Manual for the R gaga library

David Rossell
Department of Biostatistics
The University of Texas M. D. Anderson Cancer Center
Houston, TX.
rosselldavid@gmail.com

1 Introduction

Newton et al. (2001) and Kendziorski et al. (2003) introduced the Gamma-Gamma model to analyze microarray data, an elegant and parsimonious hierachical model that allows for the borrowing of information between genes. Rossell (2007) showed that the assumptions of this model are too simplistic, which resulted in a rather poor fit to several real datasets, and developed two extensions of the model: GaGa and MiGaGa. The gaga library implements the GaGa and MiGaGa generalizations, which can be used both to find differentially expressed genes and to predict the class of a future sample (*e.g.* given the mRNA measurements for a new patient, predict whether the patient has cancer or is healthy).

We now briefly outline the GaGa and MiGaGa models. Let x_{ij} be the expression measurement for gene i in array j, and let z_j indicate the group to which array belongs to (*e.g.* $z_j = 0$ for normal cells and $z_j = 1$ for cancer cells). The GaGa models envisions the observations as arising from a gamma distribution, *i.e.* $x_{ij} \sim \text{Ga}(\alpha_{i,z_j}, \alpha_{i,z_j}/\lambda_{i,z_j})$ (λ_{i,z_j} is the mean), where α_{i,z_j} and λ_{i,z_j} arise from a gamma and an inverse gamma distribution, respectively:

$$\begin{aligned} \lambda_{i,k}|\delta_i, \alpha_0, \nu &\sim \text{IGa}(\alpha_0, \alpha_0/\nu), \text{ indep. for } i = 1 \ldots n \\ \alpha_{i,k}|\delta_i, \beta, \mu &\sim \text{Ga}(\beta, \beta/\mu), \text{ indep. for } i = 1 \ldots n \\ \delta_i|\boldsymbol{\pi} &\sim \text{Mult}(1, \boldsymbol{\pi}), \text{ indep. for } i = 1 \ldots n. \end{aligned} \quad (1)$$

$\delta_1 \ldots \delta_n$ are latent variables indicating what expression pattern each gene follows (see Section 3 for more details). For example, if there are only two groups δ_i indicates whether gene i is differentially expressed or not. The

Bayesian model is completed by specifying priors on the hyper-parameters that govern the hierarchy:

$$\begin{aligned} \alpha_0 &\sim \text{Ga}(a_{\alpha_0}, b_{\alpha_0}); \nu \sim \text{IGa}(a_\nu, b_\nu) \\ \beta &\sim \text{Ga}(a_\beta, b_\beta); \mu \sim \text{IGa}(a_\mu, b_\mu) \\ \boldsymbol{\pi} &\sim \text{Dirichlet}(\mathbf{p}). \end{aligned} \tag{2}$$

The `gaga` library provides some default values for the prior parameters that are a reasonable choice when the data has been normalized via the function `just.rma` from the R library `affy` Irizarry et al. (2005) or `just.gcrma` from the R library `just.gcrma`. The MiGaGa model extends GaGa by specifying a mixture of inverse gammas for ν. Both models are fit using the routine `fit.gg`: the argument `nclust` indicates the number of components in the mixture (`nclust=1` corresponds to the GaGa model).

In the remainder of this document we generate a simulated dataset and we show how to analyze it with the routines provided with the `gaga` library. In Section 2 we simulate a dataset based on the parameter estimates obtained from the Armstrong dataset Armstrong et al. (2002), as described in Rossell (2007). Section 3 shows how to fit the model via MCMC sampling and in Section 4 we assess its goodness-of-fit. Finally, in Sections 5 and 6 we conduct inference. Section 5 shows how to find differentially expressed genes, while Section 6 addresses class prediction.

