1 Data Discretization

We will apply NEMs to data from a study on innate immune response in *Drosophila* [1]. Selectively removing signaling components blocked induction of all, or only parts, of the transcriptional response to LPS. For preprocessing, a normalization on probe level using a variance stabilizing transformation [6], and probe set summarization using a median polish fit of an additive model was performed [7]. The result is included as a dataset in the package nem.

```r
> library(nem)
> data("BoutrosRNAi2002")
```

The data *BoutrosRNAiExpression* contains the normalized expression values and *BoutroRNAiDiscrete* contains a discretized version, which was created via the function `nem.discretize` as follows [9]: First, we select the genes as effect reporters (E-genes), which are more than two-fold upregulated by LPS treatment. Next, we transform the continuous expression data to binary values. We set an E-genes state in an RNAi experiment to 1 if its expression value is sufficiently far from the mean of the positive controls, i.e. if the intervention interrupted the information flow. If the E-genes expression is close to the mean of positive controls, we set its state to 0.

```r
> res.disc <- nem.discretize(BoutrosRNAiExpression, neg.control = 1:4,
+ pos.control = 5:8, nfold = 2, cutoff = 0.7)
```

Have a look at the data structure of `res.disc`. Is the data stored in `res.disc$dat` really the same as the one stored in `BoutroRNAiDiscrete`? What influence has the parameter `cutoff` on the result?

We make plot of both, the original as well as the discretized data, on the selected E-genes. For that purpose we first rearrange the E-genes a little bit using hierarchical clustering:

```r
> disc <- cbind(res.disc$neg, res.disc$pos, res.disc$dat)
> e.2fold <- BoutrosRNAiExpression[res.disc$sel, ]
> hc <- hclust(as.dist(hamming.distance(disc[, 9:16])))
> e.2fold <- e.2fold[hc$order, ]
> disc <- disc[hc$order, ]
> par(las = 2, mgp = c(5.5, 1, 0), mar = c(6.7, 7, 4, 1), cex.lab = 1.7,
```
> cex.main = 2)
> image(x = 1:ncol(e.2fold), y = 1:nrow(e.2fold), z = scale(t(e.2fold)),
> main = "Original data", xlab = "Experiments", xaxt = "n",
> ylab = "E-genes", yaxt = "n", col = gray(seq(0, 0.95, length = 10)))
> abline(v = c(4, 8, 10, 12, 14) + 0.5)
> axis(1, 1:ncol(e.2fold), colnames(e.2fold))
> axis(2, 1:nrow(e.2fold), rownames(e.2fold))
> x11()
> par(las = 2, mgp = c(5.5, 1, 0), mar = c(6.7, 7, 4, 1), cex.lab = 1.7,
+ cex.main = 2)
> image(x = 1:ncol(disc), z = t(disc), main = "Discretized data",
+ xlab = "Experiments", xaxt = "n", ylab = "", yaxt = "n",
+ col = gray(seq(0.95, 0, length = 10)))
> abline(v = c(4, 8, 10, 12, 14) + 0.5)
> axis(1, 1:ncol(e.2fold), colnames(e.2fold))

Play around with the cutoff value in the discretization procedure. How much
does the picture change?

2 Working with p-Value Densities

In general, performing a data discretization on the expression profiles as de-
scribed before can be critical, specifically, if not both, positive and negative
controls are available. An alternative is given by taking the raw p-value profiles
obtained from testing for differential gene expression (i.e. p-values without mul-
tiple testing adjustment). In this situation we assume the individual p-values
in the data matrix to be drawn from a mixture of a uniform, a Beta(1,\(\beta\)) and
a Beta(\(\alpha\),1) distribution. The parameters of the distribution are fitted via an
EM algorithm [3]. The function getDensityMatrix conveniently does all the
fitting of the p-value densities and produces diagnostic plots into a user specified
directory. We always recommend to use the full microarray without any filtering
for fitting the p-value densities, since filtering could destroy the supposed form
of the p-value distributions.

