SNP-wise General Weights

Stuart Brabbs

Our goal here is to build a function that can output the SNP-wise regression weights, in the form of Bayes factor times prior, for a general situation of p parameters and 2^p possible models. We first define again the logBF function:

```
logBF = function(g,y,sigmaa) {
  p=dim(g)[2]
  n=dim(g)[1]
  if (is.null(dim(g)[2])){
  g = g - mean(g)
  y = y - mean(y)
  n=length(g)
  X = g
  invnu = 1/sigmaa^2
  invOmega = invnu + t(X) %*% X
  B = (t(X) %*% cbind(y))/invOmega
  invOmega0 = n
  return(-0.5*log10(det(invOmega)) + 0.5*log10(invOmega0) - log10(sigmaa) - (n/2)*(log10
(t(y- X %*% B) %*% y) - log10(t(y) %*% y)))
  }
  else {
    g = scale(g, scale = FALSE)
    y = scale(y, scale = FALSE)
    X = g
    invnu = diag(rep(1/sigmaa^2, p))
    invOmega = invnu + t(X) %*% X
    B = solve(invOmega, t(X) %*% cbind(y))
    invOmega0 = n
    return(-0.5*log10(det(invOmega)) + 0.5*log10(invOmega0) - p*log10(sigmaa) - (n/2)*(l
og10(t(y- X %*% B) %*% y) - log10(t(y) %*% y - n*mean(y)^2)))
  }
}
```

for(i in c(1:3)){ if(combs[6,i]==1){ numincl <- numincl + 1} }

```
snpwise_weights = function(X,y,sigmaa) {
  models <- c()
  wts <- c()

  #get number of parameters
  if (is.null(dim(X)[2])) {
    par <- 1
  }
  else {
    par <- dim(X)[2]
  }

  #get all possible parameter combinations
  l <- rep(list(0:1),par)
  combs <- expand.grid(l)
  m <- dim(combs)[1]

  for (i in c(1:m)){
    numlist <- c()
    incl <- c()
    numincl <- 0

    for (j in c(1:par)){
      if (combs[i,j]==1){
        incl <- cbind(incl, X[,j])
        numincl <- numincl + 1
      }
    }
    numlist <- c(numlist, numincl)

    #if only one variable in model
    if (numincl==1) {
      for (k in c(1:par)){
        if (combs[i,k]==1){
          varin <- k
        }
      }
      newmod <- paste0("X", varin)

      #get weight
      prior <- (1/par)*(1-1/par)^(1-par)
      lbf <- logBF(X[,varin], y, sigmaa)
      post <- prior * lbf
    }

    #if multiple variables in model
    else if (numincl != 0){
      newmod <- ""
      for (j in c(1:par)){

        if (combs[i,j]==1){
          var <- paste0("X", j)
          newmod <- paste(newmod, sep =",", var)
```

```
        }
      }
      newmod <- sub('.', '', newmod)
      #get weight
      prior <- (1/par)^numincl * (1-1/par)^(1-numincl)
      lbf <- logBF(incl, y, sigmaa)
      post <- prior * lbf
    }

    #if null model
    else {
      post <- (1-1/par)^(par)
      newmod <- "NULL"
    }

    models <- c(models, newmod)
    #add weight to list
    wts <- c(wts, post)
  }

  #return dataframe of each model and its posterior weight
  toreturn <- data.frame(models, wts)
  names(toreturn)[1] <- "models"
  names(toreturn)[2] <- "weights"
  return(toreturn)
}
```

We will test this function with an example of the three body problem from previously, with n=100, X1 and X3 the true model, and X2 correlated with X1 and X3:

```
x1 <- rnorm(100)
x3 <- rnorm(100)
x2 <- x1 + x3 + rnorm(100, sd = 0.5)
y <- x1 + x3 + rnorm(100)
X <- cbind(x1, x2, x3)

snpwise_weights(X, y, 0.5)
```

```
##     models    weights
## 1     NULL  0.2962963
## 2       X1  8.0043140
## 3       X2 13.1008188
## 4    X1,X2  2.9521154
## 5       X3  6.3523560
## 6    X1,X3  3.6384566
## 7    X2,X3  2.9322910
## 8 X1,X2,X3  1.7812166
```

We can also define the function so that the weights use the Bayes factor (i.e. exponentiated logBF) instead of the logBF. We exclude the text of the function because we simply change any expression with "logBF(...)" to "10^logBF(...)".

We then rerun our previous test, but with the new function:

```
exp_snpwise_weights(X, y, 0.5)
```

```
##     models      weights
## 1     NULL 2.962963e-01
## 2       X1 3.527605e+10
## 3       X2 2.202012e+17
## 4    X1,X2 8.600841e+16
## 5       X3 2.212429e+08
## 6    X1,X3 1.128726e+21
## 7    X2,X3 6.540256e+16
## 8 X1,X2,X3 1.974322e+20
```

As expected, given we are dealing with base 10, the differences between the weights are now much larger. In both cases, we can see that X1 and X3, the true model, is the most-weighted model with multiple variables, but due to the penalty from the prior on including more variables, the X2 only model is the highest weighted.

We can also implement the function with an example where we add to the three-body problem an unrelated fourth variable:

```
x1 <- rnorm(100)
x3 <- rnorm(100)
x2 <- x1 + x3 + rnorm(100, sd = 0.5)
x4 <- rnorm(100)
y <- x1 + x3 + rnorm(100)
X <- cbind(x1, x2, x3,x4)

snpwise_weights(X, y, 0.5)
```

```
##         models    weights
## 1         NULL  0.3164062
## 2           X1  4.0711292
## 3           X2 10.8616949
## 4        X1,X2  1.4968136
## 5           X3  7.6620266
## 6        X1,X3  1.8399682
## 7        X2,X3  1.6076730
## 8     X1,X2,X3  0.6051715
## 9           X4  0.3918613
## 10       X1,X4  0.5521298
## 11       X2,X4  1.5193386
## 12    X1,X2,X4  0.4965219
## 13       X3,X4  1.0448730
## 14    X1,X3,X4  0.6070051
## 15    X2,X3,X4  0.5312738
## 16 X1,X2,X3,X4  0.1999066
```

Almost all of the models including X4 have weights of less than 1. The highest weight goes to X2, and the highest weight for a multivariable model goes to X1 and X3, indicating that the function is working as desired.

On another note, we detail below a method for producing the kind of correlated data for the three-body problem in which we can input a correlation rather than a standard deviation for error. It uses the rnorm_multi function from the "faux" package, and allows you to input correlations into a vector r = c(a-b, a-c, b-c). When setting the a-c correlation to 0, the maximum possible correlation for a-b and b-c is about 0.7:

```
threebody_data = function(rows, cor){
  dat <- rnorm_multi(n=rows, mu=c(0,0,0), sd=c(1,1,1), r=c(cor,0,cor), empirical = TRUE)
  return(dat)
}
```

As an example, we produce data at the maximum correlation of 0.7, and verify:

```
df <- threebody_data(100,0.7)
#X1-X2 correlation
cor(df[,1], df[,2])
```

```
## [1] 0.7
```

```
#X2-X3 correlation
cor(df[,2], df[,3])
```

```
## [1] 0.7
```

```
#X1-X3 correlation
cor(df[,1], df[,3])
```

```
## [1] -1.403412e-16
```