2 Simulating the data

We start by loading the library and simulating mRNA expression levels for `n=100` genes and 2 groups, each with 6 samples. We set the seed for random number generation so that you can reproduce the results presented here. As we shall see in the future sections, we use the first five samples from each group to fit the model. We will then use the model to predict the class for the sixth sample.

```
> library(gaga)
> set.seed(10)
> n <- 100
> m <- c(6, 6)
> a0 <- 25.5
> nu <- 0.109
> balpha <- 1.183
> nualpha <- 1683
```

```
> probpat <- c(0.95, 0.05)
> xsim <- sim.gg(n, m, p.de = probpat[2], a0, nu, balpha, nualpha)
```

The object `xsim` is a list with 3 components. `x` is a matrix containing the expression levels for the 100 genes and 12 samples. `a` is a matrix containing the gene-specific α parameters (`a[,1]` contains parameters for the first group, `a[,2]` for the second), and `l` contains the gene-specific means λ.

```
> names(xsim)

[1] "x" "a" "l"

> dim(xsim$x)

[1] 100  12

> dim(xsim$a)

[1] 100   2

> dim(xsim$l)

[1] 100   2
```

Figure 1(a) shows the marginal distribution (kernel density estimate) of the simulated gene expression levels. Figure 1(b) plots the simulated (α, λ) pairs. The plots can be obtained with the following syntax:

```
> plot(density(xsim$x), xlab = "Expression levels", main = "")

> plot(xsim$l, xsim$a, xlab = "Mean", ylab = "1/sqrt(CV)")
```

3 Model fit

To fit the model we use the function `fit.gg`. First we need to specify a vector `groups` that indicates the group to which each sample belongs to. Second, we need to specify the gene expression patterns or hypotheses that we wish to entertain. In our example, since we have two groups there really are only two possible expression patterns:

Pattern 0 (null hypotheses): group 1 = group 2

(a) (b)

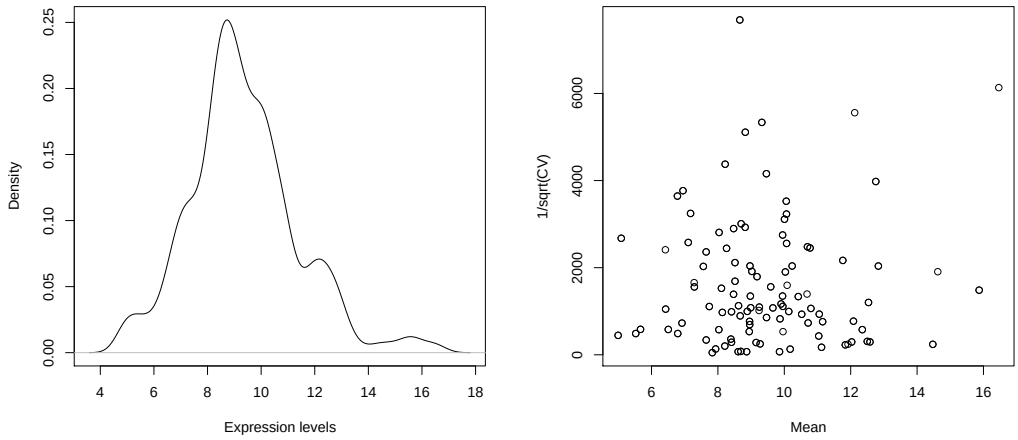

Figure 1: (a): marginal density of the simulated data; (b): plot of the simulated (α, λ) pairs

Pattern 1 (alternative hypothesis): group 1 $\neq$ group 2.