For the Drosophila immune response data, we have already stored a p-value
density matrix in the data BoutrosRNAiDens. The original p-values were calcu-
lated by comparing each knock-down against the positive control using limma
[10].

P-value densities can be interpreted as signal-to-noise ratios. A value \(> 1\)
means higher signal than noise, and a value \(< 1\) a higher noise than signal.

We first make a plot of the log-densities to see differences regarding the
pattern we have discovered in the discretized data:

> dens = BoutrosRNAiDens[hc$order, ]
> par(las = 2, mgp = c(5.5, 1, 0), mar = c(6.7, 7, 4, 1), cex.lab = 1.7,
+ cex.main = 2)
> image(x = 1:ncol(dens), y = 1:nrow(dens), z = scale(t(dens)),
+ main = "Original data", xlab = "Experiments", xaxt = "n",
+ ylab = "E-genes", yaxt = "n", col = gray(seq(0, 0.95, length = 10)))
> abline(v = c(4, 8, 10, 12, 14) + 0.5)
We are now in the position to infer a NEM model from data. This is done via the function \texttt{nem}. The function has a hyperparameter \texttt{controls} (a list object in R), which controls the workflow of this function in several ways. All values are initialized to some default values using the function \texttt{set.default.parameters}:

```r
D = BoutrosRNAiDiscrete[, 9:16]
control = set.default.parameters(unique(colnames(D)))
control
```

We may need to specify certain parameters a little bit more detailed. Most importantly this includes the appropriate likelihood model for our data, which is specified in list item \texttt{type} and can have one of the following options:

- \texttt{mLL, FULLmLL}: marginal likelihood and full marginal likelihood for discrete data \cite{9}.
- \texttt{CONTmLLBayes}: marginal likelihood for p-value log-densities \cite{3}.
- \texttt{CONTmLLMAP}: likelihood for p-value log-densities or other kinds of log ratio data using MAP estimates of E-gene positions \cite{11}.

The default choice is \texttt{type = "mLL"}. Except for \texttt{CONTmLLMAP} all likelihood models operate within the equivalence class of transitively closed graphs, i.e. two graph structures cannot be distinguished, if they only differ by transitive edges.

We now need to define an algorithm for searching for a high scoring network structure in graph space. For this purpose there are several algorithms available in parameter \texttt{inference} of the \texttt{nem} function:

- \texttt{search}: exhaustive enumeration of all possible network structures
- \texttt{nem.greedy}: greedy hill climbing in graph space \cite{4}
- \texttt{ModuleNetwork}: fast divide and conquer algorithm \cite{2, 3}
- \texttt{pairwise, triples}: inference from all combinations of S-gene pairs or triplets \cite{8}
- \texttt{nem.greedyMAP}: fast alternating MAP optimization scheme of graph structure and E-gene positions \cite{11}. This algorithm only works in combination with \texttt{type = "CONTmLLMAP"}.

There is also the possibility to allow for an automated subset selection of relevant E-genes, i.e. those E-genes supporting the network hypothesis most:

```r
control = set.default.parameters(unique(colnames(D)), selEGenes = TRUE)
mynem = nem(D, control = control)
mynem
```

We can visualize the estimated network via:
For larger networks it would also make sense to visualize the graph in terms of strongly connected components:

```r
> plot.nem(SCCgraph(mynem$graph))
```

It is further interesting to have a look at the posterior E-gene positions:

```r
> plot.nem(mynem, what = "pos")
```

or alternatively via:

```r
> plot.nem(mynem, SCC = FALSE, D = D, draw.lines = TRUE)
```

which additionally depicts the assignment of E-genes to S-genes.

Investigate, how certain combinations of likelihood models and search algorithms influence these results.

