Compare

Alan n. Inglis

2022-09-06

Load libraries:

```
library(dbarts) # for model
library(bartMan) # for visualizations
library(vivid) # for agnostic visualizations
library(ggplot2) # for visualizations
```

Read in and setup data:

```
# Create some data
f <- function(x) {
  10 * sin(pi * x[, 1] * x[, 2]) + 20 * (x[, 3] - 0.5)^2 +
    10 * x[, 4] + 5 * x[, 5]
}


set.seed(1701)
sigma <- 1.0
n <- 250
x <- matrix(runif(n * 10), n, 10)
colnames(x) <- paste0("x", 1:10)
Ey <- f(x)
y <- rnorm(n, Ey, sigma)
fData <- as.data.frame(cbind(x, y))

x <- fData[, 1:10]
y <- fData$y
```

Build models

```
set.seed(1701)
dB20 <- bart(x.train = x,
             y.train = y,
             ntree = 20,
             keeptrees = TRUE,
             nskip = 100,
             ndpost = 1000
)
```

```
##
## Running BART with numeric y
##
## number of trees: 20
## number of chains: 1, number of threads 1
## tree thinning rate: 1
## Prior:
##  k prior fixed to 2.000000
##  degrees of freedom in sigma prior: 3.000000
##  quantile in sigma prior: 0.900000
##  scale in sigma prior: 0.002377
##  power and base for tree prior: 2.000000 0.950000
##  use quantiles for rule cut points: false
##  proposal probabilities: birth/death 0.50, swap 0.10, change 0.40; birth 0.50
## data:
##  number of training observations: 250
##  number of test observations: 0
##  number of explanatory variables: 10
##  init sigma: 2.766176, curr sigma: 2.766176
##
## Cutoff rules c in x<=c vs x>c
## Number of cutoffs: (var: number of possible c):
## (1: 100) (2: 100) (3: 100) (4: 100) (5: 100)
## (6: 100) (7: 100) (8: 100) (9: 100) (10: 100)
##
## Running mcmc loop:
## iteration: 100 (of 1000)
## iteration: 200 (of 1000)
## iteration: 300 (of 1000)
## iteration: 400 (of 1000)
## iteration: 500 (of 1000)
## iteration: 600 (of 1000)
## iteration: 700 (of 1000)
## iteration: 800 (of 1000)
## iteration: 900 (of 1000)
## iteration: 1000 (of 1000)
## total seconds in loop: 0.127769
##
## Tree sizes, last iteration:
## [1] 3 2 3 2 3 4 6 2 4 3 2 2 2 3 2 4 2 3
## 4 4
##
## Variable Usage, last iteration (var:count):
## (1: 7) (2: 8) (3: 8) (4: 8) (5: 5)
## (6: 1) (7: 0) (8: 2) (9: 0) (10: 1)
##
## DONE BART
```

```
set.seed(1701)
dB100 <- bart(x.train = x,
              y.train = y,
              ntree = 100,
              keeptrees = TRUE,
              nskip = 100,
              ndpost = 1000
)
```

```
##
## Running BART with numeric y
##
## number of trees: 100
## number of chains: 1, number of threads 1
## tree thinning rate: 1
## Prior:
##  k prior fixed to 2.000000
##  degrees of freedom in sigma prior: 3.000000
##  quantile in sigma prior: 0.900000
##  scale in sigma prior: 0.002377
##  power and base for tree prior: 2.000000 0.950000
##  use quantiles for rule cut points: false
##  proposal probabilities: birth/death 0.50, swap 0.10, change 0.40; birth 0.50
## data:
##  number of training observations: 250
##  number of test observations: 0
##  number of explanatory variables: 10
