

metaArray package for meta-analysis of microarray data

Debashis Ghosh and Hyungwon Choi

May 12, 2008

Introduction

metaArray is a collection of functions for large-scale meta-analysis of microarray data. The implementation embodies the latent variable model approach described in Choi et al. (2005). The package is an ensemble of statistical tools from several research papers: 1) probability of expression (POE) with two estimation methods (Parmigiani et al., 2002; Choi et al., 2005), 2) integrative correlation (Parmigiani et al., 2004), 3) posterior mean differential expression (z -statistic) (Wang et al., 2004).

As the use of this package is for meta-analysis of microarray data, this documentation will demonstrate the features of the package with an example of meta-analysis in cancer microarray data, the comparison of expression profiles in metastatic tumors and primary tumors. We gathered three microarray data from public databases, and the reference of the four related publications can be found in the bibliography.

The synopsis of the analysis is straightforward: we transform each data into the scale of $[-1, 1]$ by POE, filter genes based on integrative correlation, merge them into a single data, and perform final downstream analysis.

Data preparation and POE transformation

The object `mdata` in *metaArray* contains the four datasets mentioned above. The main goal of this analysis is to construct a signature of genes that distinguishes metastatic tumors from primary tumors with no future development into other types of cancers.

The data included in the package went through some refinement. First of all, all three datasets have been transformed to POE scale using MCMC method. For we selected only the genes that appear in all three datasets, `mdata` contains 500 common and unique Unigene Cluster IDs. In realistic situation, this is hardly the case, thus here we assume the four data have non-overlapping genes. The package *MergeMaid* is particularly useful for merging such datasets with partially overlapping gene sets, or it can be done manually with a few more lines of command as shown below.

In the following, let us first find the common genes and reorder genes in the order of appearance in the common geneset, transform each data to POE and store them into a

single object of class `mergeExprs`. It takes 15 minutes on average to transform each of the three datasets (500 genes) to POE scale using MCMC on 32-bit dual-processor Xeon (Intel x86) box.

```
> library(metaArray)
> library(Biobase)
> library(MergeMaid)
> data(mdata)
> common <- intersect(rownames(chen.poe), rownames(garber.poe))
> common <- intersect(common, rownames(lapointe.poe))
> chen.poe <- chen.poe[match(common, rownames(chen.poe)), ]
> garber.poe <- garber.poe[match(common, rownames(garber.poe)),
+   ]
> lapointe.poe <- lapointe.poe[match(common, rownames(lapointe.poe)),
+   ]
> vars <- list("var1", "var2")
> names(vars) <- names(chen.spl)
> pdata1 <- new("AnnotatedDataFrame")
> pData(pdata1) <- chen.spl
> varLabels(pdata1) <- vars
> sample1 <- new("ExpressionSet", exprs = chen.poe, phenoData = pdata1)
> names(vars) <- names(garber.spl)
> pdata2 <- new("AnnotatedDataFrame")
> pData(pdata2) <- garber.spl
> varLabels(pdata2) <- vars
> sample2 <- new("ExpressionSet", exprs = garber.poe, phenoData = pdata2)
> names(vars) <- names(lapointe.spl)
> pdata3 <- new("AnnotatedDataFrame")
> pData(pdata3) <- lapointe.spl
> varLabels(pdata3) <- vars
> sample3 <- new("ExpressionSet", exprs = lapointe.poe, phenoData = pdata3)
> merged <- mergeExprs(sample1, sample2, sample3)
```

Now the object `merged` contains four expression data with corresponding phenotype data. If phenotypic label is either unknown or not of interest, one can skip specifying `NN` in the `poe.mcmc` function. The POE transformation using MCMC algorithm runs fairly long time, thus if fast computing is of concern, you may wish to use `poe.em` function instead of `poe.mcmc` function. We will further elaborate on the usage of the two functions shortly. If you prefer application of MCMC, we strongly encourage you to run this in batch mode (Linux/Unix). An example syntax is

```
genome> R --no-save < myscript.R &
```

where `myscript.R` file contains the portion of the R code for POE transformation with `save.image` function attached at the end or inserted intermittently for safety.

