Test association between phenotype and gene expression

Evarist Planet
Bioinformatics & Biostatistics Unit
IRB Barcelona

Contents

1 Introduction 1

2 Individual gene(s) association with phenotype(s) 2
   2.1 Creating an epheno .......................... 2
   2.2 Useful methods for the epheno object ........ 4
   2.3 Export an epheno ............................. 7

3 Gene set(s) association with phenotype(s) 7
   3.1 Plots that use epheno as input ................ 8
   3.2 GSEA (Gene Set Enrichment Analysis) ....... 10

1 Introduction

Imagine a situation where we have gene expression and phenotype variables and we want to test the association of each gene with phenotype. We would probably be interested in testing association of groups of genes (or gene sets) with phenotype. This library provides the tools to do both things in a way that is efficient, structured, fast and scalable. We also provide tools to do GSEA (Gene set enrichment analysis) of all phenotype variables at once.

The functions and methods presented on this vignette provide tools to easily test association between gene expression levels of individual genes or gene sets of genes and the selected phenotypes of a given gene expression dataset. These can be particularly useful for datasets arising from RNAseq or microarray gene expression studies.

We will load the ExpressionSet of a cohort (GSE2034) we downloaded from GEO.
For illustration purposes we selected the first 1000 genes and the first 100 samples, created a continuous variable (named Tumor.size) and added a new category to the categorical variable lymph.node.status to illustrate the functionality of the package.

```r
> Tumor.size <- rnorm(ncol(eset),50,2)
> pData(eset) <- cbind(pData(eset),Tumor.size)
> pData(eset)[1:20,'lymph.node.status'] <- 'positive'
```

2 Individual gene(s) association with phenotype(s)

2.1 Creating an epheno

The epheno object will contain the univariate association between a list of phenotype variables and the gene expression from the given ExpressionSet. We will use the ExpressionPhenoTest function to create the epheno object. We will have to tell this function which phenotype variables we want to test and the type of these variables (if they are ordinal, continuous, categorical or survival variables). For this purpose we will create a variable called (for instance) vars2test. This variable has to be of class list with components continuous, categorical, ordinal and survival indicating which phenotype variables should be tested. continuous, categorical and ordinal must be character vectors, survival a matrix with columns named time and event. The names must match names in names(pData(eset)) (being eset the ExpressionSet of the cohort we are interested in).

```r
> head(pData(eset))

<table>
<thead>
<tr>
<th>PID</th>
<th>GEOaccession</th>
<th>lymph.node.status</th>
<th>Months2Relapse</th>
<th>Relapse</th>
<th>ER.Status</th>
<th>BrainRelapse</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSM36793</td>
<td>GSM36793</td>
<td>positive</td>
<td>101</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GSM36796</td>
<td>GSM36796</td>
<td>positive</td>
<td>118</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>GSM36797</td>
<td>GSM36797</td>
<td>positive</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```
> survival <- matrix(c("Relapse", "Months2Relapse"), ncol=2, byrow=TRUE)
> colnames(survival) <- c("event", "time")
> vars2test <- list(survival=survival, categorical="lymph.node.status",
continuous="Tumor.size")
> vars2test

$survival
   event      time
[1,] "Relapse" "Months2Relapse"

$categorical
[1] "lymph.node.status"

$continuous
[1] "Tumor.size"

Now we have everything we need to create the epheno object:

> epheno <- ExpressionPhenoTest(eset, vars2test, p.adjust.method='none')

Performing analysis for continuous variable Tumor.size
Performing analysis for categorical variable lymph.node.status
Performing analysis for survival variable Relapse

> epheno

Object of class 'epheno'
featureNames: 1007_s_at, 1053_at, 117_at ... (1000) feature(s)
phenoNames: Tumor.size, lymph.node.status, Relapse. (3) phenotype(s)
P-value adjustment method: none
Annotation: hgu133a
Approach: frequentist

Type "showMethods(classes='epheno')" for a list of ALL methods

P values can also be adjusted afterwards:

> p.adjust.method(epheno)

[1] "none"

> epheno <- pAdjust(epheno, method='BH')
> p.adjust.method(epheno)

[1] "BH"
The epheno object extends the ExpressionSet object and therefore methods that are available for ExpressionSets are also available for ephenos.

The effect of both continuous, categorical and ordinal phenotype variables on gene expression levels are tested via lmFit from package limma (Smyth [2005]). For ordinal variables a single coefficient is used to test its effect on gene expression (trend test), which is then used to obtain a P-value. Gene expression effects on survival are tested via Cox proportional hazards model (Cox [1972]), as implemented in function coxph from package survival.

If we want we can compute posterior probabilities instead of p-values we can set the argument approach = "bayesian". The default value is "frequentist".

ExpressionPhenoTest implements parallel computing via the function mclapply from the package multicore. Currently multicore only operates on Unix systems. If you are a windows user you should set mc.cores = 1 (the default).

2.2 Useful methods for the epheno object

Some of the methods for the epheno objects are shown here.

The object can be subsetted by phenotype names:

```r
> phenoNames(epheno)
[1] "Tumor.size"  "lymph.node.status" "Relapse"
```

```r
> epheno[, 'Tumor.size']
```

Object of class 'epheno'
featureNames: 1007_s_at, 1053_at, 117_at ... (1000) feature(s)
phenoNames: Tumor.size. (1) phenotype(s)
P-value adjustment method: BH
Annotation: hgu133a
Approach: frequentist

Type "showMethods(classes='epheno')" for a list of ALL methods

```r
> epheno[,2]
```

Object of class 'epheno'
featureNames: 1007_s_at, 1053_at, 117_at ... (1000) feature(s)
phenoNames: Relapse. (1) phenotype(s)
P-value adjustment method: BH
Annotation: hgu133a
Approach: frequentist

Type "showMethods(classes='epheno')" for a list of ALL methods

or by class (class can be ordinal, continuous, categorical or survival):

```r
> phenoClass(epheno)
```
epheno objects contain information summarizing the association between genes and phenotypes. getMeans can be used to obtain the average expression for each group in categorical and ordinal variables, as well as for categorized version of the continuous variables.

> head(getMeans(epheno))

<table>
<thead>
<tr>
<th>Tumor.size, [45.0, 49.1)</th>
<th>Tumor.size, [49.1, 51.1)</th>
<th>Tumor.size, [51.1, 57.2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1007_s_at</td>
<td>11.768121</td>
<td>11.898366</td>
</tr>
<tr>
<td>1053_at</td>
<td>7.656931</td>
<td>7.655312</td>
</tr>
<tr>
<td>117_at</td>
<td>7.799664</td>
<td>7.846323</td>
</tr>
<tr>
<td>121_at</td>
<td>10.196792</td>
<td>10.270059</td>
</tr>
<tr>
<td>1255_g_at</td>
<td>5.601724</td>
<td>5.905409</td>
</tr>
<tr>
<td>1294_at</td>
<td>9.374232</td>
<td>9.453646</td>
</tr>
</tbody>
</table>

Here we see that tumor size has been categorized into 3 groups. The number of categories can be changed with the argument continuousCategories in the call to ExpressionPhenoTest.

epheno objects also contain fold changes and hazard ratios (for survival variables). These can be accessed with getSummaryDif, getFc and getHr.

> head(getSummaryDif(epheno))

<table>
<thead>
<tr>
<th>Tumor.size, fc, [45.0, 49.1)</th>
<th>Tumor.size, fc, [49.1, 51.1)</th>
<th>lymph.node.status, positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>1007_s_at</td>
<td>1.094480</td>
<td>-1.029039</td>
</tr>
<tr>
<td>1053_at</td>
<td>-1.001123</td>
<td>1.193289</td>
</tr>
<tr>
<td>117_at</td>
<td>1.032949</td>
<td>1.192570</td>
</tr>
<tr>
<td>121_at</td>
<td>1.052096</td>
<td>1.026478</td>
</tr>
<tr>
<td>1255_g_at</td>
<td>1.234293</td>
<td>1.516840</td>
</tr>
<tr>
<td>1294_at</td>
<td>1.056589</td>
<td>1.099222</td>
</tr>
</tbody>
</table>

Relapse, HR

| 1007_s_at | -1.290929 |
| 1053_at | -1.138198 |
| 117_at | 1.175962 |
| 121_at | 1.000616 |
| 1255_g_at | 1.063210 |
| 1294_at | 1.000612 |
ExpressionPhenoTest also computes P-values. eBayes from limma package is used for continuous, categorical and ordinal phenotypes. A Cox proportional hazards likelihood-ratio test is used for survival phenotypes. P-values can be accessed with getSignif. Notice that a single P-value is reported for each phenotype variable. For categorical variables these corresponds to the overall null hypothesis that there are no differences between groups.

We can also ask for the variables we sent to the ExpressionPhenoTest function:

```r
> getVars2test(epheno)

$continuous
[1] "Tumor.size"

$categorical
[1] "lymph.node.status"

$survival
  event time
[1] "Relapse" "Months2Relapse"
```
2.3 Export an epheno

Functions `export2csv` and `epheno2html` can be used to export to a comma separated value (csv) or an html file. The html file will have useful links to online databases that will provide information about each known gene. For more information about how to use these functions and examples read their help manuals.

3 Gene set(s) association with phenotype(s)

Gene sets can be stored in a list object. Each element of the list will contain one gene set. The names of the list will be the names of the gene sets. Here we select genes at random to build our gene sets:

```r
> set.seed(777)
> sign1 <- sample(featureNames(eset))[1:20]
> sign2 <- sample(featureNames(eset))[1:50]
> mySignature <- list(sign1,sign2)
> names(mySignature) <- c('My first signature', 'Another signature')
> mySignature

$My first signature
 [1] "200003_s_at" "200985_s_at" "200069_at" "201172_s_at" "200862_s_at" "201174_s_at"
 [7] "200062_s_at" "1487_at" "201250_s_at" "201444_s_at" "201393_s_at" "200737_at"
[13] "200616_s_at" "200047_s_at" "200924_s_at" "201138_s_at" "201263_at" "201006_at"
[19] "201037_at" "201463_s_at"

$Another signature
 [1] "200745_s_at" "200970_s_at" "200066_at" "201417_at" "200733_s_at" "200844_s_at"
 [7] "201224_s_at" "201100_s_at" "201422_at" "201005_at" "201445_at" "201345_at"
[13] "200617_at" "200030_s_at" "200988_at" "201437_s_at" "201005_s_at" "200791_s_at"
[19] "201040_at" "200084_s_at" "200029_x_at" "200653_s_at" "201123_s_at" "200089_s_at"
[25] "201102_s_at" "201165_s_at" "200034_s_at" "200368_s_at" "201414_s_at"
[31] "201332_s_at" "201313_at" "200062_s_at" "201006_at" "201109_s_at" "201076_at"
[37] "200022_at" "200762_at" "201396_s_at" "200995_s_at" "200695_s_at" "200895_s_at"
[43] "200647_x_at" "201471_s_at" "201223_s_at" "1598_g_at" "201040_at" "201061_s_at"
[49] "201394_at" "200072_s_at"
```

Gene sets can also be stored in gene set collection objects. From here on all functions have methods for gene sets stored as lists, `GeneSets` or `GeneSetCollections`. You can use the one you feel more comfortable with. We will work with `GeneSetCollection`:

```r
> library(GSEABase)
> myGeneSetA <- GeneSet(geneIds=sign1, setName='My first signature')
> myGeneSetB <- GeneSet(geneIds=sign2, setName='Another signature')
> mySignature <- GeneSetCollection(myGeneSetA,myGeneSetB)
> mySignature

GeneSetCollection
 names: My first signature, Another signature (2 total)
 unique identifiers: 200003_s_at, 200985_s_at, ... 200672_x_at (69 total)
types in collection:
 geneIdType: NullIdentifier (1 total)
 collectionType: NullCollection (1 total)
```
Figure 1: barplotSignifSignatures: Number of differentially expressed genes in each gene set that are statistically significant. P-values test for differences in each signature between the number of up and down regulated genes.

3.1 Plots that use epheno as input

barplotSignifSignatures will plot the percentage of up regulated and down regulated genes that are statistically significant in each signature. In our random selection of genes we did not find any statistically significant genes. Therefore, and just to show the plot we set the alpha value 0.99. The plot can be seen in Figure 1.

> barplotSignifSignatures(epheno[, 'lymph.node.status'], mySignature, alpha=0.99)

By default barplotSignifSignatures performs a binomial test (binom.test from package stats) for each signature to test if the proportions of up regulated and down regulated genes are different. For example, Figure 1 indicates that in the first signature the proportion of up regulated genes is higher than the proportion of down regulated genes. The second signature shows no significant statistical differences.

Sometimes we want to compare the proportions of up and down regulated genes in our signature with the proportions of up and down regulated of all genes in the genome. In this case we may provide a reference signature via the
Figure 2: barplotSignatures: Average fold change or hazard ratio.

When providing the referenceSignature argument a chi-square test comparing the proportion of up and down regulated genes in each signature with the proportion in the reference set will be computed.

When a reference gene set is provided and parameter testUpDown is TRUE (by default it is FALSE) the proportion of up regulated genes is compared with those of the reference gene set. The same is done for down regulated genes.

barplotSignatures plots the average log2 fold change or hazard ratio of each phenotype for each gene set. Figure 2 shows an example of it.

> barplotSignatures(epheno[, 'Tumor.size'], mySignature, ylim=c(0,1))

We can also cluster our samples in two clusters based on the expression levels of one gene set of genes and then test the effect of cluster on phenotypes. For ordinal and continuous variables a Kruskal-Wallis Rank Sum test is used, for categorical variables a chi-square test is used and for survival variables a Cox proportional hazards likelihood-ratio test is used. The heatmapPhenoTest function can be used to this end. Its results can be seen in Figure 3 and 4.

> pvals <- heatmapPhenoTest(eset, mySignature[[1]], vars2test=vars2test[[1]], heat.kaplan='heat')
> pvals
Figure 3: Heatmap produced with heatmapPhenoTest function. All variables in vars2test that are of class logical will be plotted under the heatmap.