More precisely, under pattern 0 we have that $\alpha_{i1} = \alpha_{i2}$ and $\lambda_{i1} = \lambda_{i2}$, while under pattern 1 $\alpha_{i1} \neq \alpha_{i2}$ and $\lambda_{i2} \neq \lambda_{i2}$. We specify the patterns with a matrix with as many rows as patterns and as many columns as groups. For each row of the matrix (*i.e.* each hypothesis), we indicate that two groups are equal by assigning the same number to their corresponding columns. For example, in our two hypothesis case we would specify:

```
> groups <- rep(1:2, each = 5)
> patterns <- matrix(c(0, 0, 0, 1), 2, 2)
> patterns
```

```
     [,1] [,2]
[1,]    0    0
[2,]    0    1
```

For illustration, suppose that instead we had 3 groups and 4 hypotheses, as follows:

Pattern 0: GROUP 1 = GROUP 2 = GROUP 3

Pattern 1: GROUP 1 $\neq$ GROUP 2 = GROUP 3

Pattern 2: GROUP 1 = GROUP 2 $\neq$ GROUP 3

Pattern 3: GROUP 1 $\neq$ GROUP 2 $\neq$ GROUP 3

In this case we would specify

```
> patterns <- matrix(c(0,0,0,0,1,1,0,0,1,0,1,2),ncol=3,byrow=TRUE)
> patterns
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    1    1
[3,]    0    0    1
[4,]    0    1    2
```

That is, the second row indicates that under the second hypothesis groups 2 and 3 are equal, since they both have a `1` in their entries. The last row indicates that they are all different by specifying a different number in each entry.

Now, to fit the GaGa model to our simulated data we use `fit.gg`, with `nclust=1` (to fit the MiGaGa model we would set `nclust` to the number of components that we want in the mixture). We remove columns 6 and 12 from the dataset, *i.e.* we do not use them for the fit so that we can evaluate the out-of-sample behavior of the classifier built in Section 6. Here we use the option `trace=FALSE` to prevent iteration information from being printed. By default the routine performs `B=1000` MCMC iterations, which is typically enough for a GaGa model and it can be done reasonably fast.

```
> patterns <- matrix(c(0, 0, 0, 1), 2, 2)
> ggfit <- fit.gg(xsim$x[, c(-6, -12)], groups, patterns = patterns,
+     nclust = 1, trace = FALSE)
```

We can obtain iteration plots to visually assess the convergence of the chain. The component `mcmc` of `ggfit` contains an object of type `mcmc`, as defined in the library `coda`. Therefore, simply typing `plot(ggfit$mcmc)` produces the plots in Figure 2 (only first plot shown). For more formal convergence diagnostics, see the documentation of the library `coda`. We see that convergence was reached very quickly.

```
> plot(ggfit$mcmc)
```

To obtain parameter estimates and the posterior probability that each gene is differentially expressed we use the function `parest`. We discard the first 100 MCMC iterations with `burnin=100`, and we ask for 95% posterior credibility intervals with `alpha=.05`. The slot `ci` of the returned object contains the credibility intervals. Note that they all include the true value.

