Bayesian Two-way Clustering for Gene Expression Data

Graeme K. Ambler

October 23, 2003
Overview

- Gene expression
- A model for clustering
- Implementation
- Examples
- Further work
Introduction to gene expression

- DNA is the molecular storage device of our cells, but it does not actually do much by itself.
- It must first be transcribed into RNA.
- The RNA is then translated into proteins, which do most of the real work.
- This happens at different rates for different genes in different cells.
Gene expression microarrays

• It is possible to extract all of the RNA from cells.

• Gene expression microarrays enable us to measure how much RNA from different genes is in a sample by a process of labelling and hybridisation.

• The expression level of several thousand genes will typically be measured by a single chip.

• Unfortunately, these microarrays are rather expensive, and so is the procedure of preparing the RNA, so we generally only have a small number of samples.
MacKay and Miskin’s model

- MacKay and Miskin (2001) proposed the following model for gene expression data:

\[ y_{gs} = \sum_{h=1}^{H} a_{sh} b_{gh} + \varepsilon_{gs}. \]

where \( g \) = gene, \( s \) = sample, and \( \varepsilon \) is a noise term.

- We think that this model is rather too general to use as-is.

- We propose some simplifications of the model which (we hope) will lead to more interpretable results.
A single sample model

- We first consider a simple model for the single-sample case:

\[ y_g = \sum_{h=1}^{H} z_{gh} b_h + \varepsilon_g, \]

where \( z_{gh} \in \{0, 1\} \) is an allocation variable and \( b_h \in \mathbb{R} \) is a constant level for layer \( h \).

- This is very similar to the standard mixture model with constant variance:

\[ y_g = \sum_{h=1}^{H} z_{gh} \mu_h + \varepsilon_g \]

except that the allocations for a mixture model are constrained to give \( \sum_h z_{gh} \equiv 1 \ \forall g \).
Prior specification

We propose a fully Bayesian MCMC implementation using the following priors:

\[ z_{gh} \sim \text{Bernoulli}\left(\frac{q}{H+1}\right) \]
\[ b_h \sim N(0, \tau^2) \]
\[ H \sim \text{Poisson}(\lambda) \]
\[ \varepsilon \sim N(0, \sigma^2). \]

We also use a conjugate hyperprior on \( \sigma \):

\[ \sigma^{-2} \sim \Gamma(\alpha, \beta). \]

All of the other hyperparameters are held constant.
Implementation continued

- I have implemented an MCMC sampler of this model in C++.

- The sampler performs updates of $z$, $b$ and $\sigma$ using Gibbs sampler steps.

- The number (and composition) of layers is updated using both birth-death and split-merge moves.

- There are two different split proposals: one to symmetric values of $b_h$, and a “bud” move, where one component retains the old value of $b_h$ from the pre-split layer.
Example: synthetic data

- The test data set consisted of 500 genes with layers at 2.2, 3.4 and 4.7.

- There were also composite values at 5.6 ($= 2.2 + 3.4$), 6.9 ($= 2.2 + 4.7$), 8.1 ($= 3.4 + 4.7$) and 10.3 ($= 2.2 + 3.4 + 4.7$), as well as some values at 0.0.

- Though the sampler determined the correct number of layers with high probability, the locations of the layers was less well determined.
Layer switching

• I suspected that one reason for the poor performance of the sampler was the way that allocations were updated.

• Suppose that a gene with expression level 4.5 is allocated to a single layer whose $b$-value is around 3.4, and that there is another layer with $b$-value around 4.7.

• Under the current scheme, in order for this gene to be re-allocated to the layer at 4.7 alone, it must first either be allocated to no layers, or to both the layer at 3.4 and the one at 4.7.
Layer switching (cont.)

- Both of these options require moving through a state of low probability, so are unlikely to happen.

- In order to overcome this problem, I introduced a layer switching move.

- Rather than just updating each allocation variable in turn, we select a non-zero allocation variable and propose to ‘swap’ this allocation with that of another layer which is not currently allocated to that gene.

- After implementing this move the sampler reliably finds the correct layer-values (the values of $b_h$).
Repulsive $b$’s(!)

- We wanted to eliminate the possibility of getting several layers with very similar values of $b$ allocated to distinct groups of genes.

- To solve this problem we propose to use some sort of repulsion between values of $b$.

- Since I have worked with area-interaction point processes before (Baddeley and van Lieshout 1995), it seemed natural to me to use a 1-d area-interaction process for this purpose.
Repulsive $b$’s (cont.)

- The prior on the $b$’s and the $H$’s is no longer separable and is specified jointly as follows:

$$p(H, b) \propto \frac{\lambda^H}{H!} \gamma^{-m(B \oplus G)} \times \prod_{h=1}^{H} \exp\left(-\frac{1}{2\tau^2} b_h^2\right).$$

- The term $\gamma^{-m(B \oplus G)}$ is the area-interaction term, with $\gamma \in (0, 1)$ giving repulsion.

- $m$ is Lebesgue measure.

- $B \oplus G = \{b_h + g : h \in \{1, \ldots, H\}, \ g \in G\}$ and $G = [-r, r]$ for some $r$. Thus $B \oplus G$ is the set of all points within $r$ of at least one of the layer-values $b_h$. 
More programming...

- I have also implemented this additional feature.
- The $b$-updates are now ‘vanilla’ random-walk Metropolis moves.
- The sampler finds the correct number and composition of layers with high probability, but the performance is not significantly better than without repulsion.
The full model

- Having considered the single-sample case, we now extend the model to the multiple-sample case by adding per-sample allocations:

\[ y_{gs} = \sum_{h=1}^{H} v_{sh} z_{gh} b_h + \epsilon_{gs}, \]

where \( s \) = sample and \( v_{sh} \in \{0, 1\} \) is an allocation variable with a simple Bernoulli prior for the \( v \)'s.

- The layers are constant-height “rectangles”.

- This model is capable of describing basic differential expression of groups of genes between samples.
A toy example of the full model

• We next simulated a dataset containing 10 genes for each of 5 samples with 2 layers at 2.2 and 3.4:

<table>
<thead>
<tr>
<th></th>
<th>2.2</th>
<th>2.2</th>
<th>0</th>
<th>0</th>
<th>2.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2</td>
<td>5.6</td>
<td>3.4</td>
<td>3.4</td>
<td>2.2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>5.6</td>
<td>3.4</td>
<td>3.4</td>
<td>2.2</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>2.2</td>
<td>0</td>
<td>0</td>
<td>2.2</td>
<td></td>
</tr>
</tbody>
</table>

• We added $N(0, (\frac{1}{2})^2)$ noise to this dataset as well.

• The algorithm found the correct number and composition of the layers without difficulty.
A real dataset

- In order to demonstrate our method with a real dataset, we looked at the human fibroblast data introduced by Lemon et al. (2002).
- This data set consists of 18 samples split into 3 categories: serum starved, serum stimulated and a 50:50 mix of starved/stimulated.
Pre-processing

• We used the natural logarithm of Lemon et al.’s calculated LWF values as our measure of expression and subtracted gene and sample mean levels.

• We then selected the 100 most variable genes across all 18 samples and used this $18 \times 100$ array as the input to our sampler.

• The following plots use the expected number of common layers that a pair of samples (genes) are in to cluster the samples (genes).
Cluster Dendrogram for Samples

hclust (*, "single")
Sorted (using hclust) gene similarities
Cluster Dendrogram for Genes
Future work

- Work on intelligent methods for choosing the parameters.

- Investigate methods for displaying the posterior in a more visually-informative way.
