Test association between phenotype and gene expression

Evarist Planet
Bioinformatics & Biostatistics Unit
IRB Barcelona

Contents

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.

```
> options(width=100)
> library(phenoTest)
> data(eset)
> eset

ExpressionSet (storageMode: lockedEnvironment)
assayData: 1000 features, 100 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: GSM36793 GSM36796 ... GSM36924 (100 total)
  varLabels: PID GEOaccession ... BrainRelapse (7 total)
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
Annotation: hgu133a
```

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.

```
> 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).

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

```
         PID GEOaccession lymph.node.status Months2Relapse Relapse ER.Status BrainRelapse
GSM36793   3     GSM36793          positive            101       0         0            0
GSM36796   5     GSM36796          positive            118       0         1            0
GSM36797   6     GSM36797          positive              9       1         0            0
GSM36798   7     GSM36798          positive            106       0         0            0
GSM36800   8     GSM36800          positive             37       1         0            0
GSM36801   9     GSM36801          positive            125       0         1            0
         Tumor.size
GSM36793   52.17096
GSM36796   48.82104
GSM36797   49.52611
GSM36798   52.35670
GSM36800   48.17986
GSM36801   49.67234
```

```
> 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 `ExpressionSet`s 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 pvalues 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 subseted by phenotype names:

```
> phenoNames(epheno)

[1] "Tumor.size"        "lymph.node.status" "Relapse"

> 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

> epheno[,2]

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
```

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

```
> phenoClass(epheno)

       Tumor.size lymph.node.status           Relapse
     "continuous"     "categorical"        "survival"

> epheno[,phenoClass(epheno)=='survival']

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
```

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))

          Tumor.size.[45.1,49.0) Tumor.size.[49.0,50.6) Tumor.size.[50.6,55.8]
1007_s_at              11.894590              11.774770              11.720118
1053_at                 7.727530               7.699912               7.794537
117_at                  7.968976               7.898973               7.826422
121_at                 10.210502              10.243247              10.247182
1255_g_at               6.060163               5.930123               5.705748
1294_at                 9.487542               9.504705               9.342913
          lymph.node.status.negative lymph.node.status.positive
1007_s_at                  11.779693                  11.868594
1053_at                     7.700153                   7.902028
117_at                      7.882203                   7.965347
121_at                     10.217469                  10.297187
1255_g_at                   5.856978                   6.073553
1294_at                     9.444534                   9.449257
```

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))

          Tumor.size.fc.[49.0,50.6) Tumor.size.fc.[50.6,55.8] lymph.node.status.positive.fc
1007_s_at                 -1.086599                 -1.128551                       1.063559
1053_at                   -1.019328                  1.047541                       1.150192
117_at                    -1.049718                 -1.103858                       1.059324
121_at                     1.022956                  1.025751                       1.056812
1255_g_at                 -1.094324                 -1.278468                       1.161972
1294_at                    1.011967                 -1.105447                       1.003280
          Relapse.HR
1007_s_at -1.299526
1053_at   -1.138198
117_at     1.175962
121_at     1.100616
1255_g_at  1.093210
1294_at    1.006512
```

```
> head(getFc(epheno))
```

```
          Tumor.size.fc.[49.0,50.6) Tumor.size.fc.[50.6,55.8] lymph.node.status.positive.fc
1007_s_at                -1.086599                -1.128551                       1.063559
1053_at                  -1.019328                 1.047541                       1.150192
117_at                   -1.049718                -1.103858                       1.059324
121_at                    1.022956                 1.025751                       1.056812
1255_g_at                -1.094324                -1.278468                       1.161972
1294_at                   1.011967                -1.105447                       1.003280
```

```
> head(getHr(epheno))
```

```
          Relapse.HR
1007_s_at  -1.299526
1053_at    -1.138198
117_at      1.175962
121_at      1.100616
1255_g_at   1.093210
1294_at     1.006512
```

`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.

```
> head(getSignif(epheno))
```

```
          Tumor.size lymph.node.status.positive.pval   Relapse
1007_s_at  0.9307513                       0.9022639 0.9801106
1053_at    0.9307513                       0.7988976 0.9801106
117_at     0.9307513                       0.9225719 0.9801106
121_at     0.9868056                       0.8689036 0.9803989
1255_g_at  0.9307513                       0.8254344 0.9803989
1294_at    0.9868056                       0.9965107 0.9907749
```

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

```
> 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:

```
> 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_x_at" "200982_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" "201402_at"   "201005_at"   "201445_at"   "201348_at"
[13] "200617_at"   "200030_s_at" "200661_at"   "200688_at"   "201437_s_at" "200791_s_at"
[19] "201040_at"   "200984_s_at" "200829_x_at" "200653_s_at" "201123_s_at" "200089_s_at"
[25] "201102_s_at" "201165_s_at" "200752_s_at" "200834_s_at" "200036_s_at" "201414_s_at"
[31] "201332_s_at" "201313_at"   "200082_s_at" "201006_at"   "201109_s_at" "201076_at"
[37] "200022_at"   "200762_at"   "201396_s_at" "200995_at"   "200959_at"   "200898_s_at"
[43] "200647_x_at" "201471_s_at" "201223_s_at" "1598_g_at"   "201061_s_at" "201130_s_at"
[49] "201344_at"   "200672_x_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 confortable with. We will work with `GeneSetCollection`:

```
> 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)
```

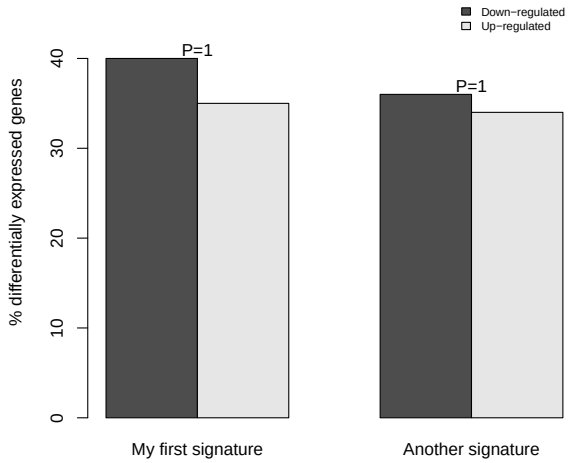

Figure 1: `barplotSignifSignatures`: Number of diferentially expressed genes in each gene set that are statistically significant. P-values test for differences in each signature between the number of up and down regulated genes.

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

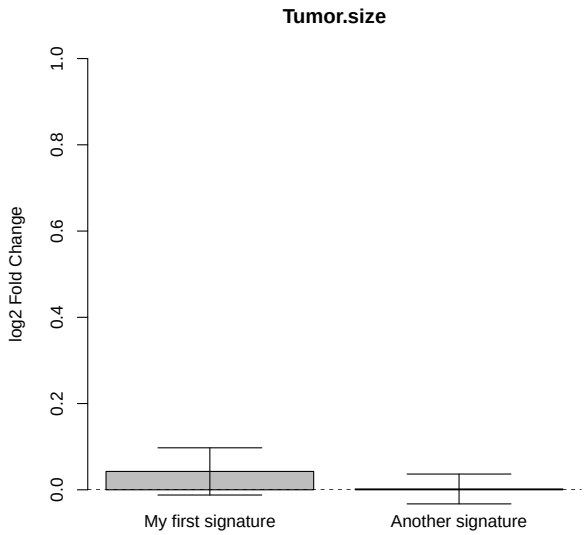

Figure 2: `barplotSignatures`: Averge fold change or hazard ratio.

the argument `referenceSignature`. 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')
```

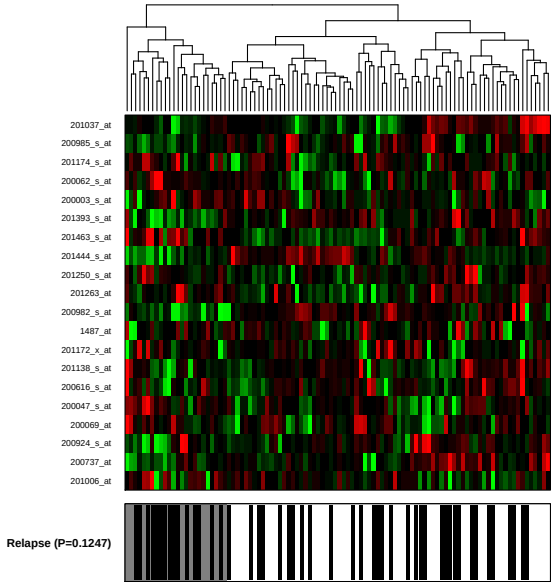

Figure 3: Heatmap produced with `heatmapPhenoTest` function. All variables in `vars2test` that are of class *logical* will be plotted under the heatmap.

