e_{25}(p) = 1.89p^{-0.63} + 0.0032 \tag{5.15}$$

$$e_{75}(p) = 1.17p^{-0.43} + 0.000. \tag{5.16}$$

We can draw several conclusions from these curves. Firstly, the decay rate α is approximately 0.5 suggesting that this problem is reasonably difficult to learn. However,

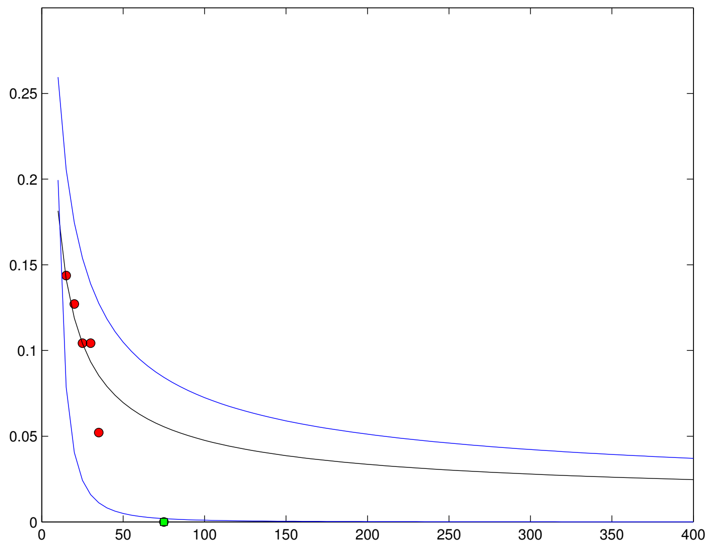


Fig. 5.6: Learning curves for the leukemia morphology data. Red circles show the empirical error rates, the black curve gives the mean error curve, the blue curves show the 25th and 75th percentiles and the green square shows the LOO error.

the Bayes error terms are very close to zero suggesting that given enough samples, very good performance is possible. The size of the envelope defined by the quantiles is approx $\pm 2\%$ indicating that the model is reasonably accurate. Approximations for larger sample sizes can be obtained by substituting the required values of n into the curve equations. For example, for $n = 400$ we have

$$e(400) = 7.3\% \tag{5.17}$$

$$e_{25}(400) = 4.7\% \tag{5.18}$$

$$e_{75}(400) = 8.9\%. \tag{5.19}$$

Therefore, the achievable error using 400 samples is $7.3 \pm 2.6\%$.

5.4.2 Leukemia morphology

In this dataset [23] we have 48 samples of acute lymphoblastic leukemia (ALL) and 25 samples of acute myeloid leukemia (AML). For each sample, we have 7,129 expression values. The classifier used was a Support Vector Machine with a linear

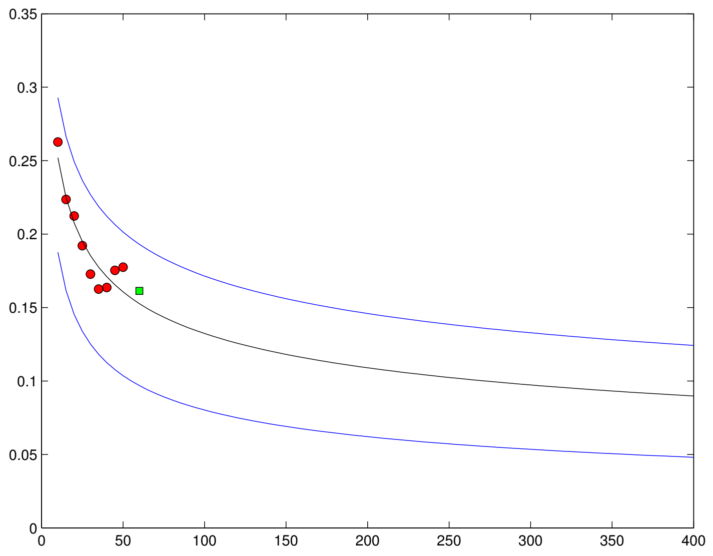


Fig. 5.7: Learning curves for the leukemia morphology data. Red circles show the empirical error rates, the black curve gives the mean error curve, the blue curves show the 25th and 75th percentiles and the green square shows the LOO error.

kernel and all 7,129 of the features were used. The sizes of training sets used were $n = 10, 15, 20, 25, 30, 35$ and only 5 samples were required to reach our 5% statistical significance level. Figure 5.6 shows the empirical error values and the learning curves for this dataset. The learning curve estimated from the empirical errors is

$$e(p) = 0.7706p^{-0.65} + 0.009. \tag{5.20}$$

The decay and learning rates in this example show that the classifier learns more quickly than in the previous example. This, coupled with the smaller number of samples required to reach significance suggests a much easier problem. In fact, as can be seen from the 25th quantile in figure 5.6, it is possible to get zero test error at about 73 samples. The quantile envelope is a lot larger than in the previous dataset - probably due to the fact that we are using a much smaller range of data set sizes to fit our model.

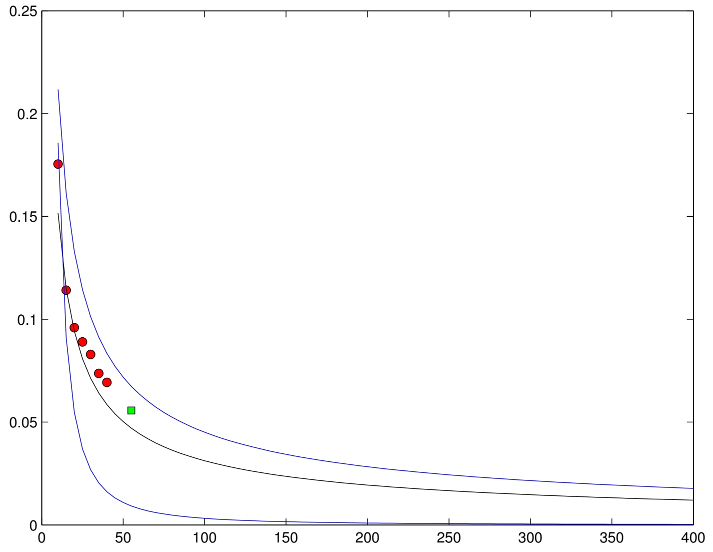


Fig. 5.8: Learning curves for the leukemia morphology data. Red circles show the empirical error rates, the black curve gives the mean error curve, the blue curves show the 25th and 75th percentiles and the green square shows the LOO error.

5.4.3 Colon cancer

The colon cancer dataset [53] consists of 22 normal samples and 40 samples corresponding to malignant tissue. For each sample, we have 2,000 expression values. Unlike most other datasets, this data was subjected to the following pre-processing step - firstly, the natural logarithm was taken and then an hyperbolic-tangent function was applied to reduce the influence of outliers. This step was applied by the originators of the dataset. Again, the classifier used was a Support Vector Machine with a linear kernel and all 2,000 features were used. Estimates of errors at $n = 10, 15, 20, 25, 30, 35, 40, 45, 50$ and the learning curves along with the leave-one-out error are shown in figure 5.7. Our 5% significance level is achieved at approximately 10 samples and the functional form of the fitted learning curve is

$$e(p) = 0.4798p^{-0.2797}. \tag{5.21}$$

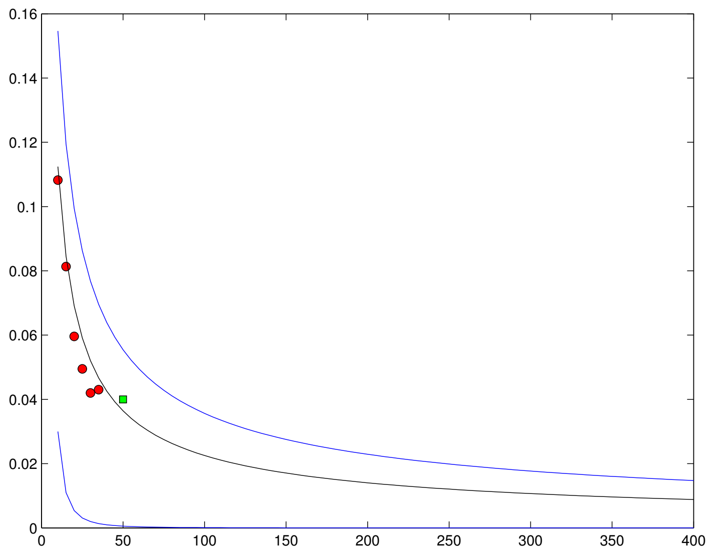


Fig. 5.9: Learning curves for the lymphoma morphology data. Red circles show the empirical error rates, the black curve gives the mean error curve, the blue curves show the 25th and 75th percentiles and the green square shows the LOO error.

5.4.4 Ovarian cancer

This dataset [66] consists of 24 samples from healthy ovarian tissue and 30 from malignant tissue. There are 1,536 expression levels for each sample. Pre-processing for this dataset involved adding 1 and then taking the natural logarithm of the expression values. All 1,536 features were used and the classifier was again a linear Support Vector Machine. As with colon cancer, statistical significance was reached at approximately 10 samples and $n = 10, 15, 20, 25, 30, 35, 40$ training sets were used to fit the learning curve. Figure 5.8. The error function is given as

$$e(p) = 0.7362p^{-0.6864}. \tag{5.22}$$

5.4.5 Lymphoma morphology

The final morphology dataset [4] consists of 24 diffuse large B-cell lymphoma samples and 12 samples of follicular and chronic lymphocyptic. Each sample has 18,432 measured expression values and the data was pre-processed by taking the $\log_2$ of each

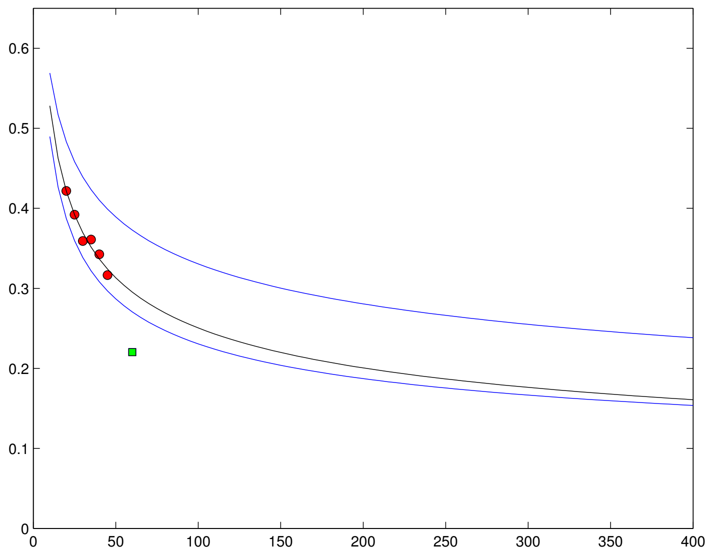


Fig. 5.10: Learning curves for the brain treatment outcome data. Red circles show the empirical error rates, the black curve gives the mean error curve, the blue curves show the 25th and 75th percentiles and the green square shows the LOO error.

expression value. This is a standard technique as it preserves the symmetry between over and under expression. A linear support vector machine was used to classify the data and all 18,432 features were used. Training set sizes of $n = 10, 15, 20, 25, 30, 35$ were used and significance was reached for $n \geq 5$. Figure 5.9 shows the empirical errors and learning curves for this dataset. The learning curve is given by

$$e(p) = 0.57p^{-0.7073} + 0.0006. \tag{5.23}$$

5.4.6 Brain cancer treatment outcome

This dataset consisted of samples from patients with childhood medulloblastomas [58]. 39 of the samples were from patients who reacted favourably to treatment (still alive after two years) whilst the other 21 were from patients with a poor treatment outcome. For each sample, 7,129 expression values were available. A linear support vector machine was used with 150 features chosen by the radius-margin criteria [10][1]. Training

[1]The set of features is chosen to be that set that minimises the radius margin bound on the number of leave-one-out errors

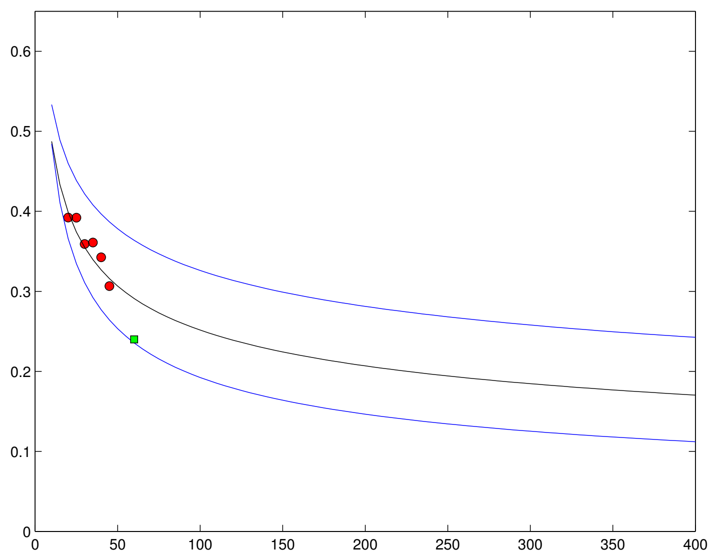


Fig. 5.11: Learning curves for the lymphoma outcome data. Red circles show the empirical error rates, the black curve gives the mean error curve, the blue curves show the 25th and 75th percentiles and the green square shows the LOO error.

sizes of $n = 20, 25, 30, 35, 40$ were used to estimate errors.

Statistical significance was reached with 45 training samples - almost the entire dataset, and a larger size than the sizes for which the error rate was estimated. Therefore, we cannot use the methodology as described so far. However, we can cautiously apply a learning curve to the non-significant values as can be seen in figure 5.10. As one might expect, the learning curve is quite inaccurate - it overestimates the leave-one-out error by about 7%. The learning curve is given by

$$e(p) = 1.115p^{-0.3295} + 0.006. \tag{5.24}$$

5.4.7 Lymphoma treatment outcome

The samples making up this dataset all came from patients with diffuse large cell lymphoma (DLCL) [68]. 32 of the samples were from patients with successful treatment outcomes whilst the other 26 samples were from patients with poor treatment outcome. Each sample consists of 7,129 expression values. A linear support vector machine was

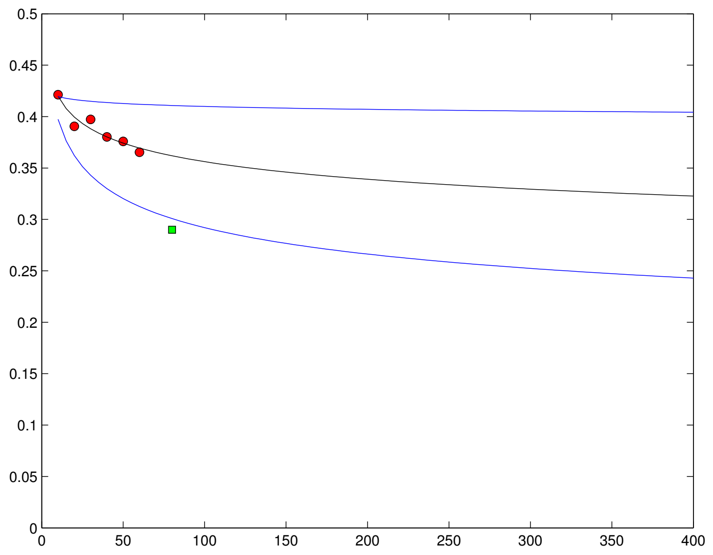


Fig. 5.12: Learning curves for the leukemia morphology data. Red circles show the empirical error rates, the black curve gives the mean error curve, the blue curves show the 25th and 75th percentiles and the green square shows the LOO error.

used with 150 features again chosen by the radius-margin criteria [10]. Training set sizes of $n = 20, 25, 30, 35, 40$ were used. As with the Brain cancer data, statistical significance was not reached until nearly all of the data set was used (in this case, $n = 50$). The learning curve is shown in 5.11 and it is given by

$$e(p) = 0.9431p^{-0.2957} + 0.01 \qquad (5.25)$$

as expected, the curve is not very accurate and overestimates the error at 57 samples (i.e. the leave-one-out error) by 9%.

5.4.8 Breast cancer treatment outcome

The final dataset contains expression levels of 34 samples from patients with breast tumours that metastasized within five years of disease onset and the remaining 44 from patients that were disease free for at least five years [75]. Each sample consisted of 24,624 expression values and again, a linear support vector machine was used to classify. Sample sizes of $n = 10, 20, 30, 40, 50, 60, 70$ were used and statistical significance was achieved at approximately 65 samples. Again, true use of our methodology is not possible but we can fit the learning curve using all values as above. The learning curve can be seen in 5.12 and it can be seen from this plot that the curve is not particularly accurate - overestimating the leave-one-out error by more than 6%. The functional form of this curve is given by

$$e(p) = 0.4852p^{-0.0733} + 0.01. \tag{5.26}$$

5.5 Using learning curves to compare experimental methods

In the previous section we investigated how our methodology could be used to predict classification performance. Here we use it to compare classification performance between data generated with microarrays and that generated with a new experimental technique, CESH [45]. For a brief introduction to the CESH technique, see 4.9.1. This work has previously been published in [46] and a more comprehensive description of the CESH method and the datasets used can be found there.

5.5.1 CESH v Microarray

Unfortunately, as it is such a new technique, the volume of CESH data available is small compared to that for microarray. Also, there are no datasets available where samples from the same tissues have been analysed using both microarrays and CESH. This makes direct comparison between the two methods impossible. However, we can compare projected error rates for CESH with those results obtained in the previous section to get some idea of how much performance we are losing with the CESH method.

We applied our methodology to three CESH datasets. The first consisted of classifying between rhabdomyosarcomas, leiomyosarcomas and prostate cancer, the second looked at the sub-classification of rhabdomyosarcomas into alveolar and embryonal and the final dataset involved predicting the treatment outcome of patients with Wilms tumours.

Table 5.4 shows the learning curves and predicted error rates obtained in these investigations. Comparison of these values with those in the previous section shows us that the performance with CESH is comparable to that obtained using microarrays. Although far from conclusive, this is a very promising result and suggests that creating and analysing further CESH datasets would certainly be worthwhile.

