Mapping extit{Hyalomma} and CCHF in Africa

Getting Started

So you're interested in using `embarcadero` to do species distribution modeling with Bayesian additive regression trees! That's great. BARTs are a powerful way to do machine learning and, while not a new method per se, they are very new for SDMs.

Most of the core functionality of `embarcadero` is actually a wrapper for `dbarts`, which runs the actual BART fitting process. This vignete will show you

1. How to run BARTs
2. Variable importance measures
3. Automated variable selection
4. Partial dependence plots
5. Visualizing the posterior distribution

There's also just going to be some general comments on the process of using BARTs, the challenges to working with them, and some things that are hopefully coming next.

```
library(embarcadero, quietly = TRUE)
#>
#> Attaching package: 'raster'
#> The following object is masked from 'package:dplyr':
#>
#>     select
#> The following object is masked from 'package:tidyr':
#>
#>     extract
#>
#> Attaching package: 'gplots'
#> The following object is masked from 'package:stats':
#>
#>     lowess
library(velox)

set.seed(12345)
```

Doors are closing; please stand clear of the doors.

Mapping Hyalomma

We're going to make a suitability layer for *Hyalomma truncatum*, a possible CCHF vector, that we can use in the CCHF map.

Data entry

Let's start by loading in the sample data and predictor set. Based on some expert opinion, I picked a handful of variables I thought might work well (and it's already reduced down to minimize redunandancy):

- BIO1: Mean annual temperature
- BIO2: Mean diurnal range

- BIO5: Max temperature of warmest month
- BIO6: Min temperature of coldest month
- BIO12: Mean annual precipitation
- BIO13: Precipitation of wettest month
- BIO14: Precipitation of driest month
- BIO15: Precipitation seasonality
- Mean NDVI (normalized difference vegetation index)
- NDVI amplitude
- Percent cropland by pixel

Let's read in the covariates, and make a bigger stack just to do some faster predictions along the way; `velox` works well for this. BARTs predict fairly slowly with `dbarts` because it's not parallelized yet. Other packages have that capacity, but less functionality for other things, so we chose functionality at the expense of speed; hopefully future versions will build on this.

```
files <- list.files('~/Github/embarcadero/vignettes/covariates', full.names=TRUE)
covs <- raster::stack(lapply(files, raster))

cov.big <- covs

  for(i in 1:nlayers(covs)) {
    vx <- velox(covs[[i]])
    vx$aggregate(factor=c(5,5), aggtype='mean')
    if (i == 1) { cov.big <- stack(vx$as.RasterLayer())
    } else { cov.big <- stack(cov.big,vx$as.RasterLayer())
    }
    print(i)
  }
#> [1] 1
#> [1] 2
#> [1] 3
#> [1] 4
#> [1] 5
#> [1] 6
#> [1] 7
#> [1] 8
#> [1] 9
#> [1] 10
#> [1] 11

names(cov.big) <- names(covs)
```

Hyalomma truncatum occurrence data is taken from the Cumming tick dataset

```
ticks <- read.csv('~/Github/embarcadero/vignettes/Hytr.csv')
head(ticks)
#>       X     ScientificName Longitude.X Latitude.Y
#> 1 16471 Hyalomma truncatum    15.83333   -8.15000
#> 2 16472 Hyalomma truncatum    14.13333  -12.46667
#> 3 16473 Hyalomma truncatum    12.20000   -5.55000
#> 4 16474 Hyalomma truncatum    15.18333  -16.50000
#> 5 16475 Hyalomma truncatum    13.95000  -13.56667
#> 6 16476 Hyalomma truncatum    15.26667  -12.43333
nrow(ticks)
#> [1] 1794
```