```r
> pvals <- heatmapPhenoTest(eset, mySignature[1], vars2test=vars2test[1], heat.kaplan='kaplan')
> pvals

Months2Relapse
"(P=0.1247)"
```

### 3.2 GSEA (Gene Set Enrichment Analysis)

A popular way to test association between gene sets’ gene expression and phenotype is GSEA (Subramanian [2005]). The main idea is to test the association between the gene set as a whole and a phenotype.

Although GSEA and several extensions are already available in other Biconductor packages, here we implement a slightly different extension. Most GSEA-like approaches assess statistical significance by permuting the values of the phenotype of interest. From a statisticall point of view this tests the null hypothesis that no genes are associated with phenotype. However in
Figure 4: Kaplan-Meier produced with heatmapPhenoTest function.

Many applications one is actually interested in testing if the proportion of genes associated with phenotype in the gene set is greater than that outside of the gene set. As a simple example, imagine a cancer study where 25% of the genes are differentially expressed. In this setup a randomly chosen gene set will have around 25% of differentially expressed genes, and classical GSEA-like approaches will tend to flag the gene set as statistically significant. In contrast, our implementation will tend to select only gene sets with more than 25% of differentially expressed genes.

We will use the gsea method to compute enrichment scores (see Subramanian [2005] for details about the enrichment scores) and simulated enrichment scores (by permuting the selection of genes). The simulated enrichment scores are used to compute P-values and FDR. We can summarize the results obtained using the summary method. The following chunk of code is an illustrative example of it:

```r
> my.gsea <- gsea(x=epheno,gsets=mySignature,B=1000,p.adjust='BH')
```

2 gene set(s) were provided and 1000 permutations were assigned, therefore 500 permutations will be computed on each gene set. 2 gene set(s) were provided and 1000 permutations were assigned, therefore 500 permutations will be computed on each gene set. 2 gene set(s) were provided and 1000 permutations were assigned, therefore 500 permutations will be computed on each gene set.
2 gene set(s) were provided and 1000 permutations were assigned, therefore 500 permutations will be computed on each gene set.

```r
> my.gsea
```

Object of class 'gseaData'
You can use the summary method to produce result summaries. You can use the getEs, getNes, getEsSim and getFchhr methods to easily access its data.

Gam approximation was not used.

The tested variables are:
- Tumor.size.fc.[49.1,51.1)
- Tumor.size.fc.[51.1,57.2]
- lymph.node.status.positive.fc
- Relapse.HR

The tested gene sets (for each variable) are:
- My first signature
- Another signature

