How To Filter Genes

November 25, 2003

Introduction

The *genefilter* package can be used to filter (select) genes from a microarray experiment according to a wide variety of different filtering mechanisms. To be concrete we will consider the artificial experiment contained in the `eset` example in the *Biobase* package. This experiment has 26 samples, there are 500 genes and 3 covariates. The covariates are named `cov1`, `cov2` and `cov3`. The first two have two levels and the last one has three.

```
> library("genefilter")

Loading required package: Biobase
Welcome to Bioconductor
         Vignettes contain introductory material.  To view,
         simply type: openVignette()
         For details on reading vignettes, see
         the openVignette help page.

> data(eset)
> table(eset$cov1)

 1  2
13 13

> table(eset$cov2)

 1  2
11 15

> table(eset$cov3)

1 2 3
8 9 9
```

One dichotomy that is of interest is to split filters into the non–specific and the specific. Here, specific means that we are filtering with reference to some particular variable. For example if we want to select genes that are differentially expressed in the two groups defined by `cov1` that is a specific filter. If on the otherhand we want to select genes that are expressed in more than 5 samples then that is an example of a non–specific filter.

First let's see how to perform a non–specific filter. Suppose we want to select genes that are expressed above 200 in at least 5 samples. To do that we use the function `kOverA`.

There are three steps that must be performed.

1. Create the filters function(s).
2. Assemble them into a filtering function.
3. Apply the filtering function to the expression matrix.

```
> f1 <- kOverA(5, 200)
> ffun <- filterfun(f1)
> which <- genefilter(exprs(eset), ffun)
> sum(which)

[1] 154
```

Here `f1` is a filter, the function `ffun` is a filtering function and we finally apply it using `genefilter`. There were 154 genes that had a level of more than 200 in at least 5 samples.

To perform a specific filter lets select genes that are differentially expressed in the groups defined by `cov2`.

```
> tf1 <- ttest(eset$cov2, 0.1)
> ff2 <- filterfun(tf1)
> wh2 <- genefilter(exprs(eset), ff2)
> sum(wh2)

[1] 31
```

Now we see that there are 31 genes that pass this selection procedure. In this example we chose a very large p–value because we have a very small sample and the covariate was made up.

Suppose that we want to combine the two filters. We want those genes for which at least 5 are over 200 and which also are differentially expressed in the groups defined by `cov2`.

```
> ff3 <- filterfun(f1, tf1)
> wh3 <- genefilter(exprs(eset), ff3)
> sum(wh3)

[1] 10
```

Now we see that there are only 10 genes that satisfy both conditions.

Our last example is to select genes that are differentially expressed in at least one of the three groups defined by `cov3`. To do that we use an Anova filter. This filter uses an analysis of variance appraoch (via the `lm`) function to test the hypothesis that at least one of the three group means is different from the other two. The test is applied, then the p–value computed. We select those genes that have a low p–value.

```
> Afilter <- Anova(eset$cov3)
> aff <- filterfun(Afilter)
> wh4 <- genefilter(exprs(eset), aff)
> sum(wh4)

[1] 14
```

We see that there are 14 genes that pass this filter and that are candidates for further exploration.

1 Aggregate Vignette

The function given below performs k–nearest neighbour classification using leave–one–out cross–validation. At the same time it aggregates the genes that were selected. The function returns the predicted classifications as its returned value. However, there is an additional side effect. The number of times that each gene was used (provided it was at least one) are recorded and stored in the environment of the aggregator `Agg`. These can subsequently be retrieved and used for other purposes.

```
> knnCV <- function(EXPR, selectfun, cov, Agg, pselect = 0.01,
+     Scale = FALSE) {
+     nc <- ncol(EXPR)
+     outvals <- rep(NA, nc)
+     for (i in 1:nc) {
+         v1 <- EXPR[, i]
+         expr <- EXPR[, -i]
+         glist <- selectfun(expr, cov[-i], p = pselect)
+         expr <- expr[glist, ]
+         if (Scale) {
+             expr <- scale(expr)
+             v1 <- as.vector(scale(v1[glist]))
+         }
+         else v1 <- v1[glist]
+         out <- paste("iter ", i, " num genes= ", sum(glist),
+             sep = "")
+         print(out)
+         Aggregate(row.names(expr), Agg)
+         if (length(v1) == 1)
+             outvals[i] <- knn(expr, v1, cov[-i], k = 5)
+         else outvals[i] <- knn(t(expr), v1, cov[-i], k = 5)
+     }
+     return(outvals)
+ }

> gfun <- function(expr, cov, p = 0.05) {
+     f2 <- ttest(cov, p = p)
+     ffun <- filterfun(f2)
+     which <- genefilter(expr, ffun)
+ }
```

Next we show how to use this function on the system dataset, `geneData`.

```
> library("class")
> geneData <- genescale(exprs(eset)[1:75, ], 1)
> Agg <- new("aggregator")
> testcase <- knnCV(geneData, gfun, eset$cov2, Agg, pselect = 0.05)

[1] "iter 1 num genes= 1"
[1] "iter 2 num genes= 2"
[1] "iter 3 num genes= 0"
[1] "iter 4 num genes= 0"
[1] "iter 5 num genes= 0"
[1] "iter 6 num genes= 1"
```

```
[1] "iter 7 num genes= 2"
[1] "iter 8 num genes= 0"
[1] "iter 9 num genes= 1"
[1] "iter 10 num genes= 0"
[1] "iter 11 num genes= 0"
[1] "iter 12 num genes= 1"
[1] "iter 13 num genes= 1"
[1] "iter 14 num genes= 1"
[1] "iter 15 num genes= 0"
[1] "iter 16 num genes= 1"
[1] "iter 17 num genes= 1"
[1] "iter 18 num genes= 0"
[1] "iter 19 num genes= 0"
[1] "iter 20 num genes= 0"
[1] "iter 21 num genes= 0"
[1] "iter 22 num genes= 1"
[1] "iter 23 num genes= 1"
[1] "iter 24 num genes= 0"
[1] "iter 25 num genes= 1"
[1] "iter 26 num genes= 0"
```

```
> genes.used <- multiget(ls(env = aggenv(Agg)), env = aggenv(Agg))
> genes.counts <- as.numeric(genes.used)
> names(genes.counts) <- names(genes.used)
> sort(genes.counts)

AFFX-BioB-M_st AFFX-PheX-5_at AFFX-BioB-3_at
             1              1              2
```

The variable `genes.counts` now contains, for each gene, the number of times it was selected in the cross-validation.

One may also compare testcase with geneCov to see how well the procedure worked at predicting.