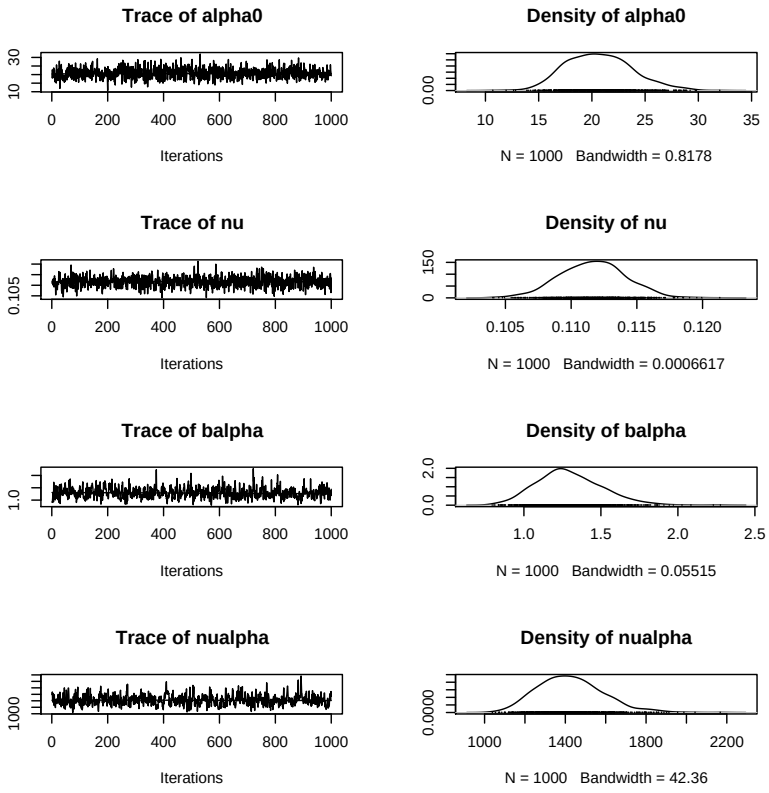

Figure 2: Plotting MCMC output

```
> ggfit <- parest(ggfit, x = xsim$x[, c(-6, -12)], groups, burnin = 100,
+     alpha = 0.05)
> ggfit

GaGa hierarchical model. Fit via MCMC (900 iterations kept)
  100 genes, 2 groups, 2 hypotheses (expression patterns)

The expression patterns are
  Pattern 0: Group 1 = Group 2
  Pattern 1: Group 1 !=Group 2

Hyper-parameter estimates

   a0 nu balpha nualpha
   20.742 0.112 1.295 1425.516

   probclus
   1

   probpat.1 probpat.2
   0.942 0.058

> ggfit$ci

$a0
    2.5%    97.5%
15.45334 27.13702

$nu
     2.5%     97.5%
0.1067497 0.1163545

$balpha
     2.5%     97.5%
0.9419924 1.7545734

$nualpha
    2.5%    97.5%
1148.298 1788.879

$probclus
```

```
[1] 1 1

$probpat
      probpat.1  probpat.2
2.5%  0.8767555 0.01578948
97.5% 0.9842105 0.12324451
```

As an alternative, one can estimate the hyper-parameters via maximum likelihood. This is achieved by specifying `method='EBayes'` when calling `fit.gg`. Fitting the model via maximum likelihood is usually quicker, and in our experience it delivers similar results to the Bayesian fit.

```
> ggfit.eb <- fit.gg(xsim$x[, c(-6, -12)], groups, patterns = patterns,
+     method = "EBayes")

Initializing parameters... Done.
Refining initial estimates... Done.
Starting EM algorithm...
Iter       a0       nu   balpha  nualpha              prob log-likelihood
   1 20.93963 0.111481 1.317409 1404.693 0.940913 0.059087 1569.088
   2 20.97746 0.111468 1.319855 1398.438 0.939806 0.060194 1569.090
   3 20.97833 0.11149 1.320031 1398.539 0.939511 0.060489 1569.090
   4 20.97833 0.11149 1.320031 1398.539 0.939436 0.060564 1569.090

> ggfit.eb <- parest(ggfit.eb, x = xsim$x[, c(-6, -12)], groups,
+     alpha = 0.05)
> ggfit.eb

GaGa hierarchical model. Fit via empirical Bayes
  100 genes, 2 groups, 2 hypotheses (expression patterns)

The expression patterns are
  Pattern 0: Group 1 = Group 2
  Pattern 1: Group 1 !=Group 2

Hyper-parameter estimates

   alpha0 nu balpha nualpha
   20.978 0.111 1.32 1398.539

   probclus
   1
```

```
probpat1 probpat2
0.939 0.061
```

The parameter estimates are indeed similar to those obtained from the fully Bayesian fit. The slot `pp` in `ggfit` and `ggfit.eb` contains a matrix with the posterior probability of each expression pattern for each gene. For example, the first gene has a posterior probability of 0.993 of following pattern 0 (*i.e.* being equally expressed) and a probability of 0.033 of following pattern 1 (*i.e.* being differentially expressed) under the fully Bayesian fit, and 0.034 under the maximum likelihood fit.

```
> dim(ggfit$pp)

[1] 100   2

> ggfit$pp[1, ]

[1] 0.96742117 0.03257883

> ggfit.eb$pp[1, ]

[1] 0.96633383 0.03366617
```