First, we extract the data from the presence points. But let's spatially thin those points first, to one point per raster grid cell, since they're a little over-aggregated. (Just a pretty normal part of ecological data.)

```
mod <- SpatialPointsDataFrame(ticks[,3:4],data.frame(ticks[,1]))
names(mod@data) <- 'Presence'
# Rasterizing makes unique points to the grid cell
tmp=rasterize(mod, covs[[1]], field="Presence", fun="min")
pts.sp1=rasterToPoints(tmp, fun=function(x){x>0})
nrow(pts.sp1)
#> [1] 1716

pres.cov <- raster::extract(covs, pts.sp1[,1:2])
head(pres.cov)
#>          bio1       bio12       bio13       bio14     bio15     bio2
#> [1,] 29.37254 0.006426269 0.008964164 0.004170780 0.1474124 23.18576
#> [2,] 30.45862 0.006053756 0.009221982 0.003706693 0.1708778 23.47737
#> [3,] 31.87390 0.009033033 0.011659561 0.006434741 0.1648283 22.30462
#> [4,] 31.69130 0.008533484 0.014504205 0.003622229 0.3815936 21.20000
#> [5,] 30.94236 0.008057307 0.015355557 0.002832554 0.4453886 20.80000
#> [6,] 31.70000 0.008724357 0.014759392 0.003717114 0.3831273 21.15962
#>          bio5      bio6  crop   ndvi.amp   ndvi.mean
#> [1,] 47.05830  8.030848 0.000 0.04955071 -0.01506902
#> [2,] 47.67215  8.694777 0.000 0.05624679  0.01562384
#> [3,] 47.77390 11.260820 0.000 0.10457005  0.09483840
#> [4,] 46.98304 14.683043 0.031 0.03216948  0.12690221
#> [5,] 46.38309 12.383087 0.011 0.01388101  0.12155575
#> [6,] 47.04038 14.900000 0.044 0.15113069 -0.03903103
```

Next, let's generate an equal number of pseudoabsences around Africa to the number of presences we have. BARTs are like BRTs in that they are sensitive to assumed prevalence, if not even more so than BRTs; I strongly suggest using an equal number of presences and absences in your training data. You can experiment with the demo data by changing "nrow(ticks)" to "5000" below if you want to see the model overfit.

```
#Generate the data
absence <- randomPoints(covs,nrow(ticks))
#> Warning in couldBeLonLat(mask): CRS is NA. Assuming it is longitude/
#> latitude
abs.cov <- raster::extract(covs, absence)

#Code the response
pres.cov <- data.frame(pres.cov); pres.cov$tick <- 1
abs.cov <- data.frame(abs.cov); abs.cov$tick <- 0

# And one to bind them
all.cov <- rbind(pres.cov, abs.cov)
head(all.cov)
#>       bio1       bio12       bio13       bio14     bio15     bio2     bio5
#> 1 29.37254 0.006426269 0.008964164 0.004170780 0.1474124 23.18576 47.05830
#> 2 30.45862 0.006053756 0.009221982 0.003706693 0.1708778 23.47737 47.67215
#> 3 31.87390 0.009033033 0.011659561 0.006434741 0.1648283 22.30462 47.77390
#> 4 31.69130 0.008533484 0.014504205 0.003622229 0.3815936 21.20000 46.98304
#> 5 30.94236 0.008057307 0.015355557 0.002832554 0.4453886 20.80000 46.38309
#> 6 31.70000 0.008724357 0.014759392 0.003717114 0.3831273 21.15962 47.04038
#>        bio6  crop   ndvi.amp   ndvi.mean tick
#> 1  8.030848 0.000 0.04955071 -0.01506902    1
```

```
#> 2  8.694777 0.000 0.05624679  0.01562384    1
#> 3 11.260820 0.000 0.10457005  0.09483840    1
#> 4 14.683043 0.031 0.03216948  0.12690221    1
#> 5 12.383087 0.011 0.01388101  0.12155575    1
#> 6 14.900000 0.044 0.15113069 -0.03903103    1

# Let's just clean it up a little bit
all.cov <- all.cov[complete.cases(all.cov),]
```

Now we have a dataset ready to model.

