Principal Component of Explained Variance

Maxime Turgeon

October 10, 2017

This package implements the dimension-reduction technique known as Principal Component of Explained Variance (PCEV), which is similar in spirit to Principal Component Analysis (PCA). The theoretical details are given in the next section. This is followed by a description of the user interface of the package `pcev`, and the last section presents a data analysis using this approach.

1 General theoretical framework

We consider the following setting: let Y be a multivariate phenotype of dimension p (e.g. methylation values at CpG sites, or brain imaging measured at p locations in the brain), let X be a q-dimensional vector of covariates of interest (e.g.: smoking, cell type or SNPs) and let C be an r-dimensional vector of confounders. We assume that the relationship between Y and X can be represented via a linear model

$$Y = BX + \Gamma C + E,$$

where B and Γ are $p \times q$ and $p \times r$ matrices of regression coefficients for the covariates of interest and confounders, respectively, and $E \sim N(0, \sigma)$ is a vector of residual errors. This model assumption allows us to decompose the total variance of Y as follows:

$$\begin{aligned}\mathrm{Var}(Y) &= \mathrm{Var}(BX) + \mathrm{Var}(\Gamma C) + \mathrm{Var}(E) \\ &= B\mathrm{Var}(X)B^T + \Gamma\mathrm{Var}(C)\Gamma^T + \Sigma \\ &= V_M + V_C + V_R,\end{aligned}$$

where $V_M = B\mathrm{Var}(X)B^T$ is the model component and $V_R = \Sigma$ is the residual component. PCEV seeks a linear combination of outcomes, w^TY , which maximises the ratio $h^2(w)$ of variance being explained by the covariates X, such that:

$$\begin{aligned}h^2(w) &= \frac{\mathrm{Var}(w^TBX)}{\mathrm{Var}(w^TE)} \\ &= \frac{w^TV_Mw}{w^TV_Rw}.\end{aligned}$$

PCEV thus seeks the vector w which maximises the criterion $h^2(w)$. It follows from a Lagrange multiplier-type argument that w is the solution to the generalised eigenvector problem

$$V_M w = \lambda V_R w,$$

and therefore standard linear algebraic results can be used to get a closed form for $w_{\text{PCEV}} := \text{argmax}_w \, h^2(w)$.

1.1 High-dimensional response vector

When p is larger than the sample size, a naïve implementation of PCEV will fail. For this reason, we proposed a novel alternative, namely a block approach to the estimation of PCEV. Assume we can partition Y into blocks (or clusters) in such a way that the number of components in a given block is small enough (i.e. smaller than n). We can then perform PCEV and get a linear combination $\tilde{Y}_j$ of the traits belonging to the jth block, for each block $j = 1, \ldots, b$. We then obtain a new multivariate pseudo-phenotype:

$$\tilde{\mathbf{Y}} = (\tilde{Y}_1, \ldots, \tilde{Y}_b),$$

on which we can again perform PCEV.

Although one might think that this stepwise approach is an ad-hoc extension of the original PCEV approach, it has nonetheless a very appealing and relevant mathematical property, described in the following result:

Theorem 1.1. *Assume one can partition the outcomes Y into blocks in such a way that blocks are uncorrelated (i.e. outcomes lying in different blocks are uncorrelated). Then the linear combination (PCEV) obtained from the traditional approach and that obtained from the stepwise block approach described above are equal.*

Another approach to dealing with a high-dimensional response vector is to replace the classical linear regression estimator of V_R by a shrinkage estimator, e.g. the Ledoit-Wolf linear shrinkage estimator. Both of these approaches, and a combination of the two, are implemented in this packages.

1.2 Association tests

Two testing procedures are implemented in this package:

1. An asymptotic test, using either Wilks' Lambda or Roy's Largest Root statistic.

2. A permutation test.

Johnstone (2009) derived an approximation of the null distribution of Roy's Largest Root; this is what we use in the implementation.

Specific details about which test can be used under a given estimation scheme are given below, with examples.

2 Examples

The main function is `computePCEV`, and indeed most users will only need this one function. Its two main parameters are `response`, which needs to be a $n \times p$ matrix containing the values for the response variables, and `covariate`, which can be either an array or a data frame containing the values of the covariates. Let's look at a simple example:

```
library(pcev)
set.seed(12345)

Y <- matrix(rnorm(100*20), nrow=100)
X <- rnorm(100)

pcev_out <- computePCEV(Y, X)
pcev_out

##
## Principal component of explained variance
##
##  100 observations, 20 response variables
##
## Estimation method: all
## Inference method: exact
## (performed using Roy's largest root test)
## P-value obtained: 0.8430205
##
## Variable importance factors (truncated)
## 0.633 0.308 0.249 0.243 0.220 0.215 0.210 0.192 0.183 0.177
```