Gene Filter and Downstream Analysis

The integrative correlation analysis (Parmigiani et al., 2004) is a convenient tool to monitor the interstudy concordance of within-study correlations of gene expression. The gene-specific reproducibility score takes the correlation between each gene and all other genes within individual study and calculate the average correlation of these correlations across all pairs of studies.

```
> merged.cor <- intcor(merged)$avg.cor
> mData <- exprs(intersection(merged))
> mCl <- NULL
> mData <- mData[merged.cor > median(merged.cor), ]
> for (i in 1:length(merged)) {
+   mCl <- c(mCl, pData(merged[i])$metastasis)
+ }
```

The function `intcor` is a modified version of `intCor` in *MergeMaid*. The original function efficiently calculates gene specific reproducibility scores by calculating correlation matrix only once so that it does not calculate correlation between two genes redundantly many times. When there exists a large common geneset among studies, R cannot allocate enough storage to the correlation matrix if it is of dimension higher than certain limit. Thus a brute-force computation of within-study correlation is inevitable for a large dataset, roughly around 2,000 common genes or more. The modified function `intcor` does this job without running into memory allocation problem regardless of size of common geneset.

The above script hence calculates the gene specific reproducibility scores and filters the genes with higher score. The function `intersection` from *MergeMaid* merges all four data into a single `data.frame` object `mdata`. The class labels, the indicator of metastasis in our example, can also be saved into a single numeric vector as illustrated in the code above.

Once the final data is formed, it is at the user's discretion to determine what downstream analysis is to be done. For instance, a simple two-sample *t*-statistic can be calculated easily (*multtest*). The risk-index approach in Choi et al. (2005) is another option, and the R script for this method can be found in <http://www.umich.edu/~hwchoi/metaArray.html>

Estimation of POE: MCMC versus EM

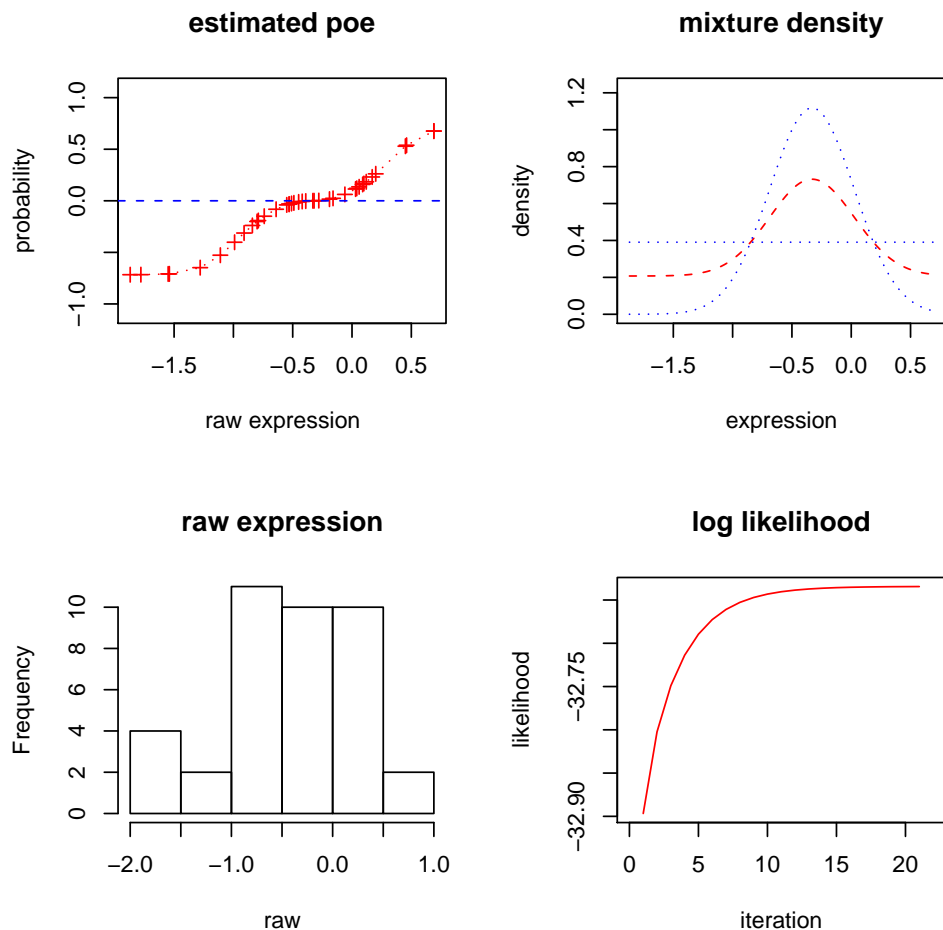
The detailed account of estimation with Gibbs sampling can be found in Parmigiani et al. (2002) and Choi et al. (2005). The estimation with EM algorithm is also explained in the latter.