4 Checking the goodness of fit

To graphically assess the goodness of fit of the model, one can used prior-predictive or posterior-predictive checks. The latter, implemented in the function `checkfit`, are based on drawing parameter values from the posterior distribution for each gene, and possibly using then to generate data values, and then compare the simulated values to the observed data. The data generated from the posterior predictive is compared to the observed data in Figure 2(a). Figure 2(b)-(d) compares draws from the posterior of α and λ with their method of moments estimate, which is model-free. All plots indicate that the model has a reasonably good fit. The figures were generated with the following code:

```
> checkfit(ggfit, x = xsim$x[, c(-6, -12)], groups, type = "data",
+     main = "")

> checkfit(ggfit, x = xsim$x[, c(-6, -12)], groups, type = "shape",
+     main = "")
```

```
> checkfit(ggfit, x = xsim$x[, c(-6, -12)], groups, type = "mean",
+     main = "")

> checkfit(ggfit, x = xsim$x[, c(-6, -12)], groups, type = "shapemean",
+     main = "", xlab = "Mean", ylab = "1/sqrt(CV)")
```

It should be noted, however, that posterior-predictive plots can fail to detect departures from the model, since there is a double use of the data. Prior-predictive checks can be easily implemented using the function `sim.gg` and setting the hyper-parameters to their posterior mean.

5 Finding differentially expressed genes

The function `findgenes` finds differentially expressed genes, *i.e.* assigns each gene to an expression pattern. The problem is formalized as minizing the false negative rate, subject to an upper bound on the false discovery rate, say `fdrmax=0.05`. In a Bayesian sense, this is achieved by assigning to pattern 0 (null hypothesis) all genes for which the posterior probability of following pattern 0 is above a certain threshold Müller et al. (2004). The problem is then to find the optimal threshold, which can be done parametrically or non-parametrically through the use of permutations (for details see Rossell (2007)). Here we explore both options, specifying `B=1000` permutations for the non-parametric option.

```
> d <- findgenes(ggfit, xsim$x[, c(-6, -12)], groups, fdrmax = 0.05,
+     parametric = TRUE)
> d.nonpar <- findgenes(ggfit, xsim$x[, c(-6, -12)], groups, fdrmax = 0.05,
+     parametric = FALSE, B = 1000)

Finding clusters of z-scores for bootstrap... Done
Starting 1000 bootstrap iterations...

> dtrue <- (xsim$l[, 1] != xsim$l[, 2])
> table(d$d, dtrue)

   dtrue
    FALSE TRUE
  0    95    1
  1     0    4

> table(d.nonpar$d, dtrue)
```

(a) (b)