The default behaviour of the function is to use the classical approach to PCEV (i.e. no block) and use Roy's Largest Root test. This test works for an arbitrary number of covariates. On the other hand, when there is only one covariate, we can also use Wilks' Lambda test, whose null distribution is known exactly in this setting:

```
pcev_out2 <- computePCEV(Y, X, Wilks=TRUE)
pcev_out2

##
## Principal component of explained variance
##
##  100 observations, 20 response variables
##
## Estimation method: all
## Inference method: exact
```

```
## (performed using Wilks' lambda test)
## P-value obtained: 0.8104
##
## Variable importance factors (truncated)
## 0.633 0.308 0.249 0.243 0.220 0.215 0.210 0.192 0.183 0.177
```

The output also gives the 10 highest variable importance factors (defined as the correlation between a response variable and the PCEV, in absolute value).

There is also the possibility of using the Ledoit-Wolf linear shrinkage estimator for the residual variance component:

```
pcev_out3 <- computePCEV(Y, X, shrink=TRUE)
pcev_out3

##
## Principal component of explained variance
##
##  100 observations, 20 response variables
##
## Estimation method: all
## Inference method: exact
## (performed using Roy's largest root test)
## P-value obtained: 0.8166869
##
## Shrinkage parameter rho was estimated at 0.9622696
##
## Variable importance factors (truncated)
## 0.672 0.376 0.333 0.291 0.278 0.261 0.240 0.232 0.223 0.212
```

The p-value is computed using a modification of Johnstone's approximation: a small number of permutations are performed[1], and for each permutation Roy's Largest Root statistic is computed. Using these values, the parameters of a location-scale variant of the Tracy-Widom distribution of order 1 are then estimated using Maximum Likelihood, and this distribution is used to compute a p-value. There is currently no theoretical justification for doing this, but preliminary empirical results look promising.

3 Data analysis

Finally, we will show how to use the block approach to PCEV using a real data analysis. We have methylation levels of 5,986 CpG sites, measured on 40 samples using bisulfite sequencing[2]. Each sample corresponds to one of three cell types: B cells (8 samples), T cells (19 samples), and monocytes (13 samples). Note

[1] This is set at 25. Currently, there is no way for the user to change this number.

[2] The data was knidly provided by Tomi Pastinen, McGill University

that the CpG sites are situated aroung the BLK gene, which is known to be differentially methylated across cell types. First, we load the data:

```
data(methylation)
data(pheno)
data(position)
```

The phenotype of interest in binary: $X = 1$ if the sample comes from a B cell, and $X = 0$ otherwise. We also have information about the genomic position of each CpG site.

First, we will look at a Manhattan plot of the data (Figure 1). Recall that the x-axis corresponds to the genomic position of a CpG, and the y-axis to the negative log of the p-value for the univariate association. As we can see, there is small region around 11.3 Mb with a strong univariate signal; this corresponds to the DMR. The code for producing the plot is given below:

```
# Compute univariate p-values
fit <- lm(methylation ~ pheno)
pval <- vapply(summary(fit), function(sum) {
  pvalue <- sum$coef[2,4]
  return(pvalue)
}, numeric(1))

# Manhattan plot univariate
plot(position$Pos/1e6, -log10(pval), xlab="Position (Mb)",
     ylab="-log10 pvalue", pch=19, cex=0.5)
abline(h=-log10(8.3*10^-6), lty=2)
```

Since the number of CpG sites is much greater than the sample size, we cannot use the classical approach to PCEV. Instead we will use the block approach. A natural to define these blocks is to use genomic distance. This can be easily done using the function `clusterMaker` in the package `bumphunter`:

```
# Break the region into sub-regions of 500 kB
cl <- bumphunter::clusterMaker(chr=position$Chr,
                               pos=position$Pos,
                               assumeSorted=TRUE,
                               maxGap = 500)

# Some blocks are too big... put limit at 30
index <- cl
maxInd <- max(index) + 1

blockLengths <- table(index)
while(sum(blockLengths > 30) > 0) {
```