Dataset	a	α	b	$e(400)$
Rhabdo, Leio, Prostate	1.172	-0.575	0	3.739×10^{-2}
Alveolar vs. embyonal	0.655	-0.321	0	9.572×10^{-2}
Wilms outcome	0.747	-0.544	0	2.869×10^{-2}

Tab. 5.4: Learning curves and error approximations for the CESH datasets.

5.6 Conclusions

In this chapter, we have introduced a novel methodology for estimating the statistical significance of microarray classification and predicting classification performances if larger datasets were available. To illustrate its use, we have applied it to eight microarray datasets depicting two distinct problems. The first five datasets involved morphological classifications - e.g. tumour versus non-tumour or discrimination between different tumour types. The second set involved predicting the outcome of treatments. From the results of the analysis, it can be seen that the inverse power law for modelling generalisation as a function of sample size fits the empirical errors reasonably well. Particularly for the morphology examples, the error extrapolation is quite accurate whilst for the treatment outcome, the accuracy decreases although results are still useful in an exploratory sense. This decrease in performance is most likely due to the increased complexity of the problems and limited dataset sizes. As one might expect, the more training set sizes used to fit the learning curves increases their accuracy.

It is generally thought that the differences between classes in morphological classification problems are much greater than those in the treatment outcome problems. This is shown in our results through the number of samples required to reach statistical significance. In the morphological examples, significance was always easily reached with 10-20 samples whereas in the outcome examples, 40-50 examples were typically required. To be able to extrapolate reasonably well, the morphological datasets typically required 30-40 training samples while the treatment outcome problems would typically need somewhere in the region of 75-100 samples.

Learning curves for the outcome problems were generally characterised by decay rates close to zero whilst for the morphology problems, these values tended to be greater than 0.5 - again reinforcing the view that treatment outcome problems are harder to learn. Along with the Bayes error, these decay rates can be used to compare the difficulty of the problems. Figure 5.13 shows Bayes error versus the decay rate for the datasets used here. Although the number of datasets studied is relatively small, we see that the treatment outcome problems (blue) tend to be situated towards the right of the plot and have higher inverse decay rates. This demonstrates the increased difficulty present in these problems.

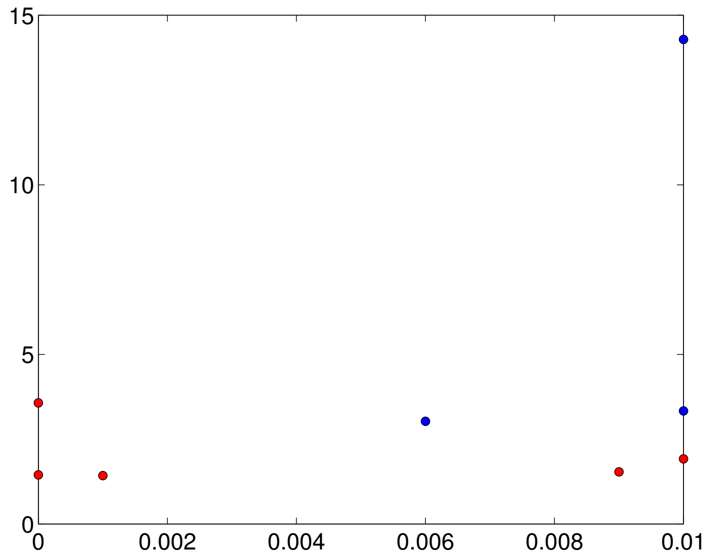


Fig. 5.13: Bayes error rate (x-axis) versus inverse of decay rate (y-axis) for morphology problems (red) and treatment outcome problems (blue).

To summarise, our proposed methodology provides reasonably accurate predictions of data set size requirements for cancer classification problems. In this sense, it can be used as a rule-of-thumb when designing an experiment or be used once an experiment has been completed to get an idea of whether or not it is worthwhile collecting more data. It can also be used to evaluate the relative complexity of a classification task and give some indication of how much noise is present in the data. In all of our examples, we have used the support vector machine as our classifier but there is no reason why the methodology presented could not be applied to any one of a number of different classification algorithms and there is no reason to limit this approach to simply classifying microarray data. Evaluating the learning curves for a variety of algorithms and data types would provide useful insights into the efficiencies of both the algorithms and the data types. Finally, we have used a new expression measurement tool (CESH) to show how the learning curve methodology can be used to compare different data types.

Chapter 6

Inferring Regulatory Networks

6.1 Introduction

The continued development of high-density microarrays in recent years is leading to an explosive growth in gene expression data. This data provides measurements of the expression of thousands of genes simultaneously and promises to be a very important tool in furthering the understanding of organisms at a cellular level. Most analysis of microarray data thus far has been interested in discovering differentially expressed genes and grouping cell samples based on their expression profiles. For example, Bhattacharjee et al [6] used hierarchical clustering to cluster microarray measurements for various human lung carcinomas and from this clustering discovered previously unknown sub-types and genes that were differentially expressed between them. Spellman et al [71] used microarrays to discover 800 yeast genes that are regulated by the cell cycle. Recently, however, there has been a shift towards trying to understand how the synthesis of proteins is regulated and how this synthesis is affected by external stimuli.

Regulation can occur at most of the distinct stages present in protein synthesis. However, the transcription of mRNA is possibly the most influential and is itself affected by proteins that bind themselves to the promoter regions upstream of particular genes. Such complex interactions between groups of genes and the proteins they synthesise are generally described as genetic regulatory networks. It is such networks that we are interested in reverse engineering from the levels of mRNA measured in microarray experiments.

The standard approach to modelling a known network is through sets of coupled differential equations. These equations describe how the concentrations of the various products evolve over time. Although probably the most realistic model of the actual reactions taking place, such a model requires knowledge of the network structure and the various reaction rates. At the other end of the spectrum, the most simple approach is to cluster genes by their expression profiles. Genes with similar expression profiles are likely to regulate each other or be regulated by another shared parent gene. A major drawback of this approach is that it is impossible to infer any notion of causality - which gene is regulated and which gene is regulating.

There is a growing body of literature that takes a middle ground between these two extreme levels of detail. Friedman et al [19] was the first such attempt and used the framework of Bayesian networks to model regulatory networks in yeast. Due to the small number of arrays relative to the number of genes, a bootstrapping procedure was used to extract features (i.e. regulatory interactions) prominent in the data. The work provided some promising results but the authors concluded that in order to infer networks from expression data alone, much larger datasets were required. Other, more complicated Bayesian network models have been proposed. For example, Rangel et al [61] introduce the concept of a set of hidden state variables, producing a Kalman filter type model. Some non Bayesian network approaches have also been proposed. Yeung et al [82] used singular value decomposition to produce a family of possible networks and then performed a robust regression step to extract the best one (in terms of sparsity). Similar methods have also been proposed by Weaver et al [79] and Zak et al [83, 84].

Some of these publications evaluate their algorithms on real data although this does not make objective assessment very easy as knowledge of the true network is incomplete or not known at all. One in particular that stands out is that of Husmeier [32] in which the author evaluates the performance of his Bayesian network model on realistic simulated data and is therefore able to make objective conclusions. We will follow this procedure in this chapter. A useful review of various approaches for reverse engineering regulatory networks can be found in [12].

With a few exceptions, approaches based on Bayesian networks have required that the expression data be discretised. This is in contrast to e.g. [82] where the network is inferred directly from the continuous expression data. There are several inherent problems with the discretisation step. Firstly, it is not clear how many discrete states should be used. Too few and too much information could be lost. Too many can make the task of learning the Bayesian network infeasible. On the other hand, most models for continuous data proposed so far have been based on explicitly linear formulations - i.e. the expression of a particular gene is modelled as a linear combination of the expression of the genes that regulate it. This type of model will therefore only be able to capture dependencies between genes that are close to being linear.

Another important issue is that of sparsity. Biologically, it is thought that regulatory networks are sparse. That is, the number of genes regulating any particular gene is small. In a Bayesian network framework, this sparsity can be easily introduced in the form of a prior on the number of connections into a particular gene. In fact, such a prior assumption can make the learning task considerably more straightforward as the space of possible networks is reduced. For most of the continuous models proposed, it is not obvious how to obtain such sparsity. Ad hoc techniques such as pruning network connections have been proposed but without knowledge of the network structure, it is unclear at what level a threshold should be set. In this chapter, we propose two separate approaches that crucially both work directly with continuous expression data. Firstly, we look at a recently proposed sparse Bayesian algorithm that provides a linear network and we then investigate a possible nonlinear distribution that would enable a continuous, nonlinear Bayesian network approach.

The remainder of this chapter is organised as follows. In the next section, we provide a very brief introduction to Bayesian networks. In section 3 we introduce the Bayesian network model that we shall use to model regulatory networks. Section 4 introduces the linear regression model used and section 5 introduces the simulated network from which we shall create data. Test results are presented in section 6. Section 7 describes a possible nonlinear conditional distribution and in section 8 we finish with some conclusions. As mentioned in the introduction, this chapter represents work currently in progress and although some results are presented it is primarily a pointer to future research directions.

6.2 Bayesian Networks

Bayesian networks provide a representation of the joint probability distribution over a set of random variables. This representation consists of two components. A *directed acyclic graph* (DAG), the vertices of which correspond to the random variables and a set of conditional probability distributions that describe the behaviour of each variable given its immediate parents.

The structure of the graph describes the conditional independency relationships present in the distribution. Edges connect *parents* with their *children* and given its set of parents, a particular child node is conditionally independent of all other (i.e. those not belonging to the set of its parents) nodes. The conditional independence properties of a distribution over a set of random variables $\mathcal{X} = [X_1, \ldots, X_n]$ allow us to factorise it into the following product form

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|\mathrm{Pa}_i) \tag{6.1}$$

where Pa_i defines the set of nodes that are parents of X_i.

It is important to note that more than one graph can imply the same set of conditional independence relationships. Graphs that imply the same relationships are said to belong to the same equivalence class. This equivalence class can be described by a *partially directed graph* (PDAG) where directed edges correspond to directed edges that appear in all members of the equivalence class an undirected edges correspond to edges that appear in all members but not always in the same direction.

6.2.1 Training Bayesian Networks

The task of training a Bayesian network varies greatly, depending on the characteristics of the training data and of the distribution being modelled. In our case, the training data is complete and the structure (i.e. the set of regulatory parents for each gene) is unknown. Once a form for the conditional distributions has been chosen, the training process amounts to performing a search over possible structures. Generally, this will involve defining a scoring function for candidate structures and then finding the structure that optimises this score. One popular scoring function is based on the posterior probability of a graph given the training data (if no prior knowledge is imposed on the structure of the graph, maximising this score is essentially a maximum likelihood approach).

An important property of the conditional independences in the distribution is that the score used to evaluate the suitability of different graphs is generally decomposable to the sum of the local contributions from each node. That is, we can optimise the local structure around each node (i.e. its set of parents) individually. This property is highly desirable in the task of inferring regulatory networks as it means that we can decompose the problem of finding a complete network into one of finding the best set of parent genes for each gene.

6.3 A Bayesian Network Model for Regulatory Networks

We wish to be able to reverse engineer regulatory networks from observed expression data. In modelling a regulatory network with a Bayesian network, we treat the expression level of each gene as a random variable and hence a node in our network. As described earlier, a Bayesian network is made up of a DAG and a set of conditional probability distributions. Unfortunately, a DAG cannot include any cyclic loops, something that is commonly found in regulatory networks. To overcome this problem, we use a dynamic Bayesian network. This requires duplication of each node where one set of nodes represents the expression at a certain time t (i.e. the input) and the other set represents the expression at time $t+1$ (the output). It is worth briefly discussing the notion of time. The training samples need not belong to some complete time series - one of the implicit assumptions that we have made is that each input/output pair is independently and identically distributed. Hence, the pairs could correspond to points in some time series or could be just the results of samples under different conditions, for example, the result of gene knockout experiments.

We must now define the form of the conditional probability distributions. In order to make this choice, we must decide whether we wish to use continuous expression values, or discretise them. Discretisation causes a loss in information and also opens up several new questions - how many discrete states should be defined over the range of the values? Too few and we may not be able to capture the behaviour of the network. Too many and the number of parameters in our network will increase such that the computations required will become infeasible. For these reasons we choose to work with continuous values.

The choice of continuous values implies that once we decompose our problem into the task of inferring several local networks, we are left with what is essentially a regression problem for each gene. In the proceeding sections we will always consider just one local network - i.e. we are interested in finding the network connections and hence the potential parents of one particular gene. This would then be repeated for each gene to obtain the complete network.

6.3.1 A Linear, continuous conditional distribution

Following Friedman et al [19], we begin by looking at the most simple form of regression model for our conditional distributions. For the expression of a particular gene, y, we assume the following functional form for its expression at time t given the expression of the other genes (its parents, in Bayesian network terms) x_j ($j = 1 \ldots n$) at time $t-1$. Notice that if we are considering gene j then x_j is actually y at $t-1$ but we have introduced the x notation to avoid the need for time and gene indices.

$$y = \sum_{j=1}^{n} w_j x_j + \epsilon \tag{6.2}$$

where w_j is the strength of the connection between into our gene from gene j. Ideally we would like a sparse solution reflecting the fact that each gene is only regulated by a small fraction of the total number of genes and hence many of the w_j terms will be zero. In Bayesian network terms, we can define the set of parents of our current gene Pa to be the set of all genes j for which $w_j > 0$. The ϵ term models the measurement noise and, modelling this as a zero mean Gaussian, with variance σ^2, implies the following distribution for y.

$$y \sim \mathcal{N}(\mathbf{w}^T\mathbf{x}, \sigma^2) \tag{6.3}$$

where $\mathcal{N}(a, b)$ represents a normal distribution with mean a and variance b. We have also introduced the vector notation of $\mathbf{x} = [x_1, \ldots, x_n]^T$ and $\mathbf{w} = [w_1, \ldots, w_n]^T$.

More generally, if we have T observations of the expression of the genes, we can consider the following multi-variate distribution

$$\mathbf{y} \sim \mathcal{N}(\mathbf{X}^T\mathbf{w}, \sigma^2\mathbf{I}) \tag{6.4}$$

where $\mathbf{y} = [y_1, \ldots, y_T]^T$ and $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]$.

The learning task now involves using the observed data to infer the weight vector $\mathbf{w}$. Notice that this includes both learning steps described above for Bayesian networks. Using Bayes law, we can compute a posterior distribution of these weight values given the observed data

$$p(\mathbf{w}|\mathcal{D}, \sigma^2) = \frac{p(\mathcal{D}|\mathbf{w}, \sigma^2)p(\mathbf{w})}{\int p(\mathcal{D}|\mathbf{w}, \sigma^2)p(\mathbf{w})d\mathbf{w}} \tag{6.5}$$

where $p(\mathbf{w})$ reflects our prior belief in the values of the weights and we use $\mathcal{D}$ to represent the whole dataset. $p(\mathcal{D}|\mathbf{w},\sigma^2)$, when viewed as a function of $\mathbf{w}$ for fixed training data is often called the *likelihood*, and assuming that all training examples are independently and identically distributed can be factorised as follows

$$p(\mathcal{D}|\mathbf{w},\sigma^2) = \prod_{m=1}^{T} p(y_m|\mathbf{w},\sigma^2,\mathbf{x}_m). \tag{6.6}$$

If we make no prior assumptions about weight values (i.e. $p(\mathbf{w}) = 1$) we can find a closed form solution for the value of $\mathbf{w}$ that maximises the log of equation 6.5. Due to the quadratic term in the likelihood, this solution is simply the least squares solution and is given by

$$\mathbf{w} = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{y}. \tag{6.7}$$

As there are no prior restrictions on the values in $\mathbf{w}$, it is often found that such a solution is prone to severe over-fitting and typically, all weight values will be non-zero. For example, see figure 6.3. Therefore our parent set Pa will include all possible genes and we have inferred a fully connected network that provides little information. This can be partially avoided by putting a prior on $\mathbf{w}$ that penalises large weight values and hence favours simpler, smoother functions. A suitable prior is a zero mean Gaussian

$$p(\mathbf{w}|\alpha) \sim \mathcal{N}(\mathbf{0},\alpha^{-1}) \tag{6.8}$$

where, for future simplicity, we have defined the *accuracy* α which is equivalent to the reciprocal of the variance.

As all distributions are Gaussians, we can still find a closed form solution for equation 6.5

$$\mathbf{w} = (\mathbf{X}\mathbf{X}^T + \alpha\mathbf{I})^{-1}\mathbf{X}\mathbf{y}. \tag{6.9}$$

This particular form of solution will prevent over-fitting. Generally however, the solution will still include many weights that are close to zero and hence our parent set Pa will still include all possible genes. For example, see figure 6.4. Sparsity could be achieved by pruning all weight values that are close to zero. However, it is not clear at which level the weights should be pruned. As we prune weights, we are removing edges from our Bayesian network and changing our representation of the joint distribution. Therefore, it would seem that the choice of threshold is crucial. It would be far more convenient if this sparseness could be achieved automatically during model training. We will see how this can be achieved in the next section.

6.3.2 Sparse Bayesian Regression

The standard method for learning connections in a Bayesian network is based upon starting with a random structure, evaluating a scoring function and then make greedy changes to the structure - i.e. changes that most increase the scoring function. van Someren et al [74] used such an approach with the form of model discussed above. Incorporating sparseness in this system is trivial as it amounts to simply putting a limit on the number of edges entering a particular node and only evaluating possible changes to structure that do not cause a violation of this constraint. However, there are problems with this method. Firstly, the greedy search routines can fall into local minima. This can be overcome by using less-greedy approaches (e.g. accepting changes that decrease the score with non-zero probability) or by sampling networks from the posterior distribution. Secondly, the problem in its exact form is known to be NP-hard. Imposing a prior on the number of edges into a particular node significantly reduces the search space but it still rapidly becomes infeasible as the number of genes reaches the order of thousands - something common in current microarrays. However, we notice an interesting parallel between this structure determination problem and the fast marginal likelihood maximisation algorithm proposed by Tipping and Faul [72].