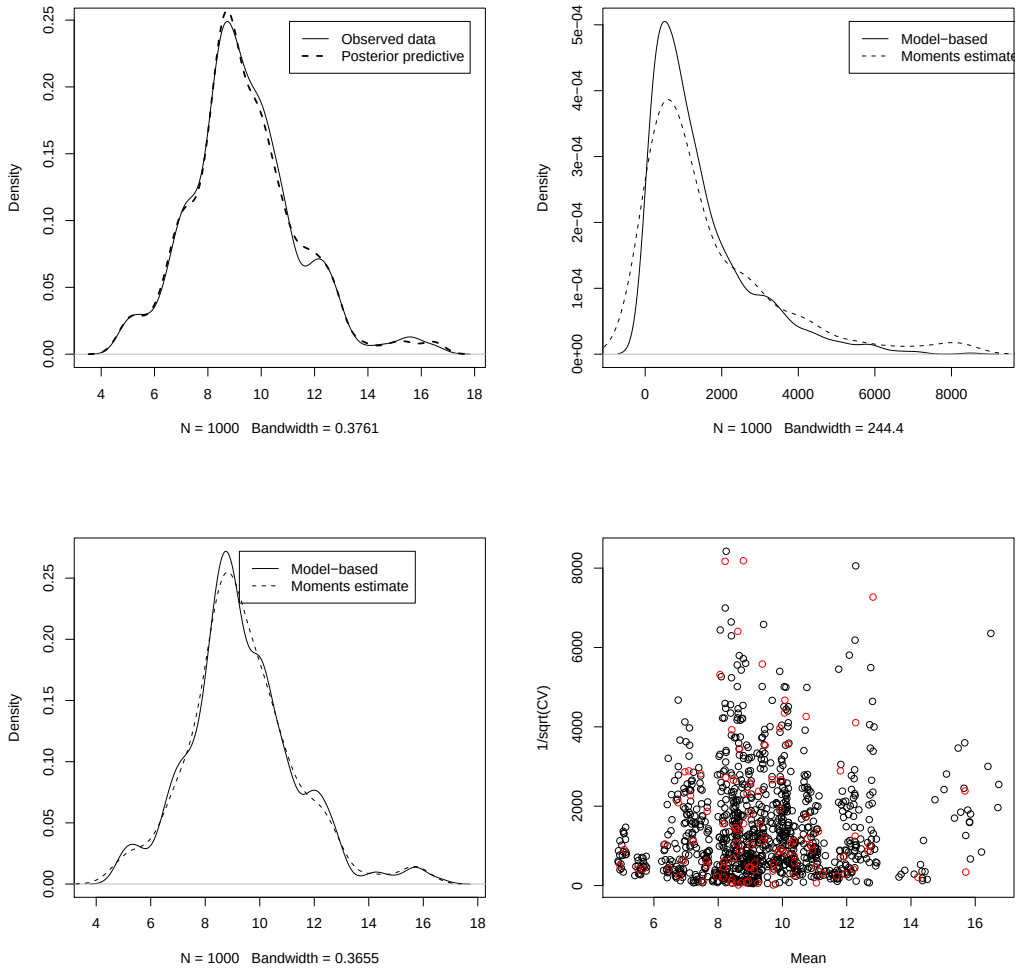

Figure 3: Assessing the goodness of fit. (a): compares samples from the posterior predictive to the observed data; (b): compares samples from the posterior of α to the method of moments estimate; (c): compares samples from the posterior of λ to the method of moments estimate; (d): as (b) and (c) but plots the pairs (α, λ) instead of the kernel density estimates

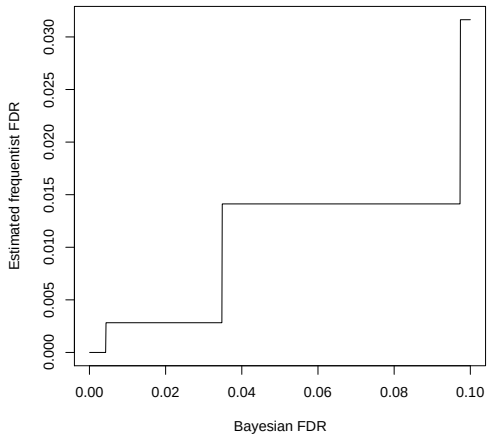

Figure 4: Estimated frequenstist FDR vs. Bayesian FDR

```
  dtrue
   FALSE TRUE
0     94    1
1      1    4
```

We set the variable `dtrue` to indicate which genes were actually differentially expressed (easily achieved by comparing the columns of `xsim$l`). The parametric version declares 4 genes to be DE, all of them true positives. It fails to find one of the DE genes. The non-parametric lists 5 genes as DE, one of them being a false positive. It fails to find one of the DE genes. To obtain an estimated frequentist FDR for each Bayesian FDR one can plot `d.nonpar$fdrest`. The result, shown in Figure 3, reveals that setting the Bayesian FDR at a 0.05 level results in an estimated frequentist FDR around 0.015. That is, calling `findgenes` with the option `parametric=TRUE` results in a slightly conservative procedure from a frequentist point of view.

```
> plot(d.nonpar$fdrest, type = "l", xlab = "Bayesian FDR", ylab = "Estimated fr
```