```
> pvals

Months2Relapse
  "(P=0.1247)"


> 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

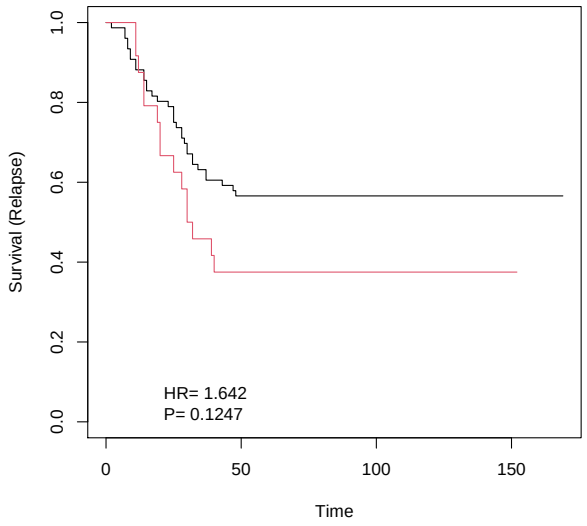

Figure 4: Kaplan-Meier produced with `heatmapPhenoTest` function.

null hypothesis that no genes are associated with phenotype. However in 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:

```
> 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.
```

```
> my.gsea
```

```
Object of class 'gseaData'
You can use the summary method to produce result summaries.
You can use the getEs, getNes, getEsSim and getFcHr methods to easily acces its data.
Gam approximation was not used.
The tested variables are:
  Tumor.size.fc.[49.0,50.6), Tumor.size.fc.[50.6,55.8], lymph.node.status.positive.fc, Relapse.HR
The tested gene sets (for each variable) are:
 My first signature, Another signature
```

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

```
                       variable            geneSet  n         es        nes   pval.es  pval.nes
1     Tumor.size.fc.[49.0,50.6) My first signature 20  0.3051632  0.9704279 0.4884673 0.4904405
2     Tumor.size.fc.[49.0,50.6)  Another signature 50 -0.2874827 -1.0767932 0.4884673 0.4904405
3     Tumor.size.fc.[50.6,55.8] My first signature 20  0.4017870  1.1715048 0.2846238 0.2944666
4     Tumor.size.fc.[50.6,55.8]  Another signature 50  0.3147134  1.1251299 0.2846238 0.2944666
5 lymph.node.status.positive.fc My first signature 20 -0.4138455 -1.1779948 0.2725070 0.2701077
6 lymph.node.status.positive.fc  Another signature 50  0.3190696  1.1344498 0.2725070 0.2701077
7                    Relapse.HR My first signature 20 -0.3636399 -1.1201729 0.4508479 0.4530801
8                    Relapse.HR  Another signature 50 -0.2645231 -1.0121094 0.4508479 0.4530801
        fdr
1 0.4904405
2 0.3500326
3 0.4778108
4 0.2944666
5 0.2390689
6 0.2701077
7 0.6147916
8 0.4530801
```

We receive one message for each phenotype we are testing.
We can produce plots as follows:

```
> 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.

```
> 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.
```

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