##  init sigma: 2.766176, curr sigma: 2.766176
##
## Cutoff rules c in x<=c vs x>c
## Number of cutoffs: (var: number of possible c):
## (1: 100) (2: 100) (3: 100) (4: 100) (5: 100)
## (6: 100) (7: 100) (8: 100) (9: 100) (10: 100)
##
## Running mcmc loop:
## iteration: 100 (of 1000)
## iteration: 200 (of 1000)
## iteration: 300 (of 1000)
## iteration: 400 (of 1000)
## iteration: 500 (of 1000)
## iteration: 600 (of 1000)
## iteration: 700 (of 1000)
## iteration: 800 (of 1000)
## iteration: 900 (of 1000)
## iteration: 1000 (of 1000)
## total seconds in loop: 0.616317
##
## Tree sizes, last iteration:
## [1] 2 2 2 3 2 3 4 2 2 2 2 2 3 3 2 2 2 2
## 7 2 3 2 2 2 2 2 2 3 2 2 4 3 2 1 2 2 4 3
## 3 2 2 3 3 1 3 2 2 2 4 2 6 2 2 3 4 2 4 3
## 3 3 2 1 3 2 2 4 3 2 2 2 1 2 2 3 2 2 3 2
## 2 3 2 2 4 2 3 4 3 2 5 3 2 2 2 3 2 3 2 4
## 3 2
##
## Variable Usage, last iteration (var:count):
## (1: 26) (2: 20) (3: 22) (4: 18) (5: 11)
## (6: 12) (7: 12) (8: 10) (9: 14) (10: 10)
##
## DONE BART
```

```r
set.seed(1701)
dB200 <- bart(x.train = x,
              y.train = y,
              ntree = 200,
              keeptrees = TRUE,
              nskip = 100,
              ndpost = 1000
)
```

```
##
## Running BART with numeric y
##
## number of trees: 200
## number of chains: 1, number of threads 1
## tree thinning rate: 1
## Prior:
##  k prior fixed to 2.000000
##  degrees of freedom in sigma prior: 3.000000
##  quantile in sigma prior: 0.900000
##  scale in sigma prior: 0.002377
##  power and base for tree prior: 2.000000 0.950000
##  use quantiles for rule cut points: false
##  proposal probabilities: birth/death 0.50, swap 0.10, change 0.40; birth 0.50
## data:
##  number of training observations: 250
##  number of test observations: 0
##  number of explanatory variables: 10
##  init sigma: 2.766176, curr sigma: 2.766176
##
## Cutoff rules c in x<=c vs x>c
## Number of cutoffs: (var: number of possible c):
## (1: 100) (2: 100) (3: 100) (4: 100) (5: 100)
## (6: 100) (7: 100) (8: 100) (9: 100) (10: 100)
##
## Running mcmc loop:
## iteration: 100 (of 1000)
## iteration: 200 (of 1000)
## iteration: 300 (of 1000)
## iteration: 400 (of 1000)
## iteration: 500 (of 1000)
## iteration: 600 (of 1000)
## iteration: 700 (of 1000)
## iteration: 800 (of 1000)
## iteration: 900 (of 1000)
## iteration: 1000 (of 1000)
## total seconds in loop: 1.137779
##
## Tree sizes, last iteration:
## [1] 2 3 2 2 3 2 2 2 2 2 4 2 3 1 2 2 2 3
## 4 3 4 4 2 2 1 2 4 2 2 2 3 2 2 2 3 2 2 2
## 2 2 3 3 4 6 3 2 2 1 3 2 2 3 2 2 2 2 2 1
## 2 1 2 3 2 2 2 3 2 2 2 1 2 3 2 3 2 3 1 3
## 2 2 2 2 3 2 2 3 2 3 2 3 2 2 2 1 2 1 2 2
## 3 2 2 1 2 3 4 4 2 2 3 2 2 5 2 2 2 2 2 2
## 3 2 2 3 2 2 3 4 2 2 1 4 3 2 2 2 2 3 4 3
## 2 2 2 2 3 4 2 3 2 2 2 1 2 2 5 2 4 2 2 3
## 3 2 4 2 2 2 2 3 2 2 3 2 3 2 3 2 3 2 3 2
## 2 2 3 3 2 2 3 2 2 3 2 3 1 3 3 2 2 2 3 2
## 2 2
##
## Variable Usage, last iteration (var:count):
## (1: 38) (2: 35) (3: 27) (4: 25) (5: 22)
## (6: 28) (7: 22) (8: 21) (9: 22) (10: 34)
##
## DONE BART
```