Recall that in the last section we overcame the problem of over-fitting by introducing a Gaussian prior distribution on the weights of our network. We can extend this idea to introduce sparseness by using a separate Gaussian prior for each weight (i.e. potential parent gene), each with it's own accuracy parameter α_i. Assuming independence across the genes we can take the product of these individual priors to give

$$p(\mathbf{w}|\boldsymbol{\alpha}) = \prod_{i=1}^{n} p(w_i|\alpha_i) \tag{6.10}$$

where $p(w_i|\alpha_i) = \mathcal{N}(w_i|0, \alpha_i^{-1})$. We are now interested in evaluating the following posterior probability of our weight vector $\mathbf{w}$

$$p(\mathbf{w}|\mathbf{y}, \boldsymbol{\alpha}, \sigma^2) = \frac{p(\mathbf{y}|\mathbf{w}, \sigma^2)p(\mathbf{w}|\boldsymbol{\alpha})}{p(\mathbf{y}|\boldsymbol{\alpha}, \sigma^2)} \tag{6.11}$$

where we have omitted the dependence on X for clarity. Again, because all of the distributions are Gaussian, this posterior is also Gaussian with mean and covariance given by

$$\boldsymbol{\Sigma} = (\mathbf{A} + \sigma^{-2}\mathbf{X}^T\mathbf{X})^{-1} \quad \boldsymbol{\mu} = \sigma^{-2}\boldsymbol{\Sigma}\mathbf{X}^T\mathbf{y} \tag{6.12}$$

where $\mathbf{A}$ is a diagonal matrix of the α_i values. We could extend this model by applying prior distributions to the α_i but as these distributions will be non-Gaussian, we can no longer obtain a tractable posterior. Therefore, Tipping and Faul suggested approximating them with point estimates obtained my maximising the log marginal likelihood

$$\log p(\mathbf{y}|\boldsymbol{\alpha}, \sigma^2) = \log \int_{-\infty}^{\infty} p(\mathbf{y}|\mathbf{w}, \sigma^2)p(\mathbf{w}|\boldsymbol{\alpha})d\mathbf{w}. \tag{6.13}$$

Again, all of these distributions are Gaussians and so again, we can obtain this likelihood in closed form which can be maximised with respect to $\boldsymbol{\alpha}$. This form is

$$\log p(\mathbf{y}|\boldsymbol{\alpha}, \sigma^2) = -\frac{1}{2}[T \log 2\pi + \log |\mathbf{C}| + \mathbf{y}^T\mathbf{C}^{-1}\mathbf{y}], \tag{6.14}$$

where

$$\mathbf{C} = \sigma^2\mathbf{I} + \mathbf{X}\mathbf{A}^{-1}\mathbf{X}^T. \tag{6.15}$$

When we perform this optimisation, we observe that many of the α_i values approach infinity. This suggests that the corresponding posterior distributions for the w_i are very sharply peaked at zero and hence this particular candidate parent gene can be pruned from the model.

The individual priors on the weights provides us with the required sparsity but at what cost? In computing the marginal likelihood during optimisation, we must invert $\mathbf{C}$, an operation of $\mathcal{O}(n^3)$. When we have a large number (n) of genes, this becomes an enormous computational burden. To overcome this, Tipping and Faul noticed that the marginal likelihood covariance $\mathbf{C}$ could be decomposed into separate components for each α_i (i.e., each gene) as follows

$$\begin{aligned}\mathbf{C} &= \sigma^2\mathbf{I} + \sum_{j\neq i}\alpha_j^{-1}\mathbf{x}_j\mathbf{x}_j^T + \alpha_i^{-1}\mathbf{x}_i\mathbf{x}_i^T && (6.16)\\ &= \mathbf{C}_{-i} + \alpha_i^{-1}\mathbf{x}_i\mathbf{x}_i^T && (6.17)\end{aligned}$$

where $\mathbf{C}_{-i}$ represents the covariance matrix with gene i removed. Substituting this into equation 6.14 we find that it is possible to decompose the log marginal likelihood into additive contributions from the individual α_i components. For a more description, the reader is referred to [72]. Essentially, we can define a sparsity factor s_i and a quality factor q_i for each of our potential parents as follows

$$s_i = \mathbf{x}_i^T\mathbf{C}_{-i}^{-1}\mathbf{x}_i \quad q_i = \mathbf{x}_i^T\mathbf{C}_{-i}^{-1}\mathbf{y}. \qquad (6.18)$$

Analysis of the decomposed likelihood reveals that it has a unique maximum with respect to each individual α_i, defined as follows

$$\begin{aligned}\alpha_i &= \frac{s_i^2}{q_i^2 - s_i}, \quad \text{if } q_i^2 > s_i && (6.19)\\ \alpha_i &= \infty, \quad \text{if } q_i^2 \leq s_i. && (6.20)\end{aligned}$$

The implications of these results is that if gene i is currently a parent (i.e. $\alpha_i < \infty$) but $q_i^2 \leq s_i$ then we should remove gene i from the parent set in order to increase the marginal likelihood (i.e. set α_i to ∞). Secondly, if gene i is not currently a parent (i.e. $\alpha_i = \infty$) but $q_1^2 > s_i$, we should add it to the model by setting $\alpha_i = s_i^2/(q_i^2 - s_i)$. Using these straightforward update rules, we are able to add and remove genes to the set of parents whilst always increasing the marginal likelihood until convergence.

6.3.3 Sequential Sparse Learning Algorithm

Following [72], we can formalise the above results into the following algorithm

1. Initialise σ^2.

2. Start with a single candidate parent $\mathbf{x}_i$, and set α_i as follows

$$\alpha_i = \frac{||\mathbf{x}_i^2||}{||\mathbf{x}_i^T \mathbf{y}||^2/||\mathbf{x}_i||^2 - \sigma^2} \tag{6.21}$$

3. Compute s_i and q_i for *all* n possible parents.

4. Select a new candidate parent $\mathbf{x}_i$ from all n possible parents.

5. Calculate $\theta_i = q_i^2 - s_i$.

6. If $\theta_i > 0$ and $\alpha_i < \infty$ (i is already a parent), re-estimate α_i.

7. If $\theta_i > 0$ and $\alpha_i = \infty$ (i is not already a parent), add $\mathbf{x}_i$ to the set of parents and calculate α_i.

8. If $\theta_i \leq 0$ and $\alpha_i < \infty$, then i is currently a parent but should be removed, so remove $\mathbf{x}_i$ from the parent set and set $\alpha_i = \infty$.

9. Update s_m and q_m.

10. If convergence has been reached, continue to step 10, otherwise, return to step 4.

11. Calculate $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ using the final parent set.

More details and steps that can be taken to improve the efficiency of the algorithm can be found in [72].

6.4 Simulated Network

In order to evaluate and compare the performance of particular models, we follow the example of Husmeier [32] and generate data from a simulated network. Although not as realistic as using genuine expression data, this method has several advantages, most importantly that we know exactly what the network structure is that we are trying to model and hence can objectively evaluate and compare different models. We use the same network topology as Husmeier - a 12 gene network inspired by a small yeast sub-network, investigated by Friedman [19]. The network topology can be seen in figure 6.1. In this figure, each node represents a gene, solid edges represent activating

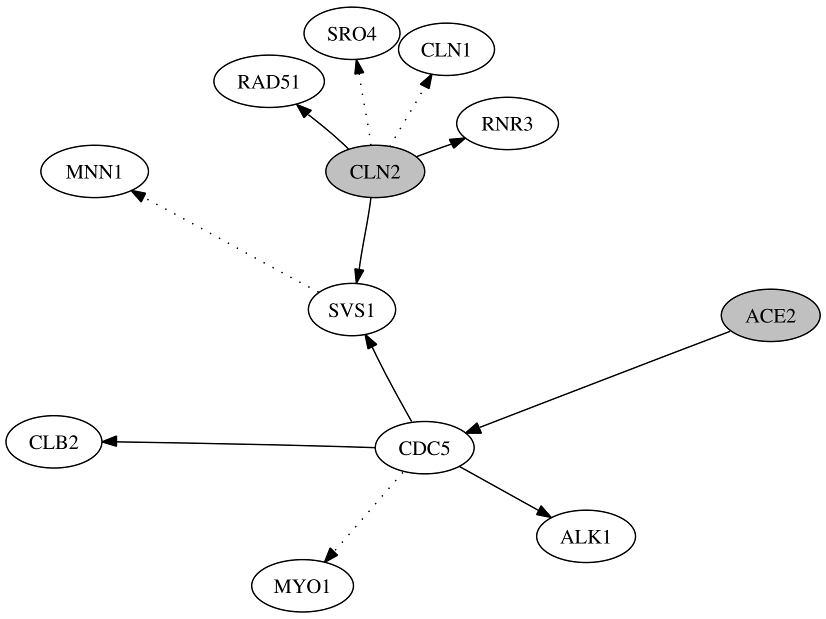


Fig. 6.1: Simulated network from [32]

regulations (and 'exclusive or' type regulations (in the case of the inputs to SVS1)) and dotted lines represent inhibitory regulations. Source genes (i.e. genes with no inputs) are shaded in gray. It will be easier in our analysis to refer to each gene by its node number rather than its name. A list of the genes with their corresponding node numbers can be seen in table 6.1. Husmeier used this network to generate discrete data. However, we are interested in continuous data and so have to modify the data generation process. To do this, we randomly choose two steady state values for each gene. The state a gene is in will stochastically depend on the state of its parents. For example, CLN1 will be in its higher state with probability 0.9 if CLN2 is in its higher state (see [32] for more details). The expression value for a particular gene is then drawn from a Gaussian distribution with mean equal to the current steady state level for that gene.

Gene	**Number**
RNR3	1
CLN1	2
SRO4	3
RAD51	4
CLB2	5
MYO1	6
ACE2	7
ALK1	8
MNN1	9
CLN2	10
SVS1	11
CDC5	12

Tab. 6.1: Genes and their associated node numbers.

6.5 Results

Figures 6.3, 6.4 and 6.4 show graphically the weight matrices generated when least squares, regularised least squares (least squares with single Gaussian prior) and the sparse Bayesian approach are applied to a 100 sample time series generated from the simulated network. In each figure, absolute weight values have been plotted as it is not easy to visualise both positive and negative weights on the same figure. The position of the actual connections are shown in figure 6.2. In all diagrams, the rows (i.e. up-wards from left to right) correspond to the weights into a particular gene. We notice that all three methods seem able to find these actual connections. However, it is in the number of false connections (false positives) that the sparse Bayesian approach comes out on top. We see that the sparse Bayesian approach manages to recover almost all of the actual connections with only two or three false positives. This is in comparison to the least squares and regularised least squares approaches that have false positives in almost every position. This distinction is incredibly important in inferring regulatory

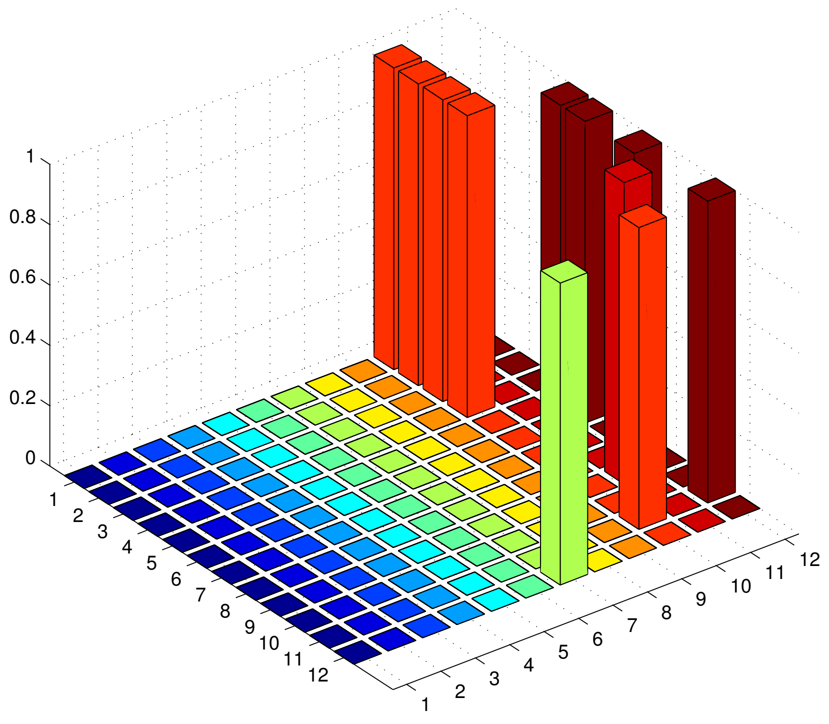


Fig. 6.2: Actual network connections for 12 gene simulated network given in figure 6.1 For ease of visualisation, absolute weight values are shown. The ith row corresponds to the weights into gene i from the other genes.

networks as we do not need to set any pruning threshold or use any bootstrap procedures to extract the significant edges.

If we do use an *adhoc* pruning procedure in an attempt to prune the false positives present in the least squares and regularised least squares approaches we can investigate whether it is possible to re-create the correct network from these methods, given the correct choice of threshold. Receiver operator characteristics (ROC) curves can be used to evaluate the performance of such a system where we are trying to balance the number of true positives and the number of false positives. To create such curves, we first count the number of true positive (TP), false positive (FP), true negative (TN) and false negative (FN) edges. We then calculate the sensitivity = TP/(TP+FN) and the complimentary specificity = FP/(TN+FP) and plot the resulting values. The larger the area under the resulting curve, the better the performance of the model. We

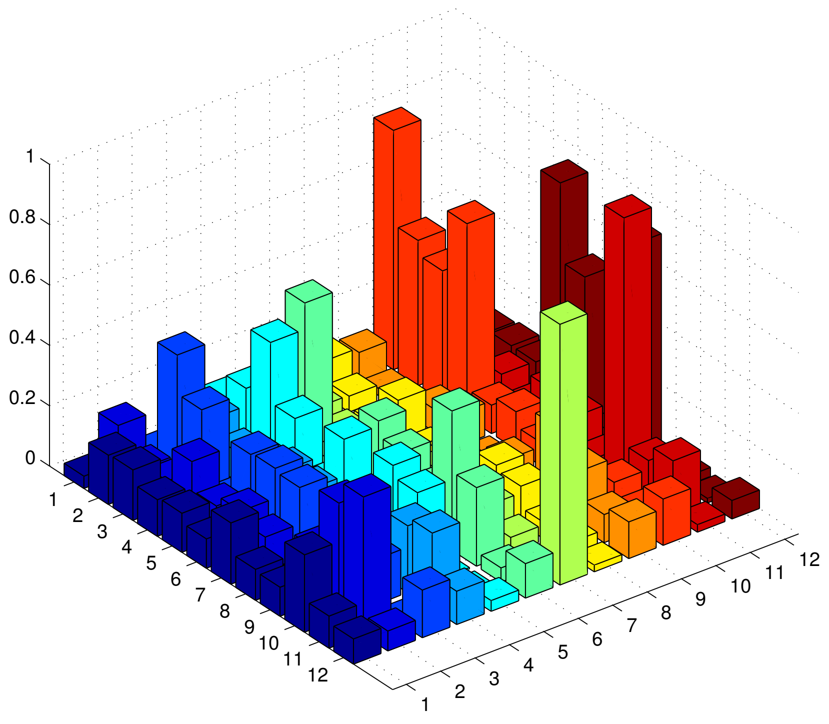


Fig. 6.3: Network weights inferred using least squares for 100 time points from the simulated network. For ease of visualisation, absolute weight values are shown. The ith row corresponds to the weights into gene i from the other genes.

can also evaluate the performance of a model by its highest value of sensitivity (y axis) for 0 complimentary specificity (x axis) as this is the highest number of correct edges we can recover with no false edges.

We can see such ROC curves for the three methods, over various data set sizes in figures 6.6, 6.7, 6.8 and 6.9. Firstly, we notice that the performance of all three systems decreases as the number of time points is decreased. It should also be pointed out that the sparse Bayesian ROC curves are not terribly informative as they are generally the result of only one or two points, due to the very low number of small weights. We also notice that the sparse Bayesian approach is almost always superior to the other approaches in the number of true positives recovered with no false positives. The fact that the least squares and regularised least squares approaches are able to get a higher number of true positives than the sparse Bayesian approach should always be taken in the context of the number of false positives and also the fact that in order to get this

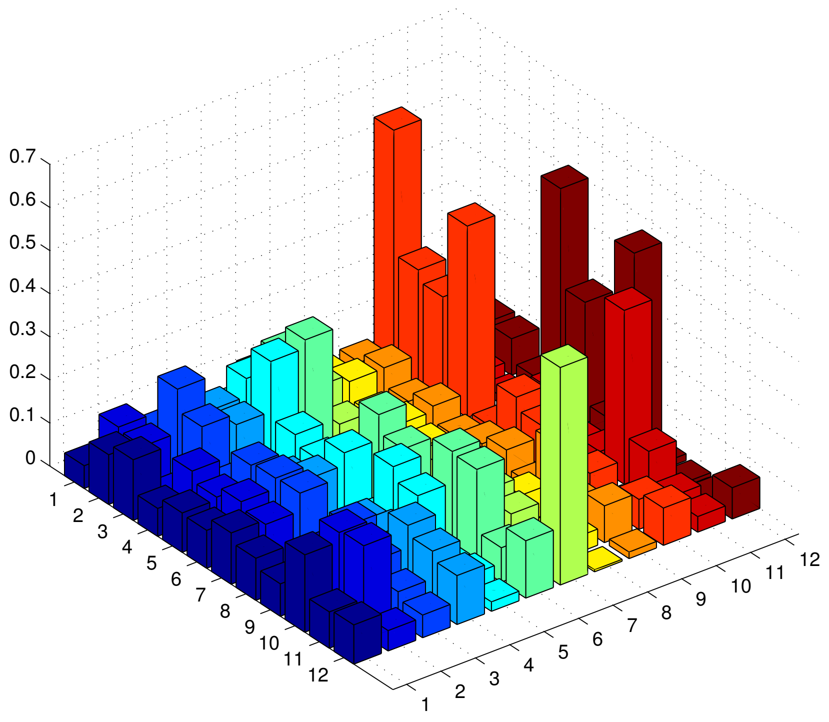


Fig. 6.4: Network weights inferred using regularised least squares for 100 time points from the simulated network. For ease of visualisation, absolute weight values are shown. The *i*th row corresponds to the weights into gene *i* from the other genes.