The POE with Gibbs sampling was initially implemented in Bioconductor package *POE*. Despite its novel structure, the functions `poe.fit` and `poe.one.iteration` in *POE* were written purely in R, and thus the order of computation for MCMC with large number of parameters becomes unmanageable quickly as the size of dataset grows. Under such context, we rewrote the `poe.one.iteration` in C language to boost the speed of posterior sampling, while directly importing the main skeleton of `poe.fit` function without modification. Hence all credits for the implementation remain to the authors of *POE* package, and the users interested in the function `poe.mcmc` should be referred to the vignette included in *POE*.

Meanwhile, the `poe.em` is a new function. This function is faster than `poe.mcmc`, but has the defect that it fits the probability model for each gene separately. One of the novel features of MCMC method is that the estimation for every single gene borrows information across all genes through posterior sampling of sample specific means and other hyperparameters.

Another feature of the method using EM algorithm is that one can graphically inspect the fit for individual genes. See Figure 1 for an example. The upper left panel shows the fitted POE values plotted against the raw expression. The upper right panel shows the fitted mixture distribution for that gene. The lower left panel is the histogram of raw expression, and finally the lower right panel shows the trajectory of likelihood during the course of estimation.

```
> em.draw(as.numeric(chen[1, ]), cl = ncol(chen))
```



The computation burden is much lighter for EM algorithm than MCMC. Run time is consequently shorter. Only a few minutes or less is required for very large datasets, for instance, ten thousand genes and one hundred samples.

Posterior Mean Differential Expression

The last tool in this package is an implementation of the method illustrated in Wang et al. (2004). The main idea is that one can use data from one study to construct a prior distribution of differential expression and thus utilize the posterior mean differential expression, weighted by variances, whose distribution is standard normal distribution due to classic Bayesian probability calculation. To apply this method,

```
> z.stat <- Zscore(merged)
```

where `merged` is a `mergeExprSet` object created the same way earlier, but without the POE transformation. The resulting vector of z -scores may be compared to other downstream

analysis based on POE transformation. If one wishes to generate permutation distribution to determine significance of the observed statistic, replacing `permute=0` with `permute=10000` will generate permutation distribution of z -statistic by permuting class labels within all studies 10,000 times..

Discussion

We have implemented an ensemble of statistical techniques that assist certain aspects of meta-analysis of microarray data. The main thrust of this package is the set of two estimation methods for the data transformation into probability of expression (POE).

There are other methods and software packages that implement meta-analytic methods from other research papers. For example, a model based effect-size approach from Choi et al. (2003) was embodied in Bioconductor package *GeneMeta* by Lusa et al. (2004). We recommend users to compare sensitivity of the analyses by using a variety of approaches as they become available.

The strengths of our implementation are the following. 1) it allows us to combine multiple data with denoising effect and enables us to apply final downstream analysis of any sort with ease, 2) scale-free expression enhances sensitivity of differential expression when the overall variation of expression is relatively ignorable on its raw scale.

References

- Choi H, Shen R, Chinnaiyan A, Ghosh D. Latent variable modelling for combining genomic data from multiple studies *Unpublished manuscript* (2005).
- Lusa L, Gentleman R, Ruschhaupt M. *GeneMeta* A collection of meta-analysis tools for analysing high throughput experimental data *A bioconductor package*
- Choi J.K, Yu U, Kim S, Yoo O.J. Combining multiple microarray studies and modeling interstudy variation *Bioinformatics* 19:84–90 (2003).
- Parmigiani G, Garrett E.S, Anbazhagan R., Gabrielson E. A statistical framework for expression-based molecular classification in cancer. *JRSS B* 64, 717 – 736 (2002).
- Parmigiani G, Garrett-Mayer E.S, Anbazhagan R, Gabrielson E. A cross-study comparison of gene expression studies for the molecular classification of lung cancer. *Clinical Cancer Research* 10: 2922–2927 (2004).
- Wang J, Coombes K.R, Highsmith W.E, Keating M.J, Abruzzo L.V. Differences in gene expression between B-cell chronic lymphocytic leukemia and normal B cells: a meta-analysis of three micorarray studies. *Bioinformatics* 20(17): 3166–3178 (2004).
- Chen X et al. Gene expression patterns in human liver cancers. *Mol. Biol. Cell* 13(6): 1929–1939

Garber M et al. Diversity of gene expression in adenocarcinoma of the lung *PNAS* 98(24): 13784–13789

Lapointe J et al. Gene expression profiling identifies clinically relevant subtypes of prostate cancer *PNAS* 101(3): 811–816