Create dataframe of trees

```
dbT20 <- extractTreeData(model = dB20, data = fData)
```

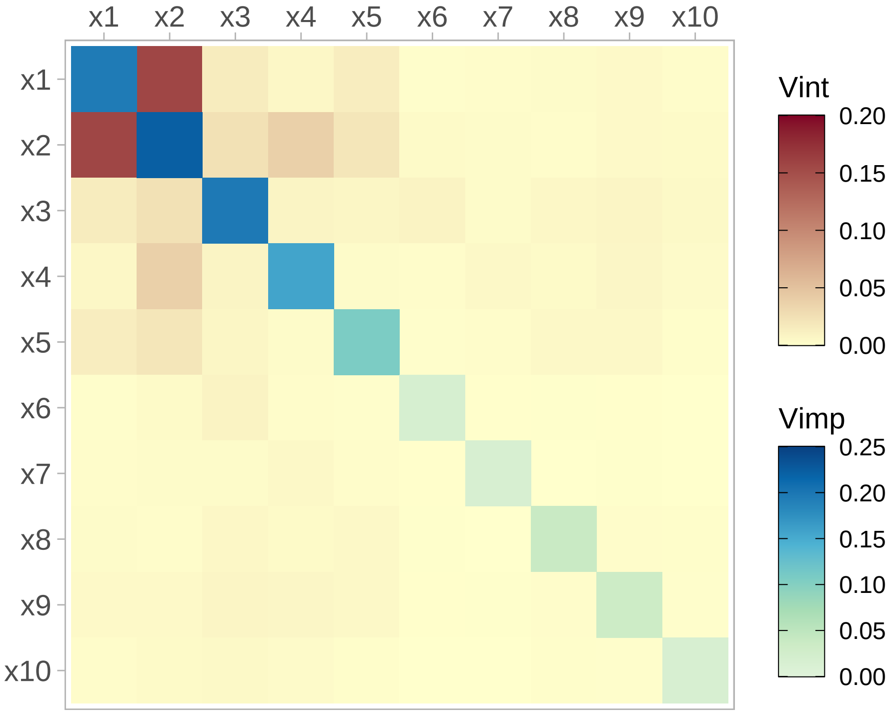
```
dbT100 <- extractTreeData(model = dB100, data = fData)
```

```
dbT200 <- extractTreeData(model = dB200, data = fData)
```

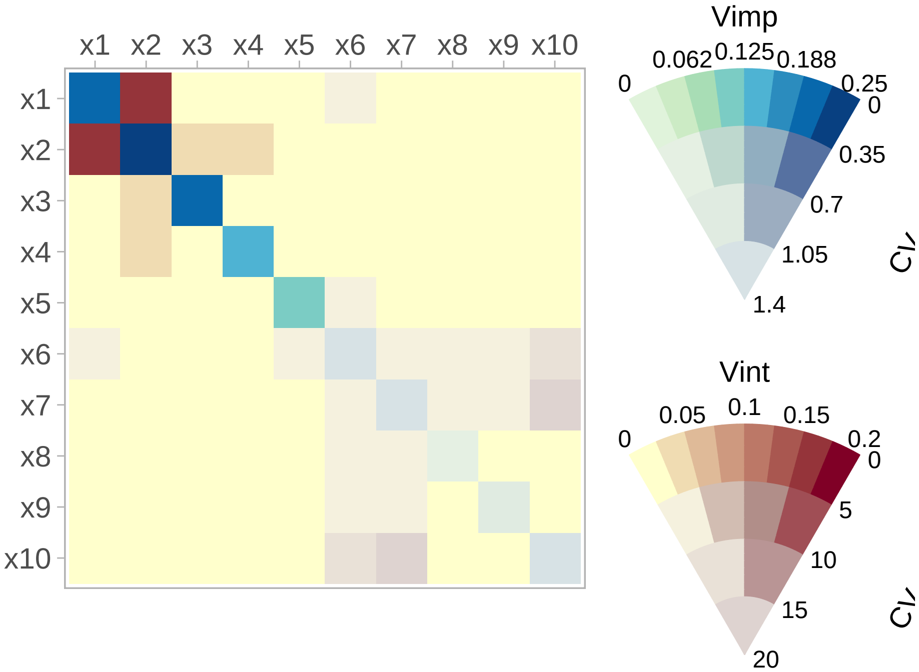
Figure 11:

20 trees

```
# plot standard heatmap
myMatStd <- viviBartMatrix(dbT20,
                           type = 'standard',
                           metric = 'propMean',
                           reorder = F)

viviBartPlot(myMatStd)
```

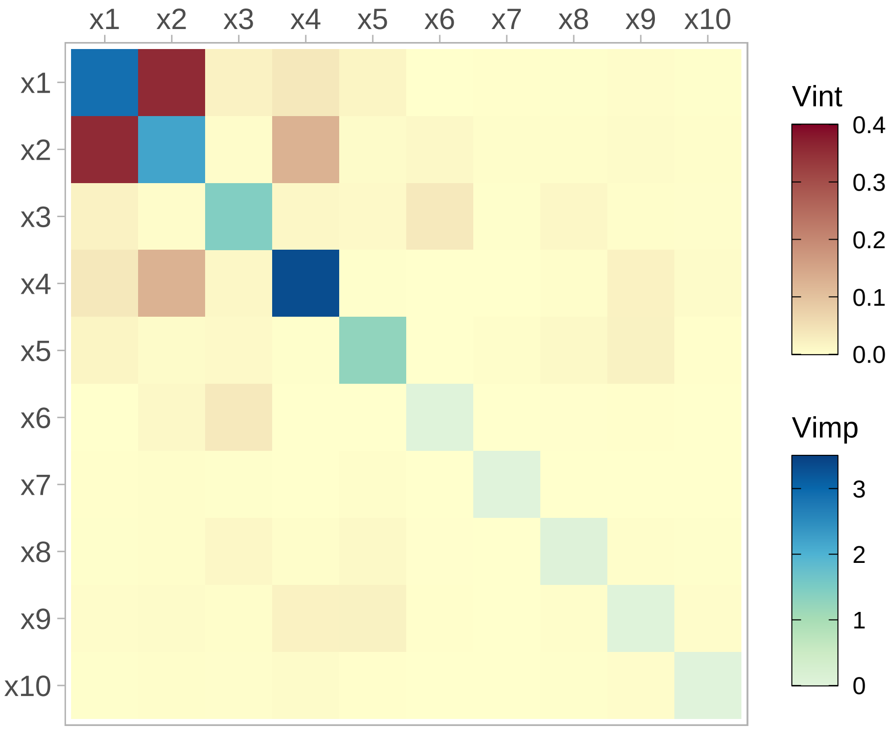


```
# plot vsup
myMat <- viviBartMatrix(dbT20,
                        type = 'vsup',
                        metric = 'propMean',
                        metricError = "CV",
                        reorder = F)


viviBartPlot(myMat,label = 'CV')
```