performance, some *adhoc* threshold needs to be set. To re-iterate this point, figure 6.10 shows the actual networks inferred by these models. The first one shows the network inferred by sparse Bayes. Comparison of this with figure 6.1 shows that all of the true connections are there except the ones between SVS1 and CDC5 and CLN2. In addition to this there are two additional connections - one between CDC5 and CLB2 and one between CLN1 and SVS1. Compare this with the second network that shows the result for regularised least squares with no thresholding and the difference is striking. Applying various thresholding values, we see the networks in the final two figures. In the first, the threshold has been set to 0.2 (that is, any weight value whose absolute value is below this is set to zero). Some structure has been recovered but there are a lot of false connections. Increasing the threshold to 0.4, we observe the final network. The structure around CDC5 has been recovered but little else. This example makes it clear how difficult it would be to select such a threshold level, without knowing in

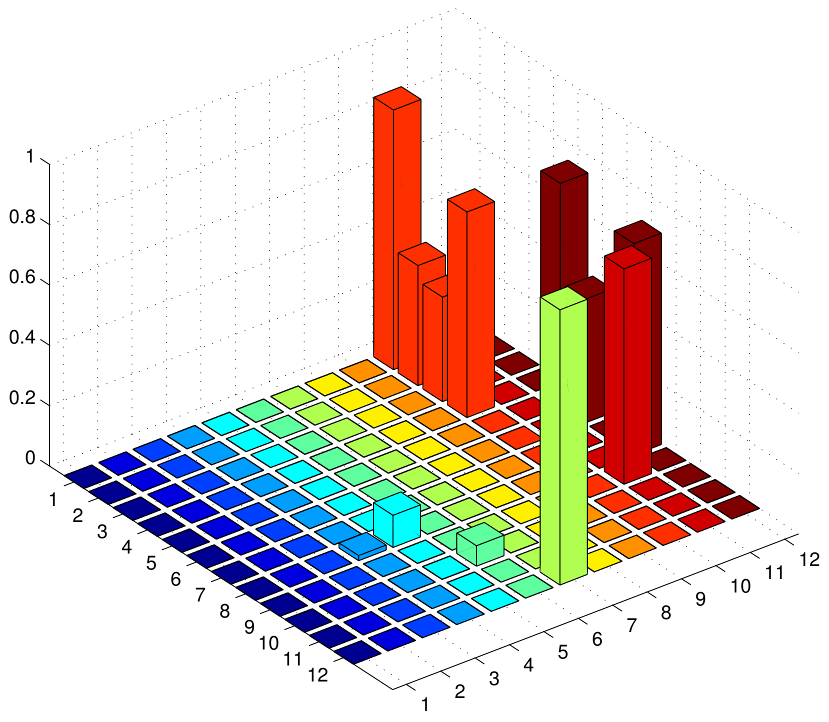
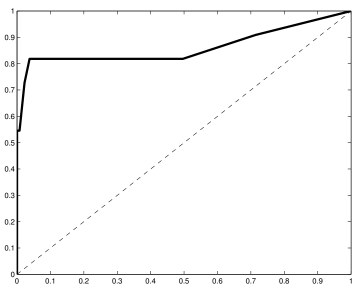


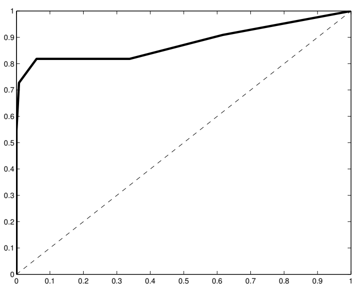
Fig. 6.5: Network weights inferred using the sparse Bayesian approach for 100 time points from the simulated network. For ease of visualisation, absolute weight values are shown. The ith row corresponds to the weights into gene i from the other genes.

advance what the topology of the network was.

6.6 A Non-linear conditional distribution

Let us investigate the results presented in the previous section more thoroughly. Specifically, why the sparse Bayesian approach performed so well but always failed to infer the connections into SVS1. The model used to create the data for this particular gene is different to that used for all the others. This has two inputs and Husmeier modelled this using an exclusive-or like relationship, i.e. the gene would be expressing if either one of its parents was expressing but not if both were. XOR is a non-linear function and our sparse Bayesian approach is based around a linear model. Therefore, it is not surprising that the model always failed to capture these dependencies. Previously, various people have tried to overcome this problem by capturing the non-linearities


(a) LS


(b) RegLS

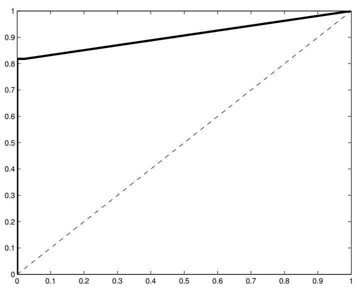
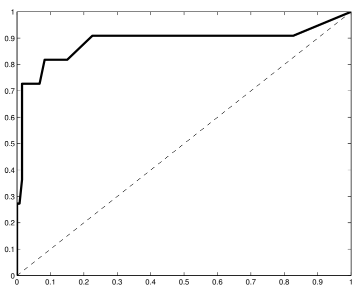
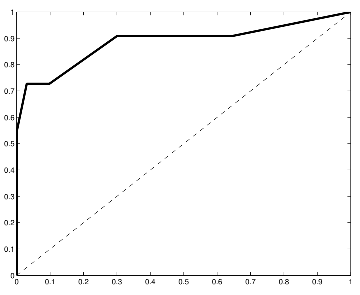

(c) Sparse Bayes

Fig. 6.6: ROC curves for 100 time points


(a) LS


(b) RegLS

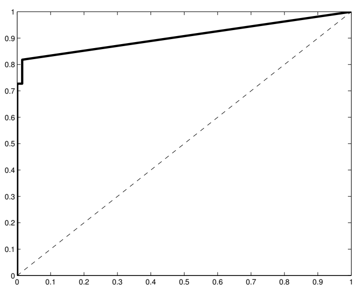
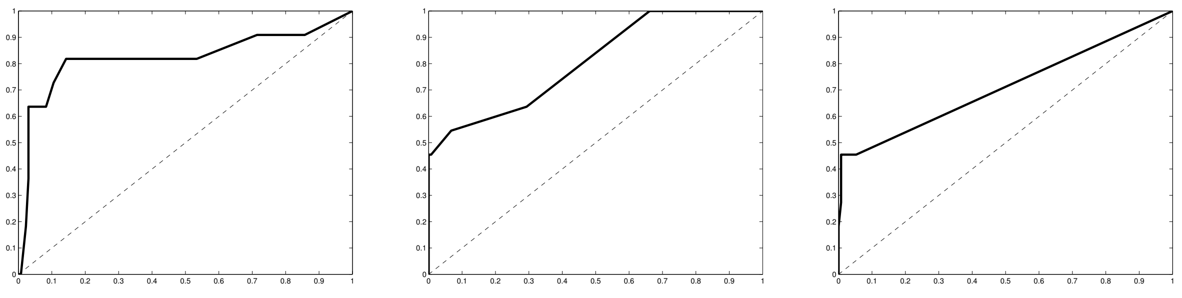

(c) Sparse Bayes

Fig. 6.7: ROC curves for 50 time points

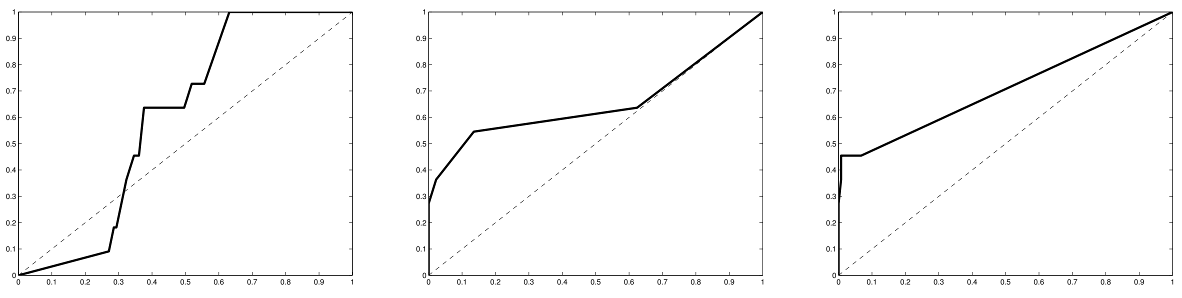
using an explicit non-linear formulation. For example, Imoto et al [33] and Kim et al [39] replaced the weighted linear representation with a weighted set of splines. The major drawbacks of such a parametric approach is that firstly we do not know what the correct parametric form is and secondly, the number of parameters that we need to learn from the data is increased dramatically. The sparse Bayesian algorithm described above can easily be kernelised and hence could be used to model a wide range of possible nonlinear functions. However, kernelising the algorithm results in sparsity in our training points (i.e. time points) and not features, making this approach unsuitable.

For these reasons, we propose using a semi-parametric approach based on the Parzen Window density estimator. The potential benefits of such an approach are that we



(a) LS (b) RegLS (c) Sparse Bayes

Fig. 6.8: ROC curves for 30 time points



(a) LS (b) RegLS (c) Sparse Bayes

Fig. 6.9: ROC curves for 15 time points

do not have to assume any particular form for the conditional distributions and only a minimal number of parameters have to be inferred from the available data. This method was introduced in the context of continuous, non-linear Bayesian networks by Hoffman and Tresp [28].

The Parzen Window method for empirical density estimation involves placing an isotropic Gaussian over each training point and then using the summation over all of these individual Gaussians as the estimate of the density. Therefore the width parameter for the Gaussians is the only parameter that needs to be set.

Recall that in the Bayesian network formulation, we require the distribution for the variable of interest conditioned on its parents. Denoting by Pa the set of parents, we can express this conditional distribution (using the product law) as follows

$$p(y|\mathrm{Pa}) = \frac{p(y, \mathrm{Pa})}{p(\mathrm{Pa})}. \tag{6.22}$$

We approximate the densities on the numerator and denominator with Parzen windows using our T training points

$$p(y|\mathrm{Pa}) = \frac{\sum_{t=1}^{T} p(y, \mathrm{Pa}|y^t, \mathrm{Pa}^t)}{\sum_{t=1}^{T} p(\mathrm{Pa}|\mathrm{Pa}^t)} \tag{6.23}$$

where all densities are Gaussians, with variance σ^2. Following Hoffman and Tresp, we need to define a prior on the number of connections in the network to induce the required sparsity. They suggest the following form

$$p(E) = \exp(-\alpha E) \tag{6.24}$$

where E is the number of edges, or can be thought of as the size of Pa.

The Sparse Bayesian algorithm described in the previous sections was developed around the basic linear regression model. This meant that given values for the hyper-parameters, all distributions of interest (and particularly the marginal likelihood) were Gaussians or convolutions of Gaussians and hence were analytically tractable. Using the Parzen window conditional distribution, this is no longer the case and we must resort to some kind of heuristic search over the possible network structures. In order to do this, we must define a network scoring function that we will aim to optimise. The scoring function that we will use here can be thought of as a form of likelihood measure. As for the sparse Bayesian approach, we consider each individual gene in turn and assume we have T training samples. The T expression values of the gene of interest are collated into a vector $\mathbf{y}$ and we use Pa to denote the current set of parents of this gene. For example Pa_t^i is the expression of the ith parent in the tth training example. The score we aim to maximise for each gene is then given as follows

$$p(\mathbf{y}, \mathrm{Pa}|\alpha) = p(\mathbf{y}|\mathrm{Pa}, \alpha)p(\mathrm{Pa}|\alpha). \tag{6.25}$$

Taking logs, we obtain the following expression that we wish to maximise with respect to the members of the set of parents

$$\begin{aligned}
\log p(\mathbf{y}, \mathrm{Pa}|\alpha) &= \sum_{\tau=1}^{T} \log \sum_{t=1}^{T} \mathcal{N}(y_\tau | y_t, \sigma^2) \prod_{i \in \mathrm{Pa}} \mathcal{N}(\mathrm{Pa}_\tau^i | \mathrm{Pa}_t^i, \sigma^2) && (6.26)\\
&- \sum_{\tau=1}^{T} \log \sum_{t=1}^{T} \prod_{i \in \mathrm{Pa}} \mathcal{N}(\mathrm{Pa}_\tau^i | \mathrm{Pa}_t^i, \sigma^2) && (6.27)\\
&- T\alpha E. && (6.28)
\end{aligned}$$

In order to search over possible structures, we first begin with all possible genes in the parent set. We then repeat the following iteration: Find the single change in the set (i.e. addition or removal of a parent) that gives the largest increase in the likelihood score and make this change. Once all candidate changes cause a decrease in the score, stop.

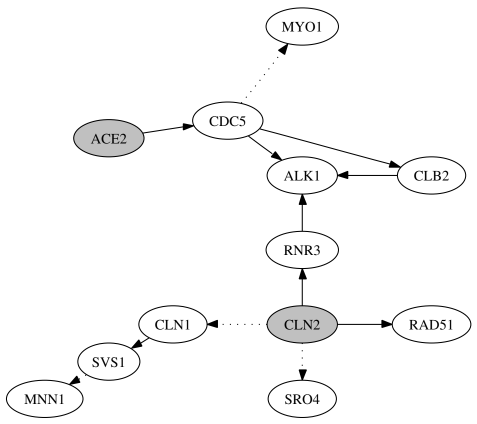
This greedy search is only guaranteed to reach a local minimum of the likelihood score and as such is not guaranteed to provide the best solution. However, many methods exist to improve such a strategy. For example, simulated annealing [40] could be used where changes that *decrease* the likelihood are accepted with non-zero probability. Generally, the literature on inferring regulatory networks using Bayesian networks has concentrated only on greedy search strategies. Investigating the possible improvements of a non-greedy method would be an interesting direction for future research.

There are also issues to be addressed regarding finding optimal values for the two hyper-parameters α (the prior constant) and σ^2 (the variance of the Gaussians that make up the Parzen window). In their original paper, Hoffman and Tresp [28] do not mention how to set this α value but do suggest a cross-validation procedure for σ^2. Some form of cross-validation procedure could be used, where some training samples are left out and are used to generate an un-biased approximation of the likelihood. However, it is likely that due to the small number of arrays, data will be at a premium and such a procedure is impractical. Therefore, at the moment, these two issues are still open and will be addressed in future work.

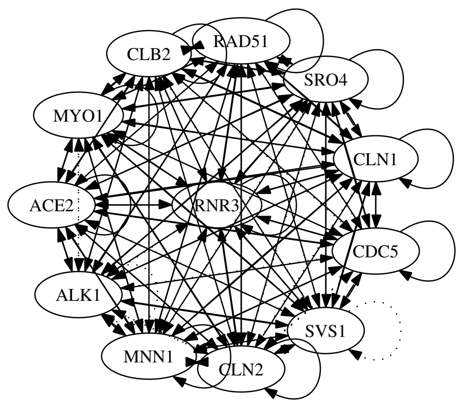
In order to evaluate whether this non-linear conditional distribution is able to infer the connections missed by the linear model we have evaluated its performance using data from the same simulated network. The result for 30 time points and $\alpha = 0.25$ can be seen in figure 6.11. We observe from this figure that the two connections that were always missing from the linear model are now present (the two connections into gene 11). This result looks very promising but it is worth remembering that this was achieved using hand-picked values of the two hyper-parameters and without knowledge of the actual network, this would not be possible. However, we are mainly interested here in whether or not this form of conditional distribution could discover these particular non-linearities and, as mentioned earlier, a rigorous method for choosing values of these hyper-parameters is a direction for future research.

6.7 Conclusions

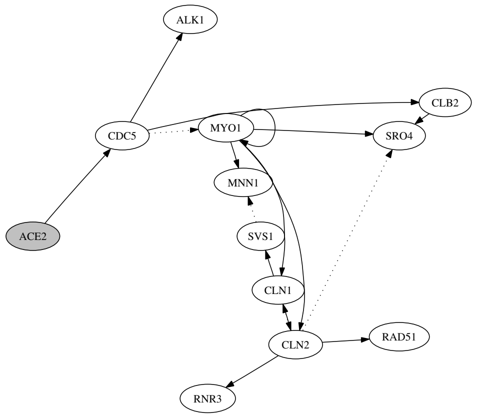
In this chapter we have investigated the problem of inferring regulatory networks from continuous expression data. We have looked at two possible algorithms. Firstly, the sparse Bayesian algorithm of Faul and Tipping that provides the desired sparsity but will never be able to capture nonlinear relationships in the data. In an attempt to overcome this inherent linearity, we have investigated the possibility of using a nonlinear conditional distribution based on the standard Parzen window approach to density estimation. This approach was able to infer the nonlinear relationships present in our simulated network but has several drawbacks that need to be overcome. The values of two hyper-parameters have to be inferred and it is presently unclear how this could be done given that the number of training points (i.e. microarrays) is generally very small making cross-validation out of the question. All testing has been done on simulated data from a small network suggested by Husmeier [32] and not on any real expression data. It is generally believed [19] that the size of current microarray datasets is insufficient to perform the task of inferring regulatory networks. Generally, experiments involving real data have produced results of questionable biological significance and the lack of knowledge of the actual underlying networks makes it very hard to truly evaluate the performance of a particular algorithm. Simulated data overcomes this problem although it is fairly obvious that simulating data in such a manner considerably simplifies the problem. Experiments with real datasets are an important direction for future research.