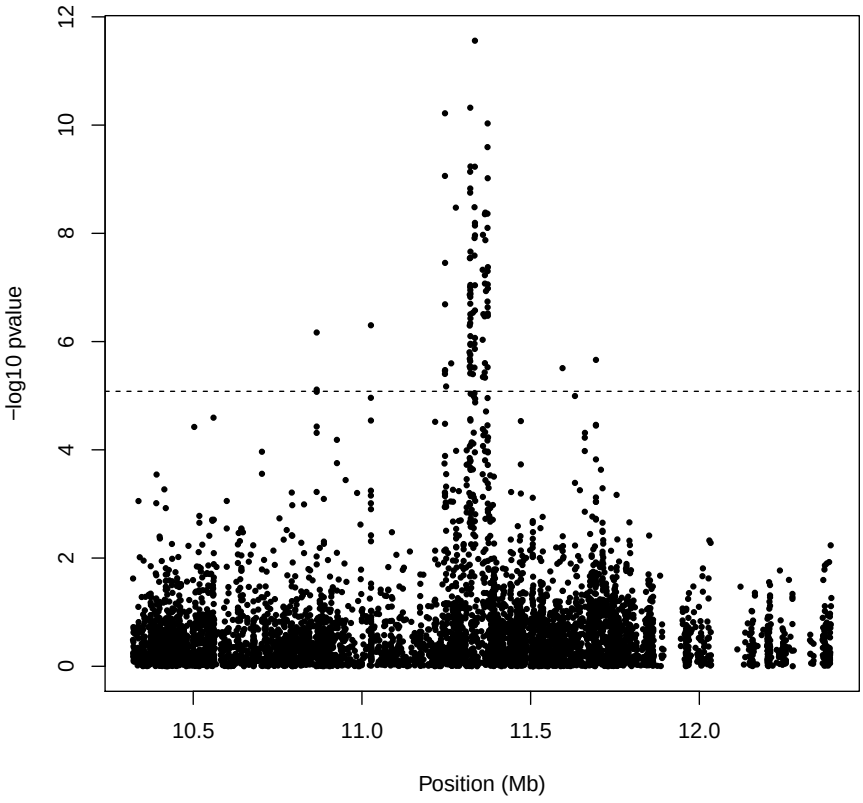

Figure 1: Manhattan plot

```
  for (j in unique(index)) {
    p <- length(index[index == j])
    if (p > 30) {
      q <- floor(p/2); r <- p - q
      index[index == j] <- c(rep_len(maxInd, q),
                             rep_len(maxInd + 1, r))
      maxInd <- maxInd + 2
    }
  }
  blockLengths <- table(index)
}

# Re-index so that we have consecutive indices
cl <- index
index <- cl
counter <- 0
for(j in sort(unique(cl))) {
  counter <- counter + 1
  index[index == j] <- counter
}
```

Another way to define the blocks would be hierarchical clustering. In any case, since we do not want this package to depend on `bumphunter`, we provide the vector of indices:

```
data(index)
table(table(index))

##
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
## 303 160  72  68  56  31  28  30  17  18  13   9  15   6  17  24  14  15
##  19  20  21  22  23  24  25  26  27  28  29  30
##  13  13   8   9   4   9   4   4   3   4   5   8
```

As we can see, there are no more than 30 CpG sites per block, and we have a total of 980 blocks. We are now ready to compute the PCEV:

```
pcev_out <- computePCEV(methylation, covariate = pheno, estimation = "block",
    inference = "permutation", index = index, nperm = 10)

## Warning in if (index != "adaptive") {: the condition has length
> 1 and only the first element will be used
## Selecting estimation by block.
## Warning in if (index == "adapative") {: the condition has length
> 1 and only the first element will be used
```

```
## Warning in beta_total/crossprod(beta_total):  Recycling array of
length 1 in vector-array arithmetic is deprecated.
##  Use c() or as.vector() instead.
## Warning in beta_total/crossprod(beta_total):  Recycling array of
length 1 in vector-array arithmetic is deprecated.
##  Use c() or as.vector() instead.
## Warning in beta_total/crossprod(beta_total):  Recycling array of
length 1 in vector-array arithmetic is deprecated.
##  Use c() or as.vector() instead.
## Warning in beta_total/crossprod(beta_total):  Recycling array of
length 1 in vector-array arithmetic is deprecated.
##  Use c() or as.vector() instead.
## Warning in beta_total/crossprod(beta_total):  Recycling array of
length 1 in vector-array arithmetic is deprecated.
##  Use c() or as.vector() instead.
## Warning in beta_total/crossprod(beta_total):  Recycling array of
length 1 in vector-array arithmetic is deprecated.
##  Use c() or as.vector() instead.
## Warning in beta_total/crossprod(beta_total):  Recycling array of
length 1 in vector-array arithmetic is deprecated.
##  Use c() or as.vector() instead.
## Warning in beta_total/crossprod(beta_total):  Recycling array of
length 1 in vector-array arithmetic is deprecated.
##  Use c() or as.vector() instead.
## Warning in beta_total/crossprod(beta_total):  Recycling array of
length 1 in vector-array arithmetic is deprecated.
##  Use c() or as.vector() instead.
## Warning in beta_total/crossprod(beta_total):  Recycling array of
length 1 in vector-array arithmetic is deprecated.
##  Use c() or as.vector() instead.
## Warning in beta_total/crossprod(beta_total):  Recycling array of
length 1 in vector-array arithmetic is deprecated.
##  Use c() or as.vector() instead.
```

```
pcev_out
```

```
##
## Principal component of explained variance
##
##  40 observations, 5986 response variables
##
## Estimation method: block
## Inference method: permutation
## P-value obtained: < 0.1
##
## Variable importance factors (truncated)
```

```
## 0.867 0.850 0.838 0.836 0.825 0.822 0.822 0.820 0.819 0.810
```

We used 10 permutations to compute the p-value, which is of course very low. Using 10,000 permutations, we got an approximate p-value of 4×10^{-4}.