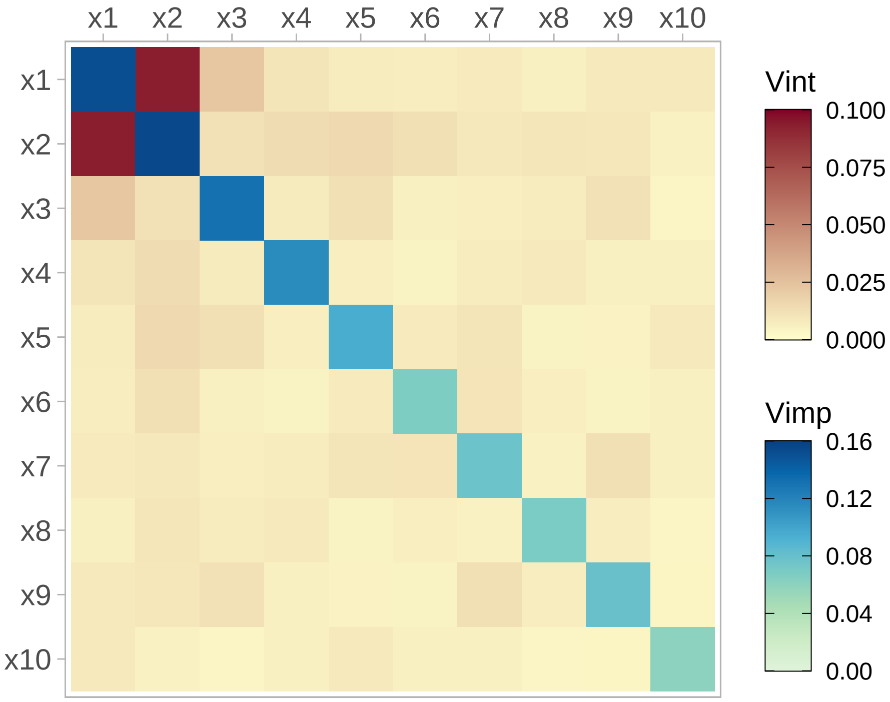
vivid for 20 trees

```r
response = 'y'
data = fData
# create predict function
responseIdx <- which(colnames(data) == response)
pFun <- function(fit, data, prob=TRUE) apply(predict(fit, data[,-responseIdx]), 2, mean)


# run vivid
set.seed(1701)
mat <- vivid::vivi(fit = dB20,
                   data = fData,
                   response = 'y',
                   reorder = F,
                   gridSize = 10,
                   nmax = 500,
                   normalized = FALSE,
                   class = 1,
                   predictFun = pFun)

colors <- scales::colour_ramp(
  colors = c(blue = '#FFFFCC', red = '#800026')
)((0:7)/7)
newCols <- RColorBrewer::brewer.pal(9, 'GnBu')
colors2 <- newCols[-1]

viviHeatmap(mat,
            intPal = colors,
            impPal = colors2)
```