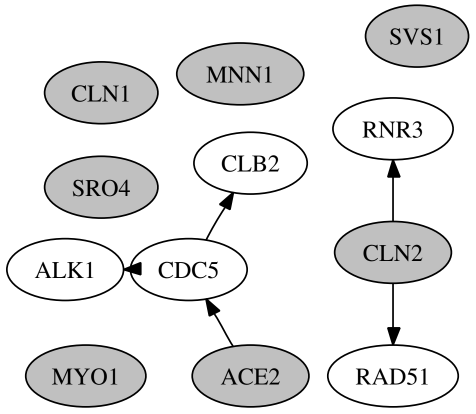
(a) Sparse Bayes



(b) RegLS, no threshold



(c) RegLS, threshold = 0.2



(d) RegLS, threshold = 0.4

Fig. 6.10: Inferred networks from the various models, with 100 time points and various threshold values

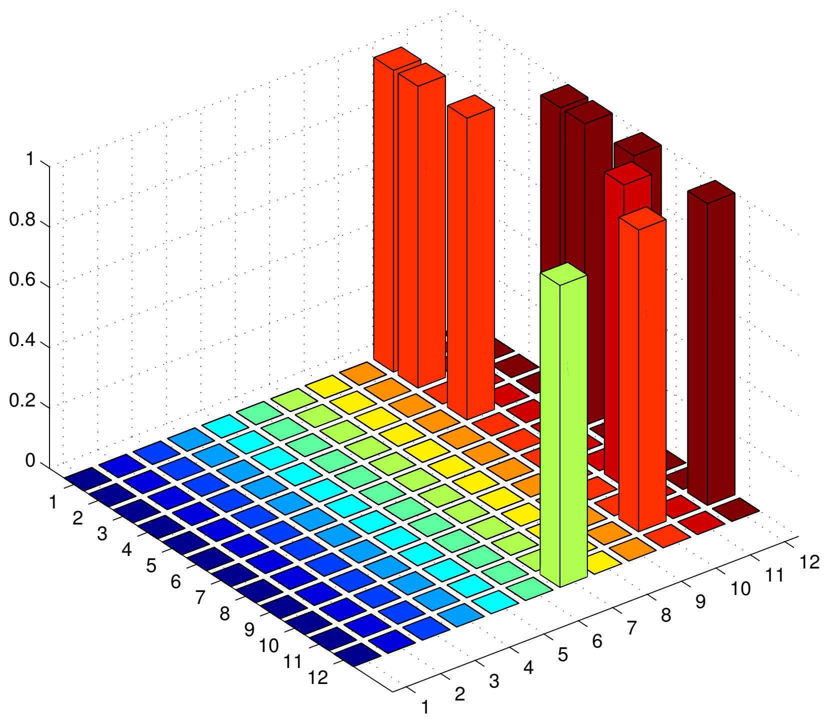


Fig. 6.11: Connections learned by Bayesian network with nonlinear conditional distribution with 30 training points.

Chapter 7

Conclusion and Future Directions

In this thesis, we have investigated various methods for the analysis of high dimensional microarray data. In this final section we will recap the work described in each chapter and reiterate the conclusions drawn. In addition, we will describe interesting future work directions.

7.1 Latent Process Decomposition

Firstly, Latent Process Decomposition - a generative probabilistic model that is able to extract biologically significant information from expression data. The LPD model inspired by the LDA model proposed by Blei et al [8] that modelled text documents as combinations of latent *topics*. In this chapter, we suggested an alternative to this bag of words style topic that would be applicable to microarray data. To demonstrate the power of LPD, we presented results on a variety of microarray datasets - several from cancer research where LPD was able to distinguish between various tumour types and two time series datasets for yeast where LPD was able to uncover real biological processes. The experimental results from this method appear very promising. Currently, LPD is being used to analyse new datasets, particularly those from breast and prostate cancer in collaboration with the institute for cancer research. In addition to this, various improvements to the basic LPD formulation are being developed. Most significantly, the brief investigation in this thesis on priors for the parameters suggested that the performance of the algorithm could be enhanced by sensible choices of priors and their associated hyper-parameters.

The prior we have provided for the variances was improper - that is, it does not integrate to 1 over its domain. Although this resulted in nice, tractable updates it is somewhat restrictive. For example, we may wish to incorporate a prior that penalises under-complexity as well as over-complexity. In addition to this, a proper prior allows us to use sampling approaches to learn the parameters of the network. Although the variational lower-bound optimisation approach appears to provide reasonable solutions within a reasonable amount of time (simulations generally take of the order of a few minutes), it is unclear how good a bound we are optimising. Given a proper prior, a sampling approach such as Metropolis Hastings could be used to sample from the posterior distribution over parameters. Implementation of a sampling method is currently being investigated. It is unclear how practical it will be given the very large number of parameters. There is one further advantage of a sampling based approach. Taking samples from the posterior would enable us to approximate the normalising constant in the Bayesian formulation - the *evidence*. This quantity can be used in model selection and may prove to be a more useful in evaluating $\mathcal{K}$, the number of processes than the currently implemented cross-validation techniques.

In appendix D we introduce the LPD model with a prior consisting of a mixture of Dirichlet's. Potentially, this provides the model with far greater flexibility. Although experiments with this prior on microarray data showed it to give little improvement over the standard, single Dirichlet prior, it may well prove useful when larger datasets become available.

Another possible extension to the LPD model is to augment it with a model for another form of biological data. For example, many probabilistic methods exist to model the DNA bases found in the upstream regions of a gene (the regions where proteins bind and affect transcription). Potentially, combining the LPD expression model with such sequence data would result in a model that could not only find genes that were similarly expressed, but also confirm whether or not these genes were co-regulated.

Finally, we briefly introduced an LPD model that would be suitable for discrete data such as that generated by CESH. Presently, there is not enough data available to realistically determine the potential of this model. However, more CESH data is becoming available and so this too will be an interesting avenue for future research. This also opens up the question of suitable priors for the multinomial parameters - something that is touched upon briefly in appendix E.

7.2 Estimating Sample Size Requirements

In this chapter, we introduced a novel methodology for assessing the significance of a particular classification performance and predicting how many microarray samples would be required to obtain a particular classification performance.

The proposed methodology first used a significance test based on random re-shufflings of the training labels to determine how many training samples were required to be able to classify at a significance level of 95%. Then, empirical error measurements were used to fit a *learning curve* that explained the relationship between the number of training samples and the generalisation error. This curve had the form

$$\epsilon(p) = ap^{-\alpha} + b \tag{7.1}$$

where a is known as the *learning rate*, α is the decay rate and gives an indication of the difficulty of the problem at hand. Finally, b is the *Bayes error* - the lowest error rate achievable.

There are several reasons why an experimentalist may wish to use such a methodology. Firstly, it provides a rigorous manner for determining whether or not a particular classification result is significant. Secondly, it enables the experimentalist to predict ahead to see how much the generalisation will improve given more training samples - this type of analysis could help decide whether collecting more samples is worth the considerable time and expense that would have to be expended.

Comparison of the learning curve parameters for different datasets gives an indication of the difficulties of the various classification problems and also allows the comparison of the noise levels present. In the experimental results presented here, we are able to support the belief that classifying between various tumour types or between tumour and normal samples is a much easier problem than classifications based on clinical data - for example, whether or not a patient suffers a relapse.

Particularly worth mentioning is how the method can be used to compare between different experimental techniques. So far only a comparison between microarrays and CESH has been performed but as more techniques are developed it will be interesting to compare their efficiencies in this manner. It would be interesting to perform a comparison between spotted and Oligonucleotide arrays - the design of Oligonucleotide arrays should make them less susceptible to noise, to what degree is this the case?

We are currently seeing an increase in the use of high throughput proteomic experiments. Such parallel measurement of protein abundances could in the future become a replacement for the mRNA measuring microarrays. After all, the protein is the end product and is what we are mainly interested in; mass spectrometry (the most widely used technique in proteomics) is also several orders of magnitude cheaper than microarray experiments. A comparison between the discriminatory power of the two approaches would be useful - such an analysis could readily be performed using the methodology described above.

7.3 Inferring Genetic Regulatory Networks

In this chapter, we have investigated the problem of inferring the regulatory interactions between genes from microarray data. Most attempts to do this in the literature have involved discretisation of the expression data and then the training of a Bayesian network. It is far from clear how many levels the data should be discretised into and where these levels should be. Therefore, we would like to be able to infer regulatory interactions from the original, continuous data. To this end, we investigated the sparse Bayesian regression algorithm of Tipping and Faul [72] which removed the need for a computationally expensive topology search. Such a search rapidly becomes infeasible as the number of genes in the investigation increases beyond small, un-realistic networks. However, the sparse Bayesian regression algorithm is inherently linear and cannot be expected to uncover more complicated, nonlinear relationships between genes.

There are many possible directions for future work in this area. The sparse Bayesian algorithm (also used in the context of microarrays by Li, Campbell and Tipping [44]) looks like it may be useful in this context although its inherent linearity may be a limiting factor. Unfortunately there is no obvious way to extend this elegant algorithm to a more flexible, nonlinear model. The Parzen window approach gets around the linearity problem but requires returning to a more ad hoc search procedure to find the network structure. This scales very badly and seriously questions whether or not it would be genuinely useful. Other nonlinear forms are possible. For example, Gaussian process priors could be used for the conditional distributions [20], however the structure determination problem is still present.

Using a kernel expansion in the sparse Bayesian algorithm does enable the network to model almost any nonlinear function. However sparsity is now achieved in training samples and not network connectivity. We are currently investigating using this method but weighting each possible parent in the kernel function and performing an additional optimisation over these weights. In this context, the sparse Bayesian algorithm provides sparsity over training (time) points. It is unclear whether or not this is a sensible thing to do, as the number of potential training points is already limited. It is currently too early to present any results from this investigation.

Various other open issues still exist. Experiments have been done with synthetic data - how do these algorithms perform when presented with real array data? Apart from the fact that a lot of real regulatory interactions are unknown, this opens other issues such as whether or not the algorithms can scale up to the large number of genes present. If they can, will the number of samples required scale up in a similar fashion? Finally, is it possible to use other forms of data as well as expression data? This could include anything from prior knowledge in the form of partial networks that are well understood to using sequence data to help narrow the search over possible parents.

Bibliography

[1] www.affymetrix.com.

[2] www.clopinet.com/isabelle/projects/svm/applist.html.

[3] C.J. Adcock. Sample size determination: a review. *Statistician*, 46(2):261–283, 1997.

[4] A.A. Alizadeh, M. Eisen, R. Davis, C. Ma, I. Lossos, A. Rosenwald, J. Boldrick, H. Sabet, T. Tran, Y. Xin, J. Powell, L. Yang, G. Marti, T. Moore, J. Hudson, L. Lu, D. Lewis, R. Tibshirani, G. Sherlock, W. Chan, T. Greiner, D. Weisenburger, J. Armitage, R. Warnke, R. Levy, W. Wilson, M. Grever, J. Byrd, D. Botstein, P. Brown, and L. Staudt. Different types of diffuse large b-cell lymphoma identified by gene expressing profiling. *Nature*, 403:503–511, 2000.

[5] A. Ben-Dor, L. Bruhn, N. Friedman, I. Nachman, M. Schummer, and Z. Yakhini. Tissue classification of gene expression profiles. *Journal of Computational Biology*, 7:559–583, 2000.

[6] A. Bhattacharjee, W. Richards, J. Staunton, C. Li, S. Monti, P. Vasa, C. Ladd, J. Beheshti, R. Bueno, M. Gillette, M. Loda, G. Weber, E. Mark, E. Lander, W. Wong, B. Johnson, T. Golub, D. Sugarbaker, and M. Meyerson. Classification of human lung carcinomas by mRNA expression profiling reveals distinct adenocarcinoma sub-classes. *Proc. Natl. Acad. Sci.*, 98:13790–13795, 2001.

[7] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1999.

[8] D.M. Blei, A.Y. Ng, and M.I. Jordan. Latent Dirichlet allocation. *Journal of machine learning research*, 3:993–1022, 2003.

[9] C-C. Chang, J-Y. Shih, Y-M. Yeng, J-L. Su, B-Z. Lin, S-T. Chen, Y-P. Chau, P-C. Yang, and M-L. Kuo. Connective tissue growth factor and its role in lung adenocarcinoma invasion and metastasis. *Journal of the National Cancer Institute*, 96(5):364–375, March 2004.

[10] O. Chappelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46:131–159, 2002.

[11] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, B*, 39:1–38, 1977.

[12] P. D'haeseleer, S. Liang, and R. Somogyi. Genetic network inference: from co-expression clustering to reverse engineering. *Bioinformatics*, 16(8):707–726, 2000.

[13] S.M. Dhanasekaran, T.R. Barrette, D. Ghosh, R. Shah, S. Varambally, K. Kurachi, K.J. Pienta, M.A. Rubin, and A.M. Chinnaiyan. Delineation of prognostic biomarkers in prostate cancer. *Nature*, 412:822–826, August 2001.

[14] R. Dietrich, M. Opper, and H. Sompolinsky. *Advances in Large Margin Classifiers*, chapter 20, Support Vectors and Statistical Mechanics. MIT press, 2000.

[15] R. Diterich, M. Opper, and H. Sompolinsky. Statistical dynamics of support vector networks. *Phys. Rev. Letters*, 82:2975–2978, 1999.

[16] M.B. Eisen, P.T. Spellman, P.O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proc. Natl. Acad. Sci USA*, 95:14863–14868, December 1998.

[17] A. Engel and C. Van den Broeck. *Statistical mechanics of learning.* Cambridge University Press, 2001.

[18] R. Fahmy, C. Dass, L-Q. Sun, C. Chesterman, and L. Khachigian. Transcription factor egr-1 supports fgf-dependent angiogenesis during neovascularization and tumor growth. *Nature Medicine*, 9(8):1026–1032, August 2003.

[19] N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using Bayesian networks to analyze expression data. *Journal of Computational Biology*, 7(3/4):601–620, 2000.

[20] N. Friedman and I. Nachman. Gaussian process networks. In *Proceedings of the 16th Annual Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 211–219, San Francisco, CA, 2000. Morgan Kaufmann Publishers.

[21] E. Garber et al. Diversity of gene expression in adenocarcinoma of the lung. *Proceedings of the national academy of sciences of the USA*, 98(24):12784–13789, 2001.

[22] A. Gasch, P. Spellman, C. Kao, O. Carmel-Harel, M. Eisen, G. Storz, D. Botstein, and P. Brown. Genomic expression programs in the response of yeast cells to environmental changes. *Molecular Biology of the Cell*, 11:4241–4257, December 2000.

[23] T. Golub, D.K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J.P. Mesirov, H. Coller, M.L. Loh, J.R. Downing, M.A. Caligiuri, C.D. Bloomfield, and E.S. Lander. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286:531–537, 1999.

[24] I. Guyon, J. Makhoul, R. Schwartz, and V. Vapnik. What size test set gives good error estimates? *IEEE transactions on pattern analysis and machine intelligence*, 20:52–64, 1998.

[25] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46:389–422, 2002.

[26] D. Haussler. Quantifying inductive bias: AI learning algorithms and valient's learning framework. *Artificial Intelligence*, 36:177–221, 1988.

[27] D. Heckerman. A tutorial on learning in Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, 1995.

[28] R. Hoffman and V. Tresp. Discovering structure in continuous variables using Bayesian networks. *Advances in Neural Information Processing Systems*, 1996.

[29] C.C. Holmes. A probabilistic nearest neighbour method for statistical pattern recognition. *Jounal Roy. Statist. Soc. B*, 64(2):295–306, 2002.

[30] http://www.maths.lth.se/help/R/com.braju.sma/.

[31] http://www.stat.berkeley.edu/users/terry/zarray/Software/smacode.html.

[32] D. Husmeier. Sensitivity and specificity of inferring genetic regulatory interactions from microarray experiments with dynamic Bayesian networks. *Bioinformatics*, 19(17):2271–2282, 2003.

[33] S. Imoto, T. Goto, and S. Miyano. Estimation of genetic networks and functional structures between genes by using Bayesian networks and nonparametric regression. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 219–227. World Scientific, 2002.

[34] T. Joachims. Estimating the generalization performance of a SVM efficiently. In *Proceedings of the Seventeenth International Conference on Machine Learning. Stanford, CA.*, 2000.

[35] M. Jordan, editor. *Learning in Graphical Models.* MIT press, 1998.

[36] M.I. Jordan. Graphical models. *Statistical Science*, 19:140–155, 2004.

[37] A. Kerr and G. Churchill. Experimental design for gene expression microarrays. *Biostatistics*, 2(183–201), 2001.

[38] A. Kerr and G. Churchill. Statistical design and the analysis of gene expression microarrays. *Genetic Research*, 77:123–128, 2001.

[39] S. Kim, S. Imoto, and S. Miyano. Dynamic Bayesian network and nonparametric regression model for inferring gene networks. *Genome Informatics*, 13:371–372, 2002.

[40] S. Kirkpatrick. Optimisation by simulated annealing: Quantitive study. *Journal of Statistical Physics*, 34:975–986, 1984.

[41] N. Lawrence, M. Milo, M. Niranjan, P. Rashbass, and S. Soullier. Reducing the variability in cDNA microarray image processing by Bayesian inference. *Bioinformatics*, 20(4):518–526, 2004.

[42] L. Lazzeroni and A. Owen. Plaid models for gene expression data. *Statistica Sinica*, 12(1):61–86, January 2002.

[43] M. Lee, F. Cuo, G. Whitmore, and J. Sklar. Importance of replication in microarray gene expression studies: Statistical methods and evidence from repetitive cdna hybridizations. *Proceedings of the National Academy of Science*, 97(18):9834–9839, 2000.

[44] Y. Li, C. Campbell, and M. Tipping. Bayesian automatic relevance determination algorithms for classifying gene expression data. *Bioinformatics*, 18:1332–1339, 2002.

[45] Y-J. Lu, D. Williamson, J. Clark, R. Wang, N. Tiffin, L. Skelton, T. Gordon, R. Williams, B. Allan, A. Jackman, C. Cooper, K. Pritchard-Jones, and J. Shipley. Comparative expressed sequence hybridization to chromosomes for tumor classification and identification of genomic regions of differential gene expression. *PNAS*, 98(16):9197–9202, 2001.

[46] Y-J. Lu, D. Williamson, R. Wang, B. Summersgill, S. Rodriguez, S. Rogers, K. Pritchard-Jones, C. Campbell, and J. Shipley. Expression profiling targeting chromosomes for tumor classification and prediction of clinical behavior. *Genes, Chromosomes and Cancer*, 38:207–214, 2003.

[47] A.M. Martoglio, J.W. Miskin, S.K. Smith, and D.J.C MacKay. A decomposition model to track gene expression signatures:preview on observer-independent classification of ovarian cancer. *Bioinformatics*, 18(12):1617–1624, 2002.

[48] G.J. McLachlan, R.W. Bean, and D. Peel. A mixture model-based approach to the clustering of microarray expression data. *Bioinformatics*, 18(3):413–422, 2002.

[49] M. Milo, A. Fazeli, M. Niranjan, and N. Lawrence. A probabilistic model for the extraction of expression levels from Oligonucleotide arrays. *Biochemical Transactions*, 31(6), 2003.

[50] T. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[51] S. Mukherkee, P. Tamayo, S. Rogers, R. Rifkin, A. Engle, C. Campbell, T. Golub, and J. Mesirov. Estimating dataset size requirements for classifying DNA microarray data. *Journal of Computational Biology*, 10:119–142, 2003.