```r
> summary.gseaData(my.gsea)
```

<table>
<thead>
<tr>
<th>variable</th>
<th>geneSet</th>
<th>n</th>
<th>es</th>
<th>nes</th>
<th>pval.es</th>
<th>pval.nes</th>
<th>fdr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tumor.size.fc.[49.1,51.1)</td>
<td>My first signature</td>
<td>20</td>
<td>0.218453</td>
<td>0.658050</td>
<td>0.914598</td>
<td>0.942526</td>
<td></td>
</tr>
<tr>
<td>Tumor.size.fc.[49.1,51.1)</td>
<td>Another signature</td>
<td>50</td>
<td>-0.232116</td>
<td>-0.844673</td>
<td>0.914598</td>
<td>0.942526</td>
<td></td>
</tr>
<tr>
<td>Tumor.size.fc.[51.1,57.2]</td>
<td>My first signature</td>
<td>20</td>
<td>0.461815</td>
<td>1.299005</td>
<td>0.288004</td>
<td>0.239398</td>
<td></td>
</tr>
<tr>
<td>Tumor.size.fc.[51.1,57.2]</td>
<td>Another signature</td>
<td>50</td>
<td>0.214914</td>
<td>0.750099</td>
<td>0.288004</td>
<td>0.239398</td>
<td></td>
</tr>
<tr>
<td>lymph.node.status.positive.fc</td>
<td>My first signature</td>
<td>20</td>
<td>-0.413846</td>
<td>-1.177994</td>
<td>0.272506</td>
<td>0.270106</td>
<td></td>
</tr>
<tr>
<td>lymph.node.status.positive.fc</td>
<td>Another signature</td>
<td>50</td>
<td>0.319069</td>
<td>1.344498</td>
<td>0.272506</td>
<td>0.270106</td>
<td></td>
</tr>
<tr>
<td>Relapse.HR</td>
<td>My first signature</td>
<td>20</td>
<td>-0.363640</td>
<td>-1.120173</td>
<td>0.272506</td>
<td>0.270106</td>
<td></td>
</tr>
<tr>
<td>Relapse.HR</td>
<td>Another signature</td>
<td>50</td>
<td>-0.264523</td>
<td>-1.012109</td>
<td>0.272506</td>
<td>0.270106</td>
<td></td>
</tr>
</tbody>
</table>

We receive one message for each phenotype we are testing.

We can produce plots as follows:

```r
> plot.gseaData(my.gsea)
```

This will produce two plots (one for enrichment score and another for normalised enrichment score) for every phenotype and gene set (in our case 12 plots). Following code shows an example on plotting only enrichment score for variable Relapse on the first gene set of genes. Plot can be seen in Figure 5.

```r
> my.gsea <- gsea(x=epheno[, 'Relapse'], gsets=mySignature[1], B=100, p.adjust='BH')
```

1 gene set(s) were provided and 100 permutations were assigned, therefore 100 permutations will be computed on each gene set.

```r
> summary.gseaData(my.gsea)
```

<table>
<thead>
<tr>
<th>variable</th>
<th>geneSet</th>
<th>n</th>
<th>es</th>
<th>nes</th>
<th>pval.es</th>
<th>pval.nes</th>
<th>fdr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relapse.HR</td>
<td>My first signature</td>
<td>20</td>
<td>-0.36365</td>
<td>-1.15641</td>
<td>0.297797</td>
<td>0.297009</td>
<td>0.297009</td>
</tr>
</tbody>
</table>
Figure 5: GSEA plot.
gsea can be used not only with epheno objects but also with objects of class numeric or matrix. For more information read the gsea function help.

Following similar ideas to [Virtanen 2001] we also implemented a Wilcoxon test. This can be used instead of the permutation test which can be slow if we use a lot of permutations and we can not use the multicore package. The plot we will obtain will also be different. Instead of plotting the enrichment scores we will plot the density function and the mean log2 fold change or hazard ratio of the genes that belong to our gene set. This will allow us to compare how similar/different from 0 the mean of our gene set is. The plot using Wilcoxon test can be seen in Figure 6.

> my.gsea <- gsea(x=epheno[, 'Relapse'], gsets=mySignature, B=100, test='wilcox', p.adjust='BH')

2 gene set(s) were provided and 100 permutations were assigned, therefore 50 permutations will be computed on each gene set.

> summary.gseaData(my.gsea)

<table>
<thead>
<tr>
<th>variable</th>
<th>geneSet</th>
<th>n</th>
<th>es</th>
<th>pval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relapse.HR</td>
<td>My first signature</td>
<td>20</td>
<td>-0.03030989</td>
<td>0.6235886</td>
</tr>
<tr>
<td>Relapse.HR</td>
<td>Another signature</td>
<td>50</td>
<td>-0.01083301</td>
<td>0.9691988</td>
</tr>
</tbody>
</table>

> plot.gseaData(my.gsea, selGsets='My first signature')

Notice that using a Wilcoxon test is conceptually very similar to the average gene set fold change presented in figure 2.

A current limitation of gseaSignatures is that it does not consider the existence of dependence between genes in the gene set. This will be addressed in future versions. Nevertheless we believe gseaSignatures is useful in that it targets the correct null hypothesis that gene set is as enriched as a randomly selected gene set, opposed to testing that there are no enriched genes in the set as is done in GSEA.

References


Figure 6: GSEA plot using Wilcoxon test.