### 4 Statistical Stability and Significance

Looking at one particular network at a local maximum of the (marginal) likelihood neglects the uncertainty we have about particular edges. While some edges may be visible in the data quite clearly, others may vanish or become spurious, if we perturb our data only slightly. We therefore advise to perform a (non-parametric) bootstrap to assess the statistical stability of each edge and only keep those, which have a bootstrap frequency larger than some threshold (e.g. 0.5). For time reasons we here only use 100 bootstrap samples and no automated selection of relevant E-genes:

```r
> mynem.boot = nem.bootstrap(D, nboot = 100)
> plot.nem(mynem.boot, SCC = FALSE, plot.probs = TRUE)
```

The bootstrap probabilities are depicted as edge labels.

Another useful idea, is to compare a network hypothesis against a random network in some way. This can, for instance, be done by permuting the node labels and comparing the likelihood against the original likelihood. Another implemented test is to randomly insert or delete 1 edge and look at the change of the likelihood. Finally, we also implemented a random sampling of graph structures to which we can compare our network hypothesis. Here, for time reasons we only run 100 permutations / random network drawings and calculate the empirical p-value:

```r
> nem.calcSignificance(D, mynem, N = 100)
```

Investigate, whether the picture changes when using different algorithms and likelihood models.

### 5 Integrating Prior Knowledge

The `nem` package allows to specify a prior on the network structure itself. This can be thought of biasing the score of possible network hypotheses towards prior knowledge. It is crucial to understand that in principle in any inference scheme
there exist two competing goals: Belief in prior assumptions / prior knowledge versus belief in data. Only trusting the data itself may lead to overfitting, whereas only trusting in prior assumptions does not give any new information and prevents learning. Therefore, we need a trade-off between both goals via a regularization constant $\lambda > 0$, which specifies the belief in our prior assumptions. In the simplest case our assumption could be that the true network structure is sparse, i.e. there are only very few edges. More complex priors could involve separate prior edge probabilities (c.f. [2, 3]).

In practice we would like to choose $\lambda$ in an automated fashion. This leads to an instance of the classical model selection problem (e.g. [5]) in statistical learning. One practical way of dealing with it is to tune $\lambda$ such that the Bayesian information criterion (BIC)

$$BIC(\lambda, \Phi_{opt}) = -2 \log P(D|\Phi_{opt}) + \log(n)d(\lambda, \Phi_{opt})$$

(1)

becomes minimal [5]. Here $d(\lambda, \Phi_{opt})$ denotes the number of parameters in the highest scoring network structure $\Phi_{opt}$ haven $n$ S-genes. Here the number of parameters is estimated as:

$$d(\lambda, \Phi) = \sum_{i,j} 1\{|\Phi_{ij} - \hat{\Phi}_{ij}| > 0\}$$

(2)

where $\hat{\Phi}$ is the prior network.

Searching for an optimal regularization constant relates to a frequentistic point of view to incorporate prior knowledge. Instead, from a Bayesian perspective one should define a prior on the regularization parameter and integrate it out. Here, this is done by assuming an inverse gamma distribution prior on $\nu = \frac{1}{\lambda^2}$ with hyperparameters $(1, 0.5)$, which leads to a simple closed form of the full prior [3]. An advantage of the Bayesian perspective is that no explicit model selection step is needed. Furthermore, there is evidence, that compared to the frequentistic method the Bayesian approach using the same amount of prior knowledge yields a higher increase of the reconstructed network’s sensitivity

As an example let us assume we have knowledge on the edges $tak1 \rightarrow key$ and $tak1 \rightarrow mkk4/hep$, whereas the rest of the network we would like to keep as sparse as possible.

```R
> control$Pm = rbind(c(0, 0, 0, 0), c(0, 0, 0, 0), c(0, 1, 0, 1),
+ c(0, 0, 0, 0))
> resultModelSel = nemModelSelection(c(0.01, 0.1, 1, 10, 100), D,
+ control = control, verbose = FALSE)
> resultBayes = nem(D, control = control, verbose = FALSE)
```

Investigate the obtained network hypothesis. How does the picture change, if the model selection is restricted to another set of possible regularization parameters? Is there an influence of the employed likelihood scheme and network reconstruction algorithm?

References