```
    variable            geneSet  n         es       nes   pval.es pval.nes      fdr
1 Relapse.HR My first signature 20 -0.3636399 -1.155416 0.2977998 0.297801 0.297801
```

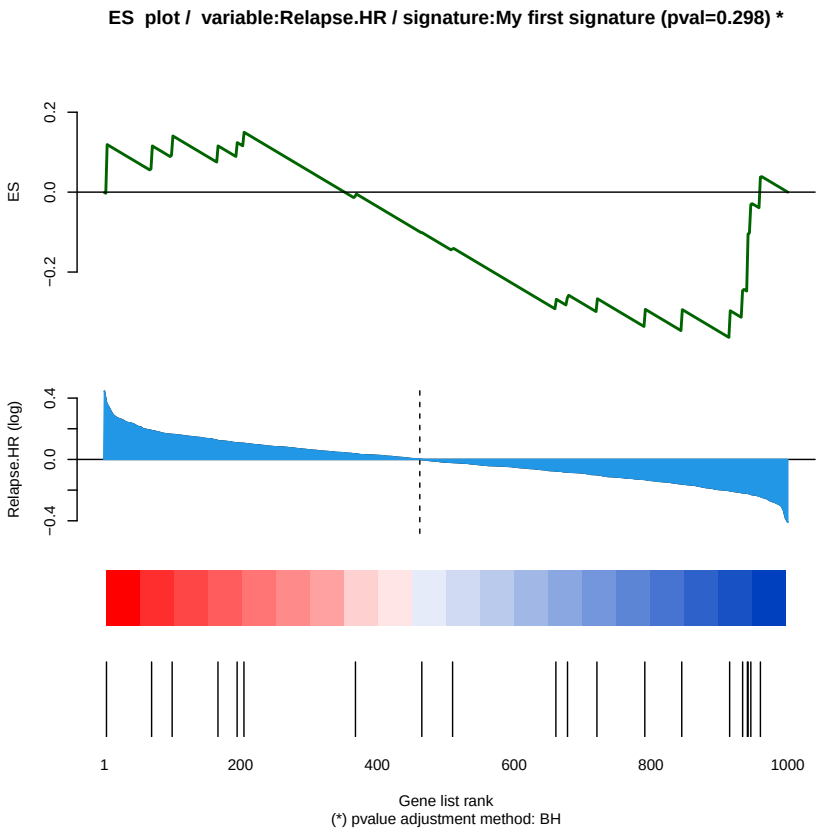

Figure 5: GSEA plot.

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

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 Virtaneva [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)

    variable            geneSet  n          es      pval
1 Relapse.HR My first signature 20 -0.03030989 0.6235886
2 Relapse.HR  Another signature 50 -0.01083301 0.9691988

> 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 existance of dependence between genes in the gene set. This will be addressed in future versions. Nevertheless we believe gseaSignatures is usefull 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

D.R. Cox. Regression models and life tables. *Journal of the Royal Statistical Society Series B*, 34:187–220, 1972.

G.K. Smyth. Limma: linear models for microarray data. In R. Gentleman, V. Carey, S. Dudoit, R. Irizarry, and W. Huber, editors, *Bioinformatics and Computational Biology Solutions using R and Bioconductor*, pages 397–420. Springer, New York, 2005.

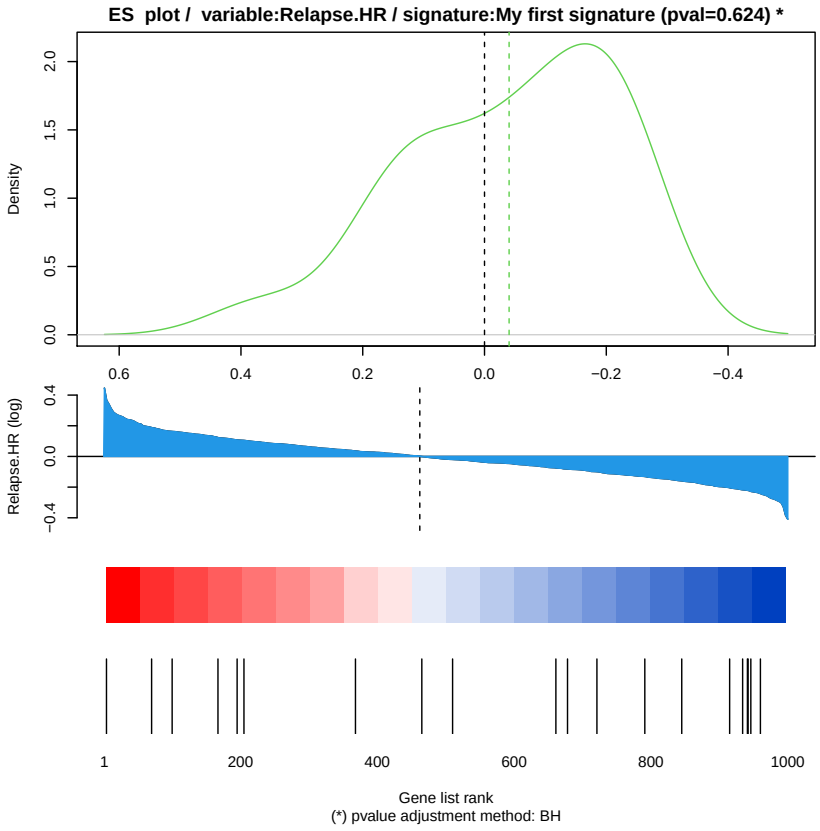

Figure 6: GSEA plot using Wilcoxon test.

Aravind Subramanian. Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles. *PNAS*, 102, 2005.

K. Virtaneva. Expression profiling reveals fundamental biological differences in acute myeloid leukemia with isolated trisomy 8 and normal cytogenetics. *Proc Natl Acad Sci U S A*, 98(98):1124–1129, January 2001. doi: http://dx.doi.org/10.1073/pnas.98.3.1124. URL `http://dx.doi.org/10.1073/pnas.98.3.1124`.