We can construct something resembling a Manhattan plot, but where the univariate p-values are replaced by the *variable importance factor*. Figure 2 shows this type of plot for our particular data. We have also identified the BLK gene using a red line:

```
# Manhattan plot VIMP
BLK_boundaries <- c(11235000, 11385000)
plot(position$Pos/1e6, pcev_out$VIMP, xlab = "Position (Mb)",
     ylab = "Variable Importance", pch = 19, cex = 0.5,
     ylim = c(0,1))
lines(x = BLK_boundaries/1e6, y = rep_len(0.9,2),
      lwd = 3, col = 'red')
```

As we can see, the VIMP can serve as surrogates for the univariate p-values when it comes to identifying the most important response variables. In our case, it is able to capture the gene of interest.

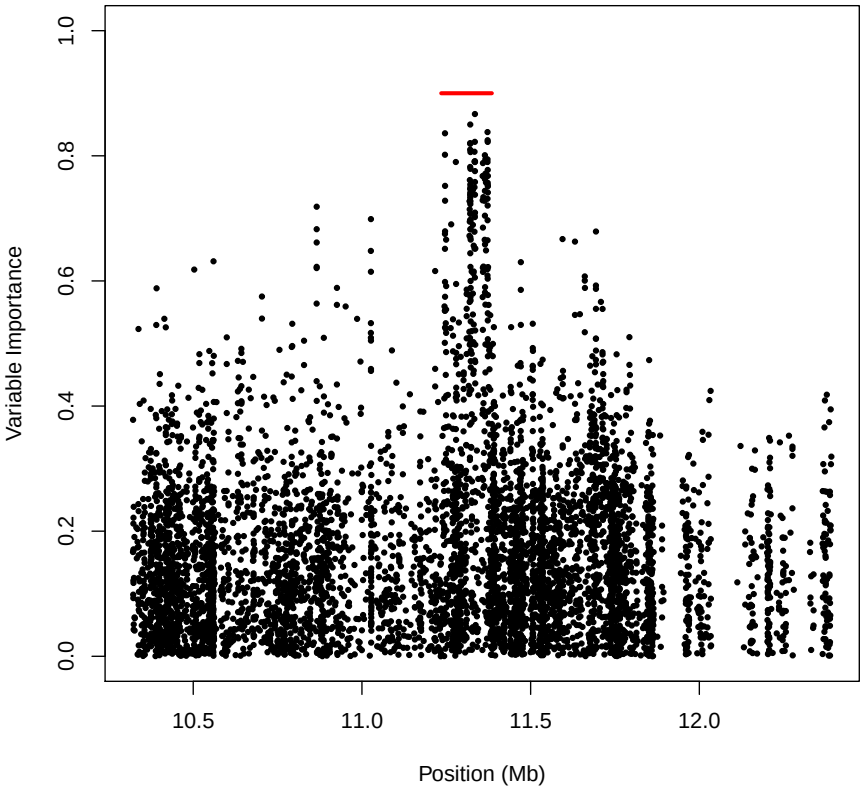

Figure 2: Manhattan-VIMP plot

4 Session Info

```
sessionInfo()

## R version 3.4.1 (2017-06-30)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu Zesty Zapus (development branch)
##
## Matrix products: default
## BLAS: /usr/lib/atlas-base/atlas/libblas.so.3.0
## LAPACK: /usr/lib/atlas-base/atlas/liblapack.so.3.0
##
## locale:
##  [1] LC_CTYPE=en_CA.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_CA.UTF-8        LC_COLLATE=C
##  [5] LC_MONETARY=en_CA.UTF-8    LC_MESSAGES=en_CA.UTF-8
##  [7] LC_PAPER=en_CA.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_CA.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] knitr_1.17 pcev_2.2.1
##
## loaded via a namespace (and not attached):
##  [1] compiler_3.4.1   magrittr_1.5     formatR_1.4      tools_3.4.1
##  [5] codetools_0.2-14 stringi_1.1.2    highr_0.6        digest_0.6.12
##  [9] stringr_1.2.0    RMTstat_0.3      evaluate_0.10.1
```