Running models with dbarts

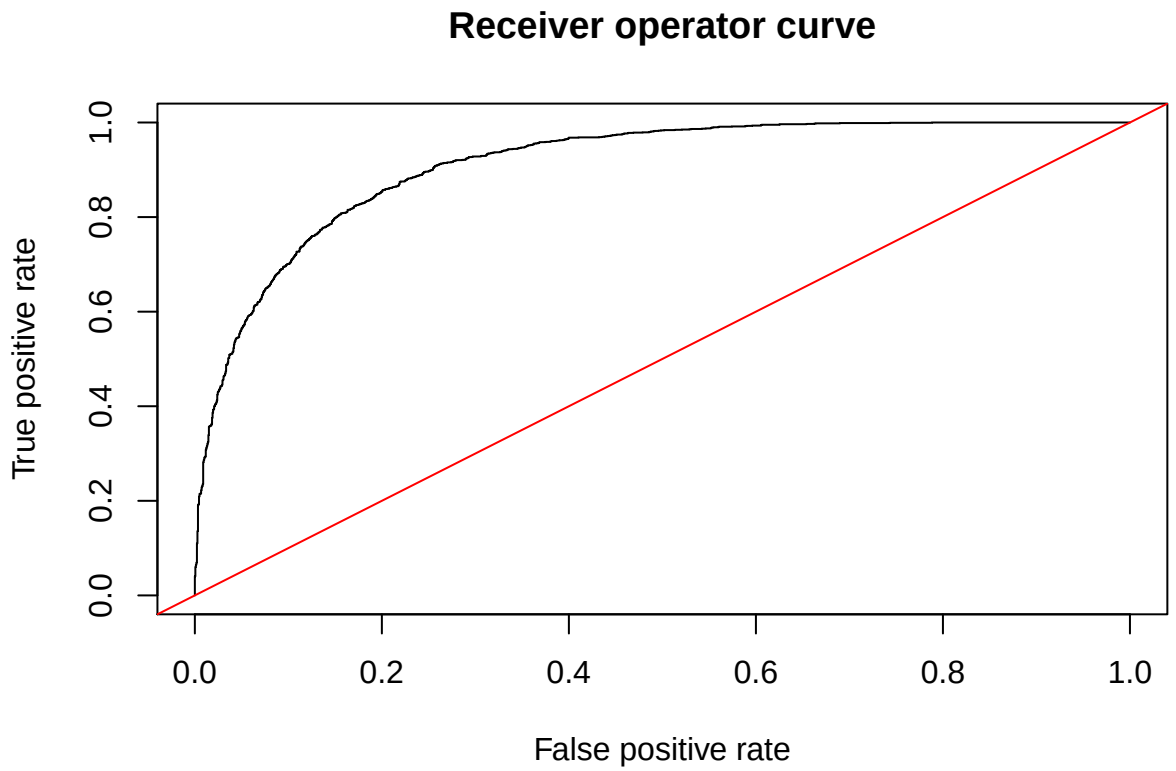
We could try something really simple on defaults, right out the gate. The *bart* function in *dbarts* can just be run on defaults:

```
first.model <- bart(all.cov[,1:11], all.cov[,'tick'], keeptrees=TRUE)
#>
#> Running BART with binary y
#>
#> number of trees: 200
#> number of chains: 1, number of threads 1
#> Prior:
#>  k: 2.000000
#>  power and base for tree prior: 2.000000 0.950000
#>  use quantiles for rule cut points: false
#> data:
#>  number of training observations: 3478
#>  number of test observations: 0
#>  number of explanatory variables: 11
#>
#> 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)
#> (11: 100)
#>
#> offsets:
#>  reg : 0.00 0.00 0.00 0.00 0.00
#> 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: 13.605066
#>
#> Tree sizes, last iteration:
```