[52] R. Neal. *Bayesian Learning for Neural Networks (Lecture Notes in Statistics 118)*. Springer, 1996.

[53] D. Noterman, U. Alon, A. Sierk, and A. Levine. Transcriptional gene expression profiles for colorectal adenoma, adenocarcinoma and normal tissue examined by Oligonucleotide array. *Cancer Research*, 61:3124–3130, 2001.

[54] M. Opper and D. Haussler. Generalisation performance of Bayes optimal classification algorithms for learning a perceptron. *Phys. Rev. Letters*, 66:2677–2680, 1991.

[55] M. Opper, W. Kinzel, J. Kleinz, and R. Nehl. On the ability of the optimal perceptron to generalise. *J. Phys. A*, 23:L581–L586, 1990.

[56] J. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin DAG's for multiclass classification. *Advances in Neural Information Processing Systems*, 12:547–553, 2000.

[57] J.C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In Alexander J. Smola, Peter Bartlett, Schölkopf Bernhard, and Dale Schuurmans, editors, *Advances in Large Margin Classifiers*, 1999.

[58] S. Pomeroy, P. Tamayo, M. Gaasenbeek, L. Sturla, M. Angelo, M. McLaughlin, J. Kim, L. Goumnerova, P. Black, C. Lau, J. Allen, D. Zagzag, J. Olson, T. Curran, C. Wetmore, J. Biegel, T. Poggio, S. Mukherjee, R. Rifkin, A. Califano, G. Stolvitzky, D. Louis, J.P. Mesirov, E. Lander, and T. Golub. Gene expression-based classification and outcome prediction of central nervous system embryonal tumors. *Nature*, 415:436–442, 2002.

[59] S. Ramaswamy, R.T. Osteen, and L.N. Shulman. *Clinical Oncology*, chapter Metastatic cancer from an unknown primary site. American Cancer Society, Atlanta, GA, 2001.

[60] S. Ramaswamy, P. Tamayo, R. Rifkin, S. Mukherjee, M. Yeang, C.H.and Angelo, C. Ladd, M. Reich, E. Latulippe, J.P. Mesirov, T. Poggio, W. Gerald, M. Loda, E.S. Lander, and T.R. Golub. Multiclass cancer diagnosis using tumor gene expression signatures. *Proceedings of the National Academy of Science*, 98(26):15149–15154, 2001.

[61] C. Rangel, J. Angus, Z. Ghahramani, M. Lioumi, E. Sotheran, E. Gaiba, D. Wild, and F. Falciani. Modeling t-cell activation using gene expression profiling and state-space models. *Bioinformatics*, 20(9):1361–1372, 2004.

[62] D.G. Rees. *Essential Statistics*. Chapman and Hall, 2001.

[63] S. Rogers, M. Girolami, and C. Campbell. A latent process decomposition model for interpreting cdna microarray datasets. *Currents in Computational Molecular Biology*, 2004. Eigth International Conference on Research in Computational Molecular Biology (RECOMB 2004).

[64] S. Rogers, M. Girolami, C. Campbell, and R. Breitling. The Latent Process Decomposition of cDNA microarray datasets. *IEEE Transactions on Bioinformatics and Computational Biology*, 2004. In Press.

[65] S. Rogers, R. Williams, and C. Campbell. *Bioinformatics using Computational Intelligence Paradigms*, chapter Class Prediction with Microarray Datasets. Springer-Verlag, 2004.

[66] M. Schummer, W. Ng, R. Bumgarner, P. Nelson, B. Schummer, D. Bednarski, L. Hassell, R. Baldwin, B. Karlan, and L. Hood. Comparative hybridization of an array of 21500 ovarian cDNAs for the discovery of genes overexpressed in ovarian carcinomas. *International Journal on Genes and Genomes*, 238:375–385, 1999.

[67] E. Segal, A. Battle, and D. Koller. Decomposing gene expression into cellular processes. In *Pacific Symposium on Biocomputing 2003*, 2003.

[68] M.A. Shipp, K.N. Ross, P. Tamayo, A.P. Weng, J.L. Kutok, R.C.T. Aguiar, M. Gaasenbeel, M. Angelo, M. Reich, G.S. Pinkus, T.S. Ray, M.A. Koval, K.W. Last, A. Norton, T.A. Lister, J. Mesirov, D.S. Neuberg, E.S. Lander, J.C. Aster, and T.R. Golub. Diffuse large b-cell lymphoma outcome prediction by gene-expression profiling and supervised machine learning. *Nature Medicine*, 8:68–74, 2002.

[69] D. Singh, P. Febbo, K. Ross, D. Jackson, J. Manola, C. Ladd, P. Tamayo, A. Renshaw, A. D'Amico, J. Richie, E. Lander, M. Loda, P. Kantoff, T. Golub, and W. Sellers. Gene expression correlates of clinical prostate cancer behavior. *Cancer Cell*, 1:203–209, 2002.

[70] T. Sørlie, C. Perou, R. Tibshirani, T. Aas, S. Geisler, H. Johnsen, T. Hastie, M. Eisen, M. van de Rijn, S. Jeffrey, T. Thorsen, H. Quist, J. Matese, P. Brown, D. Botstein, Eystein Lønning P., and A. Børresen-Dale. Gene expression patterns of breast carcinomas distinguish tumor subclasses with clinical implications. *Proc. Natl. Acad. Sci.*, 98:10869–10874, 2001.

[71] P. Spellman, G. Sherlock, M. Zhang, V. Iyer, K. Anders, M. Eisen, P. Brown, D. Botstein, and B. Futcher. Comprehensive identification of cell cycle-regulated genes of the yeast saccharomyces cerevisiae by microarray hybridization. *Molecular Biology of the Cell*, 9:3273–3297, 1998.

[72] M. Tipping and A. Faul. Fast marginal likelihood maximisation for sparse Bayesian models. In *Proceeding of Artifical Intelligence and Statistics*, 2003.

[73] G.C. Tsang, M-K. Oh, L. Rohlin, J.C. Liao, and W.H. Wong. Issues in cdna microarray analysis: Quality filtering, channel normalisation, models of variation and asseessment of gene effects. *Nucleic Acids research*, 29:2549–2557, 2001.

[74] E. van Someren, L. Wessels, M. Reinders, and E. Backer. Searching for limited connectivity in genetic network models. In *Proceedings of the Second International Conference on Systems Biology*, pages 222–230, Pasadena, California, November 2001.

[75] L. van 't Veer, H. Dai, M. van de Vijver, Y. He, A. Hart, M. Mao, H. Peterse, K. van der Kooy, M. Marton, A. Witteveen, G. Schreiber, R. Kerkhoven, C. Roberts, P. Linsley, R. Bernards, and S. Friend. Gene expression profiling predicts clinical outcome of breast cancer. *Nature*, 415:530–536, January 2002.

[76] Vapnik V.N. *Statistical Learning Theory.* John Wiley and Sons, inc., 1998.

[77] T. Watkin, A. Rau, and M. Biehl. The statistical mechanics of learning a rule. *Rev. Modern Physics*, 65:499–556, 1993.

[78] J. Watson and F. Crick. Molecular structure of nucleic acids - a structure for deoxyribose nucleic acid. *Natures*, 171(4356):737–738, 1953.

[79] D.C. Weaver, C.T. Workman, and G.D. Sotrmo. Modeling regulatory networks with weight matrices. In R.B. Altman, K. Lauderdale, A.K. Dunker, L. Hunter, and T.E. Klein, editors, *Proceesings of the Pacific Symposium on Biocomputing*, volume 4, pages 112–123. World Scientific Publishing, 1999.

[80] R. Williams, S. Hing, B. Greer, C. Whiteford, J. Wei, R. Natrajan, A. Kelsey, S. Rogers, C. Campbell, K. Pritchard-Jones, and J. Khan. Prognostic classification of relapsing favorable histology wilms tumor using cDNA microarray expression profiling and support vector machines. *Genes, Chromosomes and Cancer*, 41:65–79, 2004.

[81] C.H. Yeang, S. Ramaswamy, P. Tamayo, S. Mukherjee, R.M. Rifkin, M. Angelo, M. Reich, E. Lander, J.P. Mesirov, and T. Golub. Molecular classification of multiple tumour types. *Bioinformatics*, 17 (suppl 1):S316–S322, 2001.

[82] M.K.S. Yeung, J. Tegner, and J. Collins. Reverse engineering gene networks using singular value decomposition and robust regression. *PNAS*, 99(9):6163–6168, April 2002.

[83] D.E. Zak, F.J. Doyle, G.E. Gonye, and J.S. Schwaber. Simulation studies for the identification of genetic networks from cdna array and regulatory activity data. *Proceedings of the second international conference on systems biology*, pages 231–238, 2001.

[84] D.E. Zak, F.J. Doyle, and J.S. Schwaber. Local identifiability: when can genetic networks be inferred from microarray data? *Proceedings of the third international conference on systems biolog*, (236–237), 2002.

Appendix A

Bias Properties of leave-p-out estimators

A.1 The mean of the leave-p-out estimator is unbiased

Statement 1. The expected error of the leave p out procedure is an unbiased estimate of the expected error of the classifier trained on $N-p$ examples. i.e.

$$\mathbf{E}[L_p(z_1,\ldots,z_N)] = \mathbf{E}[Q(z,f(z_{N-p}))] \tag{A.1}$$

where

$$L_p(z_1,\ldots,z_N) = \frac{1}{p}\sum_{i=1}^{p} Q(z_i,f(z_{N-p})) \tag{A.2}$$

and $Q(z_i,f(z_{N-p}))$ is the error on z_i for a classifier trained on the data with p samples removed and $z=(x,y)$, i.e. it denotes a pattern-label pair.

Proof. The proof is given for the case of $p = 1$, extending it to $p > 1$ is straightforward

$$\begin{aligned}
\mathbf{E}[\tfrac{1}{N}L_1(z_1,\ldots,z_N)] &= \tfrac{1}{N}\int\sum_{i=1}^{N} Q(z_i, f(z_{N-1}))dP(z_1)\ldots dP(z_N) \\
&= \tfrac{1}{N}\int\sum_{i=1}^{N}(Q(z_i, f(z_{N-1}))dP(z_i))dP(z_1)\ldots dP(z_{i-1})dP(z_{i+1})\ldots dP(z_N) \\
&= \tfrac{1}{N}\sum_{i=1}^{N}\mathbf{E}[Q(z_i, f(z_{N-1}))] \\
&= \mathbf{E}[Q(z, f(z_{N-1}))]
\end{aligned} \tag{A.3}$$

so, the estimate is un-biased and the more sub-samples we perform, the more accurate the error estimate becomes.

A.2 The variance *is* biased

Statement 2. The variance of the leave-p-out estimator is less than or equal to the variance of a classifier trained on $N - p$ samples and tested on an independent sample - i.e. it is biased and generally optimistic

$$V[L_p(z_1,\ldots,z_N)] \leq V[Q(z, f(x_{N-p}))] \tag{A.4}$$

where $V[x]$ denotes the variance of x.

Proof. Again, we describe the leave-one-out proof, the extension to leave-p-out is straightforward. The variance for training sets of size $N - 1$ is

$$V[Q(z, f(x_{N-p}))] = \mathbf{E}[Q(z, f(x_{N-p}))^2] - (\mathbf{E}[Q(z, f(x_{N-p}))])^2. \tag{A.5}$$

The variance of the leave-one-out estimator is given as

$$V\left[\frac{1}{N}\sum_{i=1}^{N} t_i\right] \tag{A.6}$$

where

$$t_i = Q(z_i, f(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_N)). \tag{A.7}$$

We can re-write A.6 as

$$V\left[\frac{1}{N}\sum_{i=1}^{N} t_i\right] = \mathbf{E}\left[\left(\frac{1}{N}\sum_{i=1}^{N} t_i\right)^2\right] - \left(\mathbf{E}\left[\frac{1}{N}\sum_{i=1}^{N} t_i\right]\right)^2 \tag{A.8}$$

but from section A.1 we know that we can re-write the second term, giving

$$V\left[\frac{1}{N}\sum_{i=1}^{N} t_i\right] = \mathbf{E}\left[\left(\frac{1}{N}\sum_{i=1}^{N} t_i\right)^2\right] - (\mathbf{E}[Q(z, f(x_{N-p}))])^2. \tag{A.9}$$

Now, we can write

$$\mathbf{E}\left[\left(\frac{1}{N}\sum_{i=1}^{N} t_i\right)^2\right] = \mathbf{E}\left[\frac{1}{N^2}\sum_{i=1}^{N} t_i^2 + \frac{1}{N^2}\sum_{i\neq j}^{N} t_i t_j\right], \tag{A.10}$$

which, if the t_i and t_j are assumed to be iid, gives

$$\mathbf{E}\left[\left(\frac{1}{N}\sum_{i=1}^{N} t_i\right)^2\right] = \mathbf{E}\left[\frac{t^2}{N} + \frac{N^2 - N}{N^2}t^2\right] = \mathbf{E}[Q(z, f(x_{N-p}))^2]. \tag{A.11}$$

However, our t_i and t_j variables are not iid so

$$\mathbf{E}\left[\sum_{i\neq j}^{N} t_i t_j\right] \leq \mathbf{E}\left[\frac{N^2 - N}{N^2}t^2\right]. \tag{A.12}$$

Therefore

$$\mathbf{E}\left[\left(\frac{1}{N}\sum_{i=1}^{N} t_i\right)^2\right] - \left(\mathbf{E}\left[\frac{1}{N}\sum_{i=1}^{N} t_i\right]\right)^2 \leq \mathbf{E}[Q(z, f(x_{N-p}))^2] - (\mathbf{E}[Q(z, f(x_{N-p}))])^2 \tag{A.13}$$

as required.

Appendix B

Naive-Bayes Gaussian Mixture

The likelihood of our training dataset $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ is given by

$$\mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\pi}) = \sum_{i=1}^{n} \log \sum_{k=1}^{\mathcal{K}} \pi_k \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_l, \boldsymbol{\sigma}_k). \tag{B.1}$$

Introducing n training sample specific variational distributions Q_i such that

$$\sum_{k=1}^{\mathcal{K}} Q_{ik} = 1 \tag{B.2}$$

we can lower bound this likelihood as follows

$$\begin{aligned} \mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\pi}) &= \sum_{i=1}^{n} \log \sum_{k=1}^{\mathcal{K}} \tfrac{Q_{ik}\pi_k}{Q_{ik}} \prod_{j=1}^{d} p(x_{ij} | \mu_{kj}, \sigma_{kj}) \\ &\geq \sum_{i=1}^{n} \sum_{k=1}^{\mathcal{K}} Q_{ik} \log \left\{ \tfrac{\pi_k}{Q_{ik}} \prod_{j=1}^{d} p(x_{ij} | \mu_{kj}, \sigma_{kj}) \right\} \\ &= \mathcal{L}'(\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\pi}, \mathbf{Q}). \end{aligned} \tag{B.3}$$

Expanding the various log terms allows us to expand the bound thus

$$\begin{aligned} \mathcal{L}'(\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\pi}, \mathbf{Q}) &= \sum_{i=1}^{n} \sum_{k=1}^{\mathcal{K}} Q_{ik} \log \pi_k \\ &- \sum_{i=1}^{n} \sum_{k=1}^{\mathcal{K}} Q_{ik} \log Q_{ik} \\ &+ \sum_{i=1}^{n} \sum_{k=1}^{\mathcal{K}} \sum_{j=1}^{d} Q_{ik} \log \mathcal{N}(x_{ij} | \mu_{kj}, \sigma_{kj}) \end{aligned} \tag{B.4}$$

where we have used the naive-Bayes assumption to factorise the multivariate Gaussians. The variational-EM algorithm introduced in section 3.3 describes how we can increase the likelihood by iteratively maximising with respect to the variational and model parameters whilst keeping the others fixed. To obtain an update equation for the variational parameters, we need to take partial derivatives of $\mathcal{L}'$ with respect to Q_{ik}, set to zero and solve for Q_{ik}. To ensure the summation constraint over Q_{ik} is enforced, we must also introduce a suitable Lagrangian term and multiplier (λ). i.e., we need to take partial derivatives and solve the following expression

$$\sum_{i=1}^{n}\sum_{k=1}^{\mathcal{K}}\left[Q_{ik}\log\pi_k - Q_{ik}\log Q_{ik} + \sum_{j=1}^{d} Q_{ik}\log\mathcal{N}(x_{ij}|\mu_{kj},\sigma_{kj})\right] - \lambda\left(\sum_{k=1}^{\mathcal{K}} Q_{ik} - 1\right) \tag{B.5}$$

as follows

$$\frac{\delta\mathcal{L}'}{\delta Q_{ik}} = 0 = \log\pi_k - (\log Q_{ik} + 1) + \sum_{j=1}^{d}\log\mathcal{N}(x_{ij}|\mu_{kj},\sigma_{kj}) - \lambda. \tag{B.6}$$

Solving for λ and then Q_{ik} results in the following update

$$Q_{ik} = \frac{\pi_k \prod_{j=1}^{d}\mathcal{N}(x_{ij}|\mu_{kj},\sigma_{kj})}{\sum_{k'=1}^{\mathcal{K}}\pi_{k'}\left[\prod_{j=1}^{d}\mathcal{N}(x_{ij}|\mu_{k'j},\sigma_{k'j})\right]}. \tag{B.7}$$

We proceed in a similar fashion to derive updates for the three sets of model parameters. Firstly, π_k which only appears in the first term in the bound. Adding a Lagrangian term to ensure $\sum_k \pi_k = 1$, we are left with the following expression to differentiate and solve

$$\sum_{i=1}^{n}\sum_{k=1}^{\mathcal{K}} Q_{ik}\log\pi_k - \lambda\left(\sum_{k=1}^{\mathcal{K}}\pi_k\right) \tag{B.8}$$

so,

$$\frac{\delta\mathcal{L}'}{\delta\pi_k} = 0 = \frac{1}{\pi_k}\sum_{i=1}^{n} Q_{ik} - \lambda \tag{B.9}$$

giving the following update

$$\pi_k = \frac{1}{n}\sum_{i=1}^{n} Q_{ik}. \tag{B.10}$$

Finally, the Gaussian means and variances only appear in the final term in the bound. Inserting the form of the Gaussian pdf into this expression we have

$$\sum_{i=1}^{n}\sum_{k=1}^{\mathcal{K}}\sum_{j=1}^{d} Q_{ik}\left[\log\left\{\frac{1}{\sqrt{2\pi}\sigma_{kj}}\right\} - \frac{(x_{ij}-\mu_{kj})^2}{2\sigma_{kj}^2}\right]. \tag{B.11}$$

Partial derivatives with respect to μ_{kj} and σ_{jk} lead to the following two updates

$$\mu_{kj} = \frac{\sum_{i=1}^{n} Q_{ik}x_{ij}}{\sum_{i=1}^{n} Q_{ik}} \tag{B.12}$$

$$\sigma_{kj}^2 = \frac{\sum_{i=1}^{n} Q_{ik}(x_{ij}-\mu_{kj})^2}{\sum_{i=1}^{n} Q_{ik}}. \tag{B.13}$$

So, starting with an initial *guess* of the various parameter values, we can increase the likelihood by iterating the following two steps. Firstly, update variational parameters with B.7 (*E-step*). Secondly update model parameters with B.10, B.12, B.13 (*M-step*).