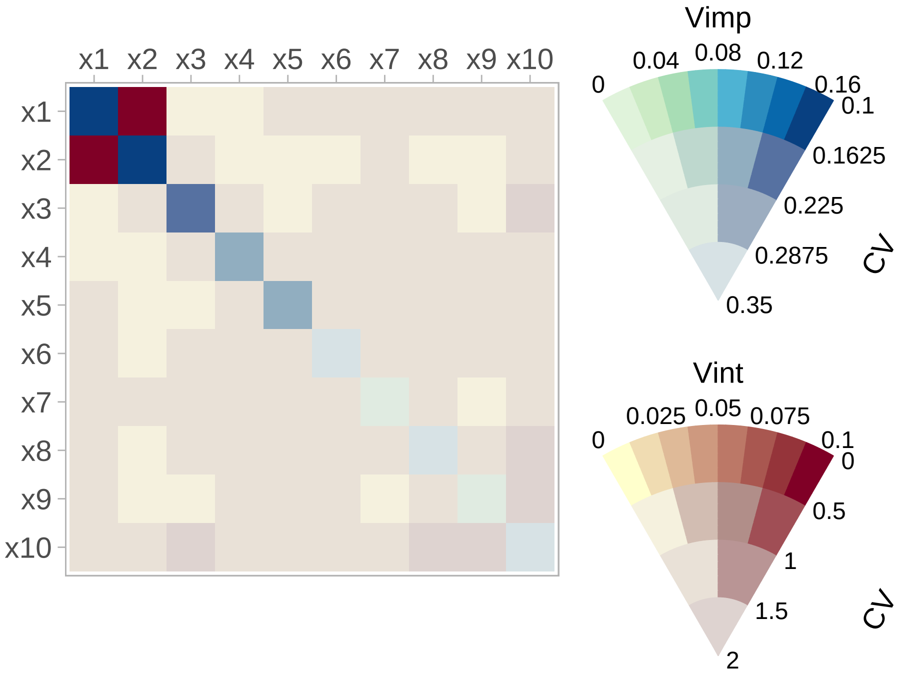


100 trees

```
# plot standard heatmap
myMatStd <- viviBartMatrix(dbT100,
                           type = 'standard',
                           metric = 'propMean',
                           reorder = F)

viviBartPlot(myMatStd, impLims = c(0, 0.16))
```

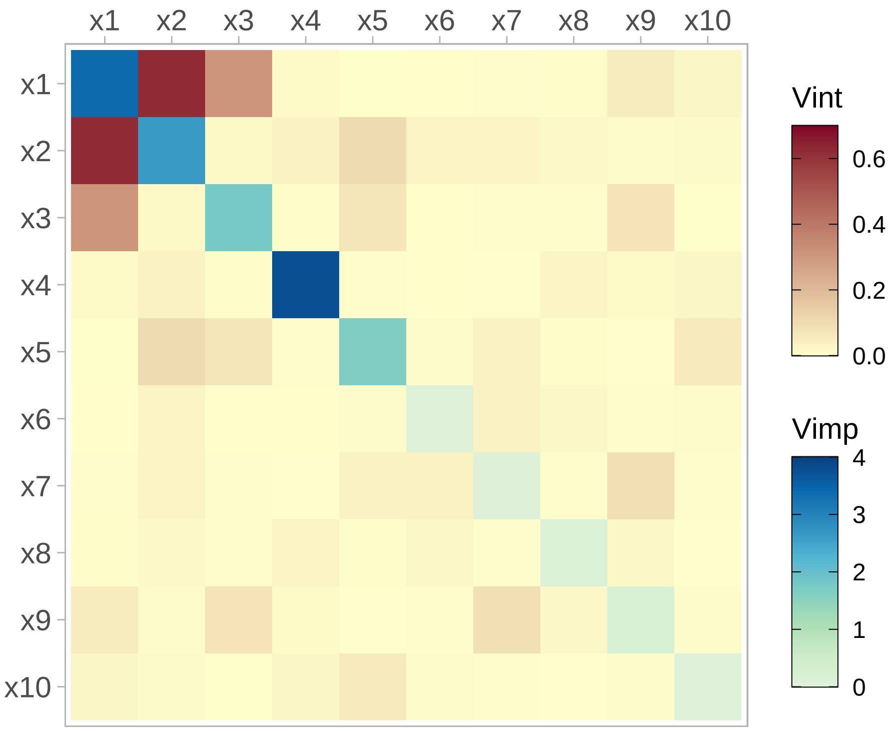


```
# plot vsup
myMat <- viviBartMatrix(dbT100,
                        type = 'vsup',
                        metric = 'propMean',
                        metricError = "CV",
                        reorder = F)


viviBartPlot(myMat, label = 'CV', impLims = c(0, 0.16))
```



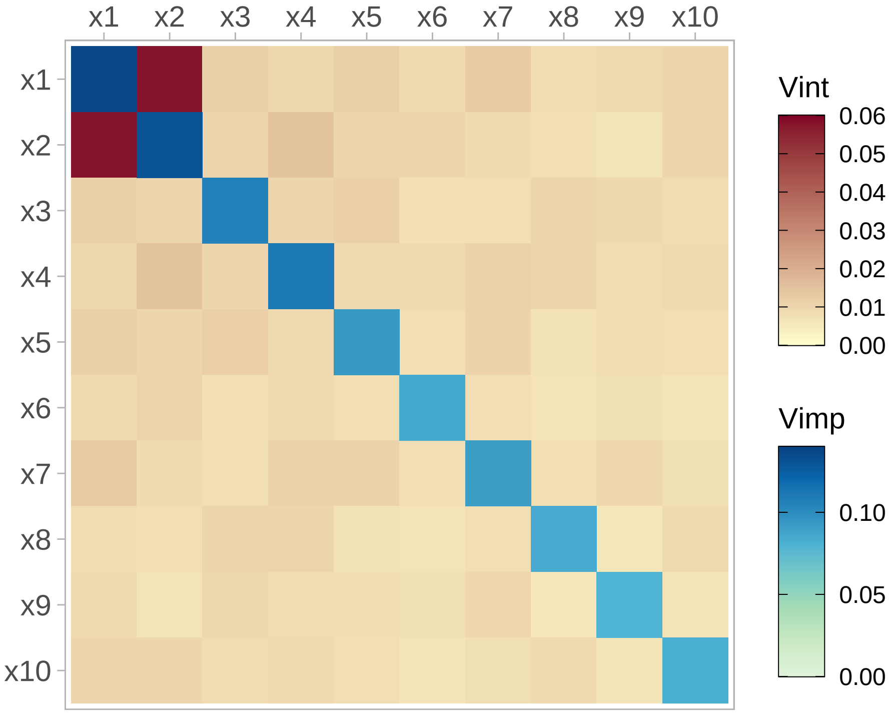
vivid for 100 trees

```
response = 'y'
data = fData
# create predict function
responseIdx <- which(colnames(data) == response)
pFun <- function(fit, data, prob=TRUE) apply(predict(fit, data[,-responseIdx]), 2, mean)


# run vivid
set.seed(1701)
mat <- vivid::vivi(fit = dB100,
                   data = fData,
                   response = 'y',
                   reorder = F,
                   gridSize = 10,
                   nmax = 500,
                   normalized = FALSE,
                   class = 1,
                   predictFun = pFun)

colors <- scales::colour_ramp(
  colors = c(blue = '#FFFFCC', red = '#800026')
)((0:7)/7)
newCols <- RColorBrewer::brewer.pal(9, 'GnBu')
colors2 <- newCols[-1]

viviHeatmap(mat,
            intPal = colors,
            impPal = colors2)
```

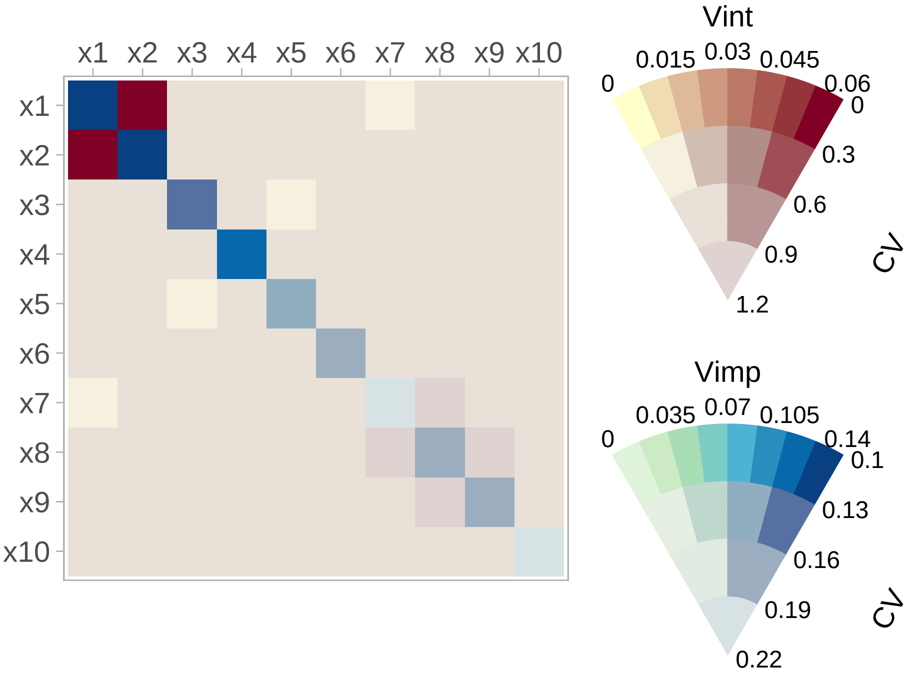


200 trees

```
# plot standard heatmap
myMatStd <- viviBartMatrix(dbT200,
                           type = 'standard',
                           metric = 'propMean',
                           reorder = F)

viviBartPlot(myMatStd, impLims = c(0, 0.14))
```

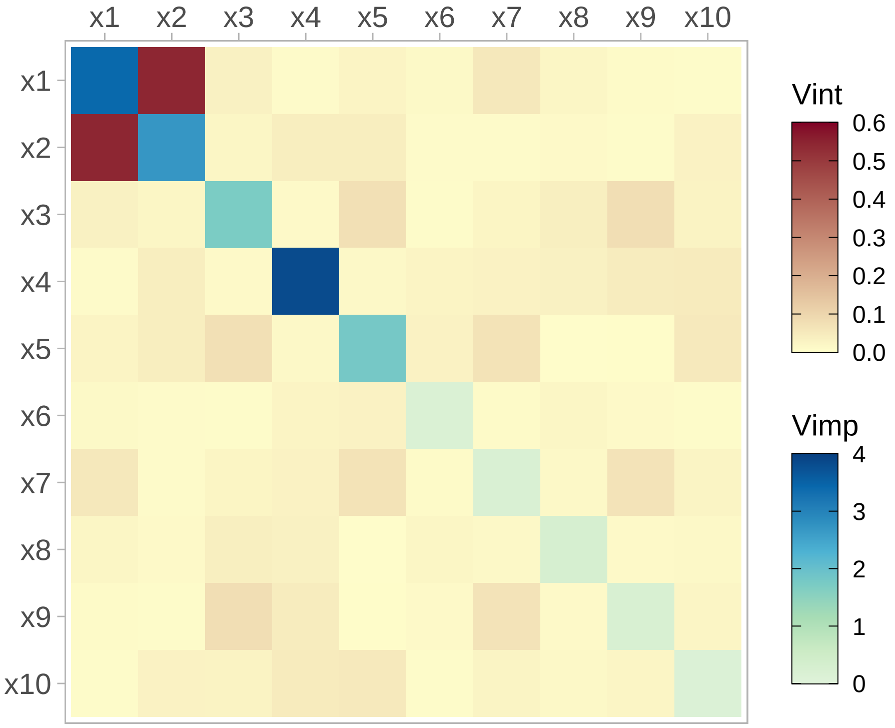


```
# plot vsup
myMat <- viviBartMatrix(dbT200,
                        type = 'vsup',
                        metric = 'propMean',
                        metricError = "CV",
                        reorder = F)


viviBartPlot(myMat, label = 'CV',impLims = c(0, 0.14))
```



vivid for 200 trees

```r
response = 'y'
data = fData
# create predict function
responseIdx <- which(colnames(data) == response)
pFun <- function(fit, data, prob=TRUE) apply(predict(fit, data[,-responseIdx]), 2, mean)


# run vivid
set.seed(1701)
mat <- vivid::vivi(fit = dB200,
                   data = fData,
                   response = 'y',
                   reorder = F,
                   gridSize = 10,
                   nmax = 500,
                   normalized = FALSE,
                   class = 1,
                   predictFun = pFun)

colors <- scales::colour_ramp(
  colors = c(blue = '#FFFFCC', red = '#800026')
)((0:7)/7)
newCols <- RColorBrewer::brewer.pal(9, 'GnBu')
colors2 <- newCols[-1]

viviHeatmap(mat,
            intPal = colors,
            impPal = colors2)
```


x1
x2
x3
x4
x5
x6
x7
x8
x9
x10
x1
x2
x3
x4
x5
x6
x7
x8
x9
x10
Vint
Vimp