6 Class prediction

We now use the fitted model to predict the class of the arrays number 6 and 12, neither of which were used to fit the model. We assume that the prior

probability is 0.5 for each group, though in most settings this will not be realistical. For example, if `groups==2` indicates individuals with cancer, one would expect the prior probability to be well below 0.5, say around 0.1. But if the individual had a positive result in some test that was administered previously, this probability would have increased, say to 0.4.

Class prediction is implemented in the function `classpred`. The argument `xnew` contains the gene expression measurements for the new individuals, `x` is the data used to fit the model and `ngene` indicates the number of genes that should be used to build the classifier. It turns out that array 6 is correctly assigned to group 1 and array 12 is correctly assigned to group 2. `classpred` also returns the posterior probability that the sample belongs to each group. We see that for the dataset at hand the posterior probability of belonging to the wrong group is essentially zero. Similarly good results are obtained when using setting `ngene` to either 1 (the minimum value) or to 100 (the maximum value). The fact that adding more gene to the classifier does not change its performance is not surprising, since the classifier assigns little weight to genes with small probability of being DE. We have observed a similar behavior in many datasets. The fact that the classifier works so well with a single is typically not observed in real datasets, where it is rare to have a gene with such a high discrimination power.

```
> pred1 <- classpred(ggfit, xnew = xsim$x[, 6], x = xsim$x[, c(-6,
+     -12)], groups, ngene = 50, prgroups = c(0.5, 0.5))
> pred2 <- classpred(ggfit, xnew = xsim$x[, 12], x = xsim$x[, c(-6,
+     -12)], groups, ngene = 50, prgroups = c(0.5, 0.5))
> pred1

$d
[1] 1

$posgroups
[1] 1.000000e+00 1.858797e-17

> pred2

$d
[1] 2

$posgroups
[1] 1.831219e-17 1.000000e+00
```

References

S.A. Armstrong, J.E. Staunton, L.B. Silverman, R. Pieters, M.L. Boer, M.D. Minden, E.S. Sallan, E.S. Lander, T.R. Golub, and S.J. Korsmeyer. Mll translocations specify a distinct gene expression profile that distinguishes a unique leukemia. *Nature Genetics*, 30:41–47, 2002.

R.A. Irizarry, L. Gautier, B.M. Bolstad, M. Miller, C. with contributions from Astrand, L.M. Cope, R. Gentleman, J. Gentry, C. Halling, W. Huber, MacDonald J., B.I.P Rubinstein, C. Workman, and J. Zhang. *affy: Methods for Affymetrix Oligonucleotide Arrays*, 2005. R package version 1.8.1.

C.M. Kendziorski, M.A. Newton, H. Lan, and M.N. Gould. On parametric empirical bayes methods for comparing multiple groups using replicated gene expression profiles. *Statistics in Medicine*, 22:3899–3914, 2003.

P. Müller, G. Parmigiani, C. Robert, and J. Rousseau. Optimal sample size for multiple testing: the case of gene expression microarrays. *Journal of the American Statistical Association*, 99:990–1001, 2004.

M.A. Newton, C.M. Kendziorski, C.S Richmond, F.R. Blattner, and K.W. Tsui. On differential variability of expression ratios: Improving statistical inference about gene expression changes from microarray data. *Journal of Computational Biology*, 8:37–52, 2001.

D. Rossell. GaGa: a simple and flexible hierarchical model for microarray data analysis. Technical report, M.D. Anderson cancer center, 2007. URL `http://rosselldavid.googlepages.com`.