Appendix C

LPD Update Derivations

Our lower-bound on the likelihood is given by

$$\begin{aligned}
\sum_{g=1}^{\mathcal{G}} \log p(g|\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\alpha}) \geq\ & \sum_{g=1}^{\mathcal{G}} \int_{\Theta} p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) \log p(\boldsymbol{\theta}|\boldsymbol{\alpha}) d\boldsymbol{\theta} \\
-\ & \sum_{g=1}^{\mathcal{G}} \int_{\Theta} p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) \log p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) d\boldsymbol{\theta} \\
+\ & \sum_{g=1}^{\mathcal{G}} \sum_{a=1}^{\mathcal{A}} \sum_{k=1}^{\mathcal{K}} Q_{agk} \log p(e_{ga}|\mu_{ak},\sigma_{ak}) \\
+\ & \sum_{g=1}^{\mathcal{G}} \sum_{a=1}^{\mathcal{A}} \sum_{k=1}^{\mathcal{K}} \int_{\Theta} Q_{agk} p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) \log \theta_k d\boldsymbol{\theta} \\
-\ & \sum_{g=1}^{\mathcal{G}} \sum_{a=1}^{\mathcal{A}} \sum_{k=1}^{\mathcal{K}} Q_{agk} \log Q_{agk}.
\end{aligned}$$

As described in section 3.3, we can take partial derivatives of this with respect to the variational and model parameters to obtain iterative updates that are guaranteed to increase the likelihood. We will now deal with the variational and model parameters in turn.

C.1 Variational Parameters

C.1.1 Q_{agk}

Q_{agk} appears in terms 3, 4 and 5 of the bound. Isolating these and removing summations over a, g and k, we are left with the following

$$Q_{agk} \log p(e_{ga}|\mu_{ak}, \sigma_{ak}) + Q_{agk} \mathbf{E}_{p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)}[\log \theta_k] - Q_{agk} \log Q_{agk} - \lambda \left(\sum_{k=1}^{\mathcal{K}} Q_{agk} - 1 \right) \tag{C.1}$$

where we have added the Lagrangian multiplier λ to ensure the summation constraint on Q_{agk} is satisfied. Taking partial derivatives gives

$$\frac{\delta \mathcal{L}''}{\delta Q_{agk}} = \log p(e_{ga}|\mu_{ak}, \sigma_{ak}) + \mathbf{E}_{p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)}[\log \theta_k] - (\log Q_{agk} + 1) - \lambda = 0. \tag{C.2}$$

Solving for λ and substituting leads to the following expression for Q_{agk}

$$Q_{agk} = \frac{p(e_{ga}|\mu_{ak}, \sigma_{ak}) \exp \left\{ \mathbf{E}_{p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)}[\log \theta_k] \right\}}{\sum_{k'=1}^{\mathcal{K}} p(e_{ga}|\mu_{ak'}, \sigma_{ak'}) \exp \left\{ \mathbf{E}_{p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)}[\log \theta_{k'}] \right\}}. \tag{C.3}$$

where

$$\mathbf{E}_{p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)}[\log \theta_k] = \Psi(\gamma_{gk}) - \Psi(\sum_{j=1}^{\mathcal{K}} \gamma_{gj}). \tag{C.4}$$

and $\Psi(z)$ is the digamma function - the first derivative of the log gamma function.

C.1.2 γ_{gk}

γ_{gk} appears in terms 1, 2 and 4 of the bound. Isolating these terms and removing summations over g and k leaves

$$\int_{\Theta} p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) \log p(\boldsymbol{\theta}|\boldsymbol{\alpha}) d\boldsymbol{\theta} - \int_{\Theta} p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) \log p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) d\boldsymbol{\theta} + \sum_{a=1}^{\mathcal{A}} Q_{agk} p \int_{\Theta} (\boldsymbol{\theta}|\boldsymbol{\gamma}_g) \log \theta_k d\boldsymbol{\theta}. \tag{C.5}$$

Consider the first term in C.5. Substituting in the parametric form for the Dirichlet $p(\boldsymbol{\theta}|\boldsymbol{\alpha})$ leaves us with the following

$$\begin{aligned}
\int_{\Theta} p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) \log p(\boldsymbol{\theta}|\boldsymbol{\alpha}) d\boldsymbol{\theta} &= \int_{\Theta} p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) \log \left\{ \frac{\Gamma(\sum_{k=1}^{\mathcal{K}} \alpha_k)}{\prod_{k=1}^{\mathcal{K}} \Gamma(\alpha_k)} \prod_{k=1}^{\mathcal{K}} \theta_k^{\alpha_k - 1} \right\} d\boldsymbol{\theta} \\
&= \log C_{\boldsymbol{\alpha}} + \int_{\Theta} p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) \left(\sum_{k=1}^{\mathcal{K}} (\alpha_k - 1) \log \theta_k \right) d\boldsymbol{\theta} \\
&= \log C_{\boldsymbol{\alpha}} + \sum_{k=1}^{\mathcal{K}} (\alpha_k - 1) \int_{\Theta} p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) \log \theta_k d\boldsymbol{\theta} \\
&= \log C(\boldsymbol{\alpha}) + \sum_{k=1}^{\mathcal{K}} (\alpha_k - 1) [\Psi(\gamma_{gk}) - \Psi(\sum_{j=1}^{\mathcal{K}} \gamma_{gj})]
\end{aligned}$$

where

$$C(\boldsymbol{\alpha}) = \frac{\Gamma(\sum_{k=1}^{\mathcal{K}} \alpha_k)}{\prod_{k=1}^{\mathcal{K}} \Gamma(\alpha_k)}. \tag{C.6}$$

Similarly,

$$\int_{\Theta} p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) \log p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) d\boldsymbol{\theta} = \log C(\boldsymbol{\gamma}_g) + \sum_{k=1}^{\mathcal{K}} (\gamma_{gk} - 1) [\Psi(\gamma_{gk}) - \Psi(\sum_{j=1}^{\mathcal{K}} \gamma_{gj})]. \tag{C.7}$$

Therefore, C.5 becomes (removing summations over g and k wherever possible)

$$\log C(\boldsymbol{\alpha}) - \log C(\boldsymbol{\gamma}_g) + \left(\alpha_k - \gamma_{gk} + \sum_{a=1}^{\mathcal{A}} Q_{agk} \right) [\Psi(\gamma_{gk}) - \Psi(\sum_{j=1}^{\mathcal{K}} \gamma_{gj})]. \tag{C.8}$$

Taking partial derivatives with respect to γ_{gk} gives

$$\frac{\delta \mathcal{L}''}{\delta \gamma_{gk}} = \left(\alpha_k - \gamma_{gk} + \sum_{a=1}^{\mathcal{A}} Q_{agk} \right) [\Psi'(\gamma_{gk}) - \Psi'(\sum_{j=1}^{\mathcal{K}} \gamma_{gj})] = 0 \tag{C.9}$$

where $\Psi'(z)$ is the tri-gamma function, the second derivative of the log gamma function. For positive real z, $\Psi'(z)$ is monotonically decreasing. Therefore, given that $\gamma_{gk} > 0$,

$$\Psi'(\sum_{j=1}^{\mathcal{K}} \gamma_{gj}) < \Psi'(\gamma_{gk}) \tag{C.10}$$

and hence

$$\Psi(\gamma_{gk}) - \Psi(\sum_{j=1}^{\mathcal{K}} \gamma_{gj}) \neq 0. \tag{C.11}$$

Therefore, we have

$$\left(\alpha_k - \gamma_{gk} + \sum_{a=1}^{\mathcal{A}} Q_{agk}\right) = 0 \tag{C.12}$$

and so

$$\gamma_{gk} = \alpha_k + \sum_{a=1}^{\mathcal{A}} Q_{agk}. \tag{C.13}$$

C.2 Model Parameters

C.2.1 μ_{ak}

μ_{ak} appears in term 3 of our bound. Extracting this term, and removing summations over a and k leaves us with

$$\sum_{g=1}^{\mathcal{G}} Q_{agk} \log p(e_{ga}|\mu_{ak}, \sigma_{ak}) = \sum_{g=1}^{\mathcal{G}} Q_{agk} \log \left\{ \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{ -\frac{(e_{ga} - \mu_{ak})^2}{2\sigma_{ak}^2} \right\} \right\}. \tag{C.14}$$

Taking partial derivatives with respect to μ_{ak} leaves

$$\frac{\delta \mathcal{L}''}{\delta \mu_{ak}} = \sum_{g=1}^{\mathcal{G}} Q_{agk} \frac{e_{ga} - \mu_{ka}}{\sigma^2} = 0. \tag{C.15}$$

Therefore, we have the following update for μ_{ka}

$$\mu_{ka} = \frac{\sum_{g=1}^{\mathcal{G}} Q_{agk} e_{ga}}{\sum_{g=1}^{\mathcal{G}} Q_{agk}} \tag{C.16}$$

C.2.2 σ^2_{ak}

As with μ_{ak}, σ^2_{ak} only appears in term 3 of the bound. Taking partial derivatives of C.14 with respect to σ_{ak} gives

$$\frac{\delta \mathcal{L}''}{\delta \mu_{ak}} = -\frac{1}{\sigma} \sum_{g=1}^{\mathcal{G}} Q_{agk} + \frac{1}{\sigma^3} \sum_{g=1}^{\mathcal{G}} Q_{akg} (e_{ga} - \mu_{ka})^2 = 0. \tag{C.17}$$

This leads to the following update for σ^2_{ak}

$$\sigma^2_{ak} = \frac{\sum_{g=1}^{\mathcal{G}} Q_{agk} (e_{ga} - \mu_{ak})^2}{\sum_{g=1}^{\mathcal{G}} Q_{agk}}. \tag{C.18}$$

C.2.3 α_k

α_k appears only in the first term of the bound. Expanding this term gives and removing summations over k where possible gives us

$$\mathcal{G}\left(\log \Gamma(\sum_{j=1}^{\mathcal{K}} \alpha_j) - \log \Gamma(\alpha_k) \right) + \sum_{g=1}^{\mathcal{G}} (\alpha_k - 1)[\Psi(\gamma_{gk}) - \Psi(\sum_{j=1}^{\mathcal{K}} \gamma_{gj})]. \tag{C.19}$$

Unfortunately, there is no closed-form solution for α_k and so we have to resort to second order techniques. Following, [8] we use a Newton-Raphson technique given by

$$\boldsymbol{\alpha}^{new} = \boldsymbol{\alpha}^{old} - \mathbf{H}^{-1}\mathbf{g} \tag{C.20}$$

where $\mathbf{g}$ has elements $\delta/\delta\alpha_i$ and $\mathbf{H}$ has elements $\delta/\delta\alpha_i\alpha_j$. Taking first derivatives of our expression with respect to α_k gives us the following

$$\frac{\delta\mathcal{L}''}{\delta\alpha_k} = g_k = \mathcal{G}\Psi(\sum_{j=1}^{\mathcal{K}}\alpha_j) - \mathcal{G}\Psi(\alpha_k) + \sum_{g=1}^{\mathcal{G}}\left(\Psi(\gamma_{gk}) - \Psi(\sum_{j=1}^{\mathcal{K}}\gamma_{gj})\right). \tag{C.21}$$

Differentiating again, with respect to α_i gives us the following

$$\frac{\delta^2\mathcal{L}''}{\delta\alpha_k\alpha_i} = H_{ki} = \mathcal{G}\Psi'(\sum_{j=1}^{\mathcal{K}}\alpha_j) - \delta_{ki}\mathcal{G}\Psi'(\alpha_k) \tag{C.22}$$

where $\delta_{ki} = 1$ iff $i = k$ and 0 otherwise.

We can perform this iteration until some suitable convergence criteria is met. For large values of $\mathcal{K}$ we may wish to avoid the repeated inversion of the Hessian, $\mathbf{H}$. This can be achieved by noting that $\mathbf{H}$ is of the following form

$$\mathbf{H} = \mathbf{h}^T\mathbf{I} + \mathbf{1}a\mathbf{1}^T \tag{C.23}$$

i.e. a constant value, a plus a vector $\mathbf{h}$ on the diagonal ($\mathbf{I}$ is the identity and $\mathbf{1}$ is a matrix full of ones). The matrix inversion lemma states

$$(\mathbf{A} + \mathbf{X}\mathbf{B}\mathbf{X}^T)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{X}(\mathbf{B}^{-1} + \mathbf{X}^T\mathbf{A}^{-1}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{A}^{-1}. \tag{C.24}$$

Therefore, the inverse of the Hessian is given by

$$\mathbf{H}^{-1} = (\mathbf{h}^T\mathbf{I})^{-1} - \frac{(\mathbf{h}^T\mathbf{I})^{-1}\mathbf{1}\mathbf{1}^T(\mathbf{h}^T\mathbf{I})^{-1}}{a^{-1} + \sum_{j=1}^{\mathcal{K}} h_j^{-1}}. \tag{C.25}$$

If we multiply this by $\mathbf{g}$ we get the following update vector for $\boldsymbol{\alpha}$

$$\mathbf{u}_i = (\mathbf{H}^{-1}\mathbf{g})_i = \frac{g_i - c}{h_i} \tag{C.26}$$

where g_i is given above, $h_i = \Psi'(\alpha_i)$ and

$$c = \frac{\sum_{j=1}^{\mathcal{K}} g_j/h_j}{a^{-1} + \sum_{j=1}^{\mathcal{K}} h_j^{-1}}. \tag{C.27}$$

Appendix D

LPD with mixture of Dirichlet Priors

In this appendix, we introduce a slight variation on the original LPD model. Recall that in our original model, for each data point (gene or cell sample, depending on the type of analysis being performed) we sampled a multinomial from a Dirichlet distribution with parameters $\boldsymbol{\alpha}$. It may be that for heterogeneous data, a single Dirichlet is not flexible enough to capture complicated structure present. For this reason, we have extended the basic model from a single Dirichlet prior to a mixture prior. The graphical representation of this can be seen in figure D.1. We must now sample a prior component m from a multinomial $\mathbf{b}$ which selects the Dirichlet from which we sample our $\boldsymbol{\theta}$. The rest of the model remains unchanged.

If we have $\mathcal{M}$ prior components, the likelihood of a dataset of $\mathcal{G}$ genes and $\mathcal{A}$ arrays is

$$\sum_{g=1}^{\mathcal{G}} \log p(g|\mathbf{b}, \boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\alpha}) = \sum_{g=1}^{\mathcal{G}} \log \sum_{m=1}^{\mathcal{M}} b_m \int_{\Theta} p(g|\boldsymbol{\alpha}, \boldsymbol{\mu}, \boldsymbol{\sigma}) p(\boldsymbol{\theta}|\boldsymbol{\alpha}_m) d\boldsymbol{\theta} \tag{D.1}$$

Introducing a set of sample specific multinomials R_{mg} such that $\sum_{m=1}^{\mathcal{M}} R_{mg} = 1$ and then

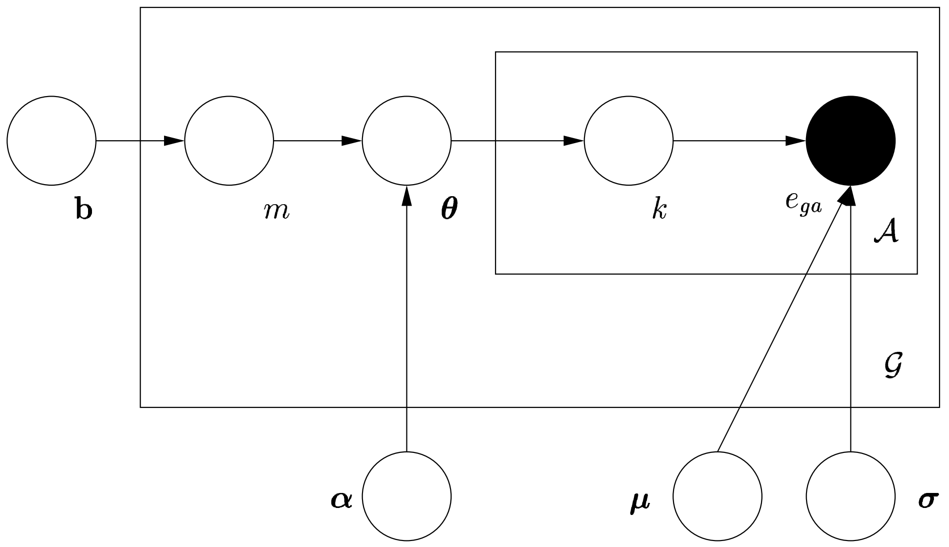


Fig. D.1: Plates diagram for the LPD with mixture of Dirichlet priors model

using Jensen's inequality we can lower bound this quantity

$$\begin{aligned}\sum_{g=1}^{\mathcal{G}} \log p(g|\mathbf{b},\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\alpha}) \;&\geq \sum_{g=1}^{\mathcal{G}}\sum_{m=1}^{\mathcal{M}} R_{mg}\log b_m \\ &- \sum_{g=1}^{\mathcal{G}}\sum_{m=1}^{\mathcal{M}} R_{mg}\log R_{mg} \\ &+ \sum_{m=1}^{\mathcal{M}} R_{mg}\log\left\{\int_\Theta p(g|\boldsymbol{\alpha},\boldsymbol{\mu},\boldsymbol{\sigma})p(\boldsymbol{\theta}|\boldsymbol{\alpha}_m)d\boldsymbol{\theta}\right\}.\end{aligned} \tag{D.2}$$

Introducing variational, gene specific Dirichlet's $\boldsymbol{\gamma}_g$, we can perform a similar operation to bound the final term

$$\begin{aligned}\sum_{m=1}^{\mathcal{M}} R_{mg}\log\left\{\int_\Theta p(g|\boldsymbol{\alpha},\boldsymbol{\mu},\boldsymbol{\sigma})p(\boldsymbol{\theta}|\boldsymbol{\alpha}_m)d\boldsymbol{\theta}\right\} \;&\geq \sum_{g=1}^{\mathcal{G}} \mathbf{E}_{p(\theta|\gamma_g)}[\log p(g|\boldsymbol{\alpha},\boldsymbol{\mu},\boldsymbol{\sigma})] \\ &+ \sum_{g=1}^{\mathcal{G}}\sum_{m=1}^{\mathcal{M}} R_{mg}\mathbf{E}_{p(\theta|\gamma_g)}[\log p(\boldsymbol{\theta}|\boldsymbol{\alpha}_m)] \\ &- \sum_{g=1}^{\mathcal{G}} \mathbf{E}_{p(\theta|\gamma_g)}[\log p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)]\end{aligned} \tag{D.3}$$

where redundant R_{mg} and $\sum_m$ terms have been removed. One final set of variational parameters Q_{kga} are introduced such that $\sum_k Q_{kga} = 1$ to the first term above. These parameters can be thought of as the posterior distribution over the different processes for each g and a. Jensen's inequality can be used one final time to give the following bound on this one term

$$\begin{aligned}\sum_{g=1}^{\mathcal{G}} \mathbf{E}_{p(\theta|\gamma_g)}[\log p(g|\boldsymbol{\alpha},\boldsymbol{\mu},\boldsymbol{\sigma})] \;\geq\; & \sum_{g=1}^{\mathcal{G}}\sum_{a=1}^{\mathcal{A}}\sum_{k=1}^{\mathcal{K}} Q_{kga}\log[p(e_{ga}|k,\mu_{ak},\sigma_{ak})] \\ & -\sum_{g=1}^{\mathcal{G}}\sum_{a=1}^{\mathcal{A}}\sum_{k=1}^{\mathcal{K}} Q_{kga}\log Q_{kga} \\ & +\sum_{g=1}^{\mathcal{G}}\sum_{a=1}^{\mathcal{A}}\sum_{k=1}^{\mathcal{K}} Q_{kga}\mathbf{E}_{p(\theta|\gamma_g)}[\log \boldsymbol{\theta}_k]\end{aligned} \tag{D.4}$$

Combining these three bounds into one gives the following

$$\begin{aligned}\sum_{g=1}^{\mathcal{G}} \log p(g|\mathbf{b},\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\alpha}) \;\geq\; & \sum_{g=1}^{\mathcal{G}}\sum_{m=1}^{\mathcal{M}} R_{mg}\log b_m \\ & -\sum_{g=1}^{\mathcal{G}}\sum_{m=1}^{\mathcal{M}} R_{mg}\log R_{mg} \\ & -\sum_{g=1}^{\mathcal{G}} \mathbf{E}_{p(\theta|\gamma_g)}[\log p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)] \\ & +\sum_{g=1}^{\mathcal{G}}\sum_{m=1}^{\mathcal{M}} R_{mg}\mathbf{E}_{p(\theta|\gamma_g)}[\log p(\boldsymbol{\theta}|\boldsymbol{\alpha}_m)] \\ & +\sum_{g=1}^{\mathcal{G}}\sum_{a=1}^{\mathcal{A}}\sum_{k=1}^{\mathcal{K}} Q_{kga}\log p(e_{ga}|k,\mu_{ak},\sigma_{ak}) \\ & -\sum_{g=1}^{\mathcal{G}}\sum_{a=1}^{\mathcal{A}}\sum_{k=1}^{\mathcal{K}} Q_{kga}\log Q_{kga} \\ & +\sum_{g=1}^{\mathcal{G}}\sum_{a=1}^{\mathcal{A}}\sum_{k=1}^{\mathcal{K}} Q_{kga}\mathbf{E}p(\theta|\gamma_g)[\log \theta_k].\end{aligned} \tag{D.5}$$

D.1 Model Parameters

The process mean and variance parameters μ_{ak} and σ_{ak}^2 only appear in the 5th term above. This term can be expanded as follows

$$\begin{aligned}\sum_{g=1}^{\mathcal{G}}\sum_{a=1}^{\mathcal{A}}\sum_{k=1}^{\mathcal{K}} Q_{kga}\log p(e_{ga}|k,\mu_{ak},\sigma_{ak}) \;=\; & \sum_{g=1}^{\mathcal{G}}\sum_{a=1}^{\mathcal{A}}\sum_{k=1}^{\mathcal{K}} Q_{kga}\log\left[\frac{1}{\sqrt{2\pi\sigma_{ak}^2}}\right] \\ & -\sum_{g=1}^{\mathcal{G}}\sum_{a=1}^{\mathcal{A}}\sum_{k=1}^{\mathcal{K}} Q_{kga}\frac{(e_{ga}-\mu_{ak})^2}{2\sigma_{ak}^2}.\end{aligned} \tag{D.6}$$

Taking partial derivatives of this expression with respect to μ_{ak} and σ_{ak}, setting the results to zero and solving yields the following updates

$$\mu_{ak} = \frac{\sum_{g=1}^{\mathcal{G}} Q_{kga} e_{ga}}{\sum_{g=1}^{\mathcal{G}} Q_{kga}} \tag{D.7}$$

$$\sigma_{ak}^2 = \frac{\sum_{g=1}^{\mathcal{G}} Q_{kga}(e_{ga} - \mu_{ak})^2}{\sum_{g=1}^{\mathcal{G}} Q_{kga}}. \tag{D.8}$$

The mixture component priors b_m only appear in term 1 of the likelihood bound. Adding a suitable Lagrangian to ensure $\sum_m b_m = 1$, taking partial derivatives and solving yields the following iterative update

$$b_m = \frac{1}{\mathcal{G}} \sum_{g=1}^{\mathcal{G}} R_{mg}. \tag{D.9}$$

The $\boldsymbol{\alpha}_m$ parameters only appears in term 4 of the above bound but partial differentiation of this term with respect to α_{mk} reveals that a simple iterative procedure is not possible and a second order technique is required. The vector of first derivatives has the following components

$$g_i = [\Psi(\sum_{k'} \alpha_{k'm}) - \Psi(\alpha_{km})] \sum_{g=1}^{\mathcal{G}} R_{mg} + \sum_{g=1}^{\mathcal{G}} R_{mg} [\Psi(\sum_{k'} \gamma_{gk'}) - \Psi(\gamma_{gk})] \tag{D.10}$$

and the Hessian

$$H_{ij} = [\Psi'(\sum_{k'} \alpha_{k'm}) - \delta(i,j)\Psi'(\alpha_{km})] \sum_{g=1}^{\mathcal{G}} R_{mg} \tag{D.11}$$

A Newton-Raphson technique can now be used, as described in Appendix C.

D.2 Variational Parameters

Firstly, Q_{kga} appears in terms 5, 6 and 7 of the bound. Taking these terms, removing summations over k, a and a and adding a Lagrangian to satisfy the summation constraint we have

$$Q_{kga} \log p(e_{ga}|k, \mu_{ak}, \sigma_{ak}) - Q_{kga} \log Q_{kga} + Q_{kga} \mathbf{E}_{p(\theta|\gamma_g)}[\log \theta_k] - \lambda(\sum_{k=1}^{\mathcal{K}} Q_{kga} - 1) \tag{D.12}$$

taking partial derivatives wrt Q_{kga} and setting to zero gives

$$0 = \log p(e_{ga}|k, \mu_{ak}, \sigma_{ak}) - (\log[Q_{kga}] + 1) + \mathbf{E}_{p(\theta|\gamma_g)}[\log \theta_k] - \lambda \tag{D.13}$$

solving for λ and re-arranging gives the following update for Q_{kga}

$$Q_{kga} = \frac{p(e_{ga}|k, \mu_{ak}, \sigma_{ak}) \exp[\mathbf{E}_{p(\theta|\gamma_g)}[\log \theta_k]]}{\sum_{g'=1}^{\mathcal{G}} p(e_{g'a}|k, \mu_{ak}, \sigma_{ak}) \exp[\mathbf{E}_{p(\theta|\gamma_{g'})}[\log \theta_k]]} \tag{D.14}$$

where

$$\mathbf{E}p(\theta|\gamma_g)[\log \theta_k] = \Psi(\gamma_{gk}) - \Psi(\sum_j \gamma_{gj}). \tag{D.15}$$

Now, γ_{gk}. This variable appears in terms 3, 4 and 7. Extracting these terms and removing summations over k and g leaves

$$-\mathbf{E}_{p(\theta|\gamma_g)}[\log p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)] + \sum_{m=1}^{\mathcal{M}} R_{mg} \mathbf{E}_{p(\theta|\gamma_g)}[\log p(\boldsymbol{\theta}|\boldsymbol{\alpha}_m)] + \sum_{a=1}^{\mathcal{A}} Q_{kga} \mathbf{E}_{p(\theta|\gamma_g)}[\log \theta_k]. \tag{D.16}$$

Now,

$$\begin{aligned} \mathbf{E}p(\theta|\gamma_g)[\log p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g)] &= \int_\Theta \log\left[\frac{\Gamma(\sum_k \gamma_{gk})}{\prod_k \Gamma(\gamma_{gk})} + \sum_{k=1}^{\mathcal{K}} (\gamma_{gk} - 1)\log[\theta_k]\right] p(\boldsymbol{\theta}|\boldsymbol{\gamma}_g) d\boldsymbol{\theta} \\ &= \log\left[\frac{\Gamma(\sum_k \gamma_{gk})}{\prod_k \Gamma(\gamma_{gk})}\right] + \sum_{k=1}^{\mathcal{K}} (\gamma_{gk} - 1)\mathbf{E}_{p(\theta|\gamma_g)}[\log \theta_k] \\ &= \log\left[\frac{\Gamma(\sum_k \gamma_{gk})}{\prod_k \Gamma(\gamma_{gk})}\right] + \sum_{k=1}^{\mathcal{K}} (\gamma_{gk} - 1)[\Psi(\gamma_{gk}) - \Psi(\textstyle\sum_j \gamma_{gj})] \end{aligned} \tag{D.17}$$

and similarly for the $p(\boldsymbol{\theta}|\boldsymbol{\alpha})$ term. Therefore, the relevant terms simplify to the following (again, omitting unnecessary summations and non γ_{gk} terms)

$$\left(\sum_{m=1}^{\mathcal{M}} R_{mg}\alpha_{mk} - \gamma_{gk} + \sum_{g=1}^{\mathcal{G}} Q_{kga}\right)[\Psi(\gamma_{gk}) - \Psi(\sum_j \gamma_{gj})] - \log\left[\frac{\Gamma(\sum_k \gamma_{gk})}{\prod_k \Gamma(\gamma_{gk})}\right] \tag{D.18}$$

Taking partial derivatives with respect to γ_{gk} and setting to zero leaves

$$\begin{aligned} 0 = \quad & \left(\sum_{m=1}^{\mathcal{M}} R_{mg}\alpha_{mk} - \gamma_{gk} + \sum_{g=1}^{\mathcal{G}} Q_{kga}\right)[\Psi'(\gamma_{gk}) - \Psi'(\textstyle\sum_j \gamma_{gj})] \\ & -[\Psi(\gamma_{gk}) - \Psi(\textstyle\sum_j \gamma_{gj})] \\ & +[\Psi(\gamma_{gk}) - \Psi(\textstyle\sum_j \gamma_{gj})] \end{aligned} \tag{D.19}$$

Giving the following update for γ_{gk}

$$\gamma_{gk} = \sum_{m=1}^{\mathcal{M}} R_{mg}\alpha_{mk} + \sum_{g=1}^{\mathcal{G}} Q_{kga}. \tag{D.20}$$

Finally, R_{mg} appears in terms 1, 2 and 4 of the bound. Extracting these, removing summations over m and g and adding a Lagrangian to ensure $\sum_m R_{mg} = 1$ we are left with the following

$$R_{mg}\log b_m - R_{mg}\log R_{mg} + R_{mg}\mathbf{E}_{p(\theta|\gamma_g)}[p(\boldsymbol{\theta}|\boldsymbol{\alpha}_m)] - \lambda(\sum_{m=1}^{\mathcal{M}} R_{mg} - 1). \tag{D.21}$$

Taking partial derivatives with respect to R_{mg} and setting to zero leaves us with

$$0 = \log b_m - (\log[R_{mg}] + 1) + \mathbf{E}_{p(\theta|\gamma_g)}[p(\boldsymbol{\theta}|\boldsymbol{\alpha}_m)] - \lambda. \tag{D.22}$$

Solving for λ and re-arranging gives us the following update for R_{mg}

$$R_{mg} = \frac{b_m \exp\{\mathbf{E}_{p(\theta|\gamma_g)}[\log(p(\boldsymbol{\theta}|\boldsymbol{\alpha}_m)]\}}{\sum_{m'=1}^{\mathcal{M}} b_{m'} \exp\{\mathbf{E}_{p(\theta|\gamma_g)}[\log(p(\boldsymbol{\theta}|\boldsymbol{\alpha}_{m'})]\}} \tag{D.23}$$

where

$$\exp\{\mathbf{E}_{p(\theta|\gamma_g)}[\log(p(\boldsymbol{\theta}|\boldsymbol{\alpha}_m)]\} = \frac{\Gamma\left(\sum_k \alpha_{mk}\right)}{\prod_k(\Gamma(\alpha_{mk}))} \exp\left\{\sum_k [(\alpha_{mk} - 1)(\Psi(\gamma_{gk}) - \Psi(\sum_j \gamma_{gj}))]\right\}. \tag{D.24}$$

As with the standard LPD model, these updates can be used iteratively to find a local maximum of the likelihood.

Appendix E

A prior for the discrete LPD

To avoid overfitting, we can place priors on the multinomial paramaters of our descrete LPD model. As the parameters are multinomial distributions, a suitable form of prior is a Dirichlet distribution. In a similar manner to when we added the priors to the standard LPD model, we find that the derivation remains mainly unchanged with only changes in the parameter and Q updates.

The term of our original bound concerned with the paramater updates is given by

$$\sum_{a=1}^{\mathcal{A}} \sum_{g=1}^{\mathcal{G}} \sum_{k=1}^{\mathcal{K}} Q_{agk} \log p(\mathbf{v}_{ga}|\mathbf{p}_{gk}). \tag{E.1}$$

Incorporating our prior term, this becomes

$$\sum_{a=1}^{\mathcal{A}} \sum_{g=1}^{\mathcal{G}} \sum_{k=1}^{\mathcal{K}} Q_{agk} \log \int_{\mathbf{P}} p(\mathbf{v}_{ga}|\mathbf{p}) p(\mathbf{p}|\eta) d\mathbf{p} \tag{E.2}$$

where $p(\mathbf{p}|\eta)$ is Dirichlet, with paramaters η, sensible values of which would be 1.

Introducing a set of gene and process specific Dirichlets - λ_{kg} we obtain the following lower bound on the above quantity

$$\sum_{a=1}^{\mathcal{A}} \sum_{g=1}^{\mathcal{G}} \sum_{k=1}^{\mathcal{K}} Q_{agk} \int_{\mathbf{P}} p(\mathbf{p}|\boldsymbol{\lambda}) \log \left[p(\mathbf{v}_{ga}|\mathbf{p}) \frac{p(\mathbf{p}|\eta)}{p(\mathbf{p}|\boldsymbol{\lambda})} \right] d\mathbf{p}. \tag{E.3}$$

We can proceed as in all the other derivations by taking partial derivatives with respect to the various parameters to obtain fixed-point iterative updates. Omitting the details, the **p** update is replaced by the following update for the new variational Dirichlets

$$\lambda_{gki} = \eta_i + \frac{\sum_{a=1}^{\mathcal{A}} Q_{agk} v_{gai}}{\sum_{a=1}^{\mathcal{A}} Q_{agk}} \tag{E.4}$$

and the update for the variational multinomial paramaters Q_{agk} becomes

$$Q_{agk} = \frac{\exp\{\sum_i v_{gai}[\psi(\lambda_{gki}) - \psi(\sum_j \lambda_{gkj})]\} \exp\{\psi(\gamma_{ak}) - \psi(\sum_j \gamma_{aj})\}}{\sum_{k'} \exp\{\sum_i v_{gai}[\psi(\lambda_{gk'i}) - \psi(\sum_j \lambda_{gk'j})]\} \exp\{\psi(\gamma_{ak'}) - \psi(\sum_j \gamma_{aj})\}}. \tag{E.5}$$

Appendix F

LPD Numerical trick

When evaluating likelihood values for LPD, we are faced with the sum of a product of the following form

$$\log \sum_i \prod_j Q_{ij} \tag{F.1}$$

where Q_{ij} may involve another summation. The problem is in evaluating the product. This is often over genes and so can involve a product over thousands of small numbers that rapidly approaches the machine precision. However, as we are ultimately interested in the log of this quantity, we can perform the following trick. Firstly, we can insert an exponential and a log between the summation and the product, converting the product into a summation of logs.

$$\log \sum_i \exp \left\{ \sum_j \log Q_{ij} \right\}. \tag{F.2}$$

This summation of logs is now tractable so we are left with

$$\log \sum_i \exp \zeta_i. \tag{F.3}$$

Obviously, we cannot directly evaluate $\exp \zeta_i$ but we can make use of the identity $\exp(A+B) = \exp(A)\exp(B)$ to give

$$\log \sum_i \frac{\exp\{\zeta_i + K\}}{\exp K} \tag{F.4}$$

which leads to

$$\log\left\{\sum_i \exp(\zeta_i + K)\right\} - K \tag{F.5}$$

which we can evaluate, given a suitable choice of constant K. Given that the ζ_i values will all be negative, a suitable choice is $-\max[\zeta_i]$ so that in effect we are moving all ζ_i values up so that the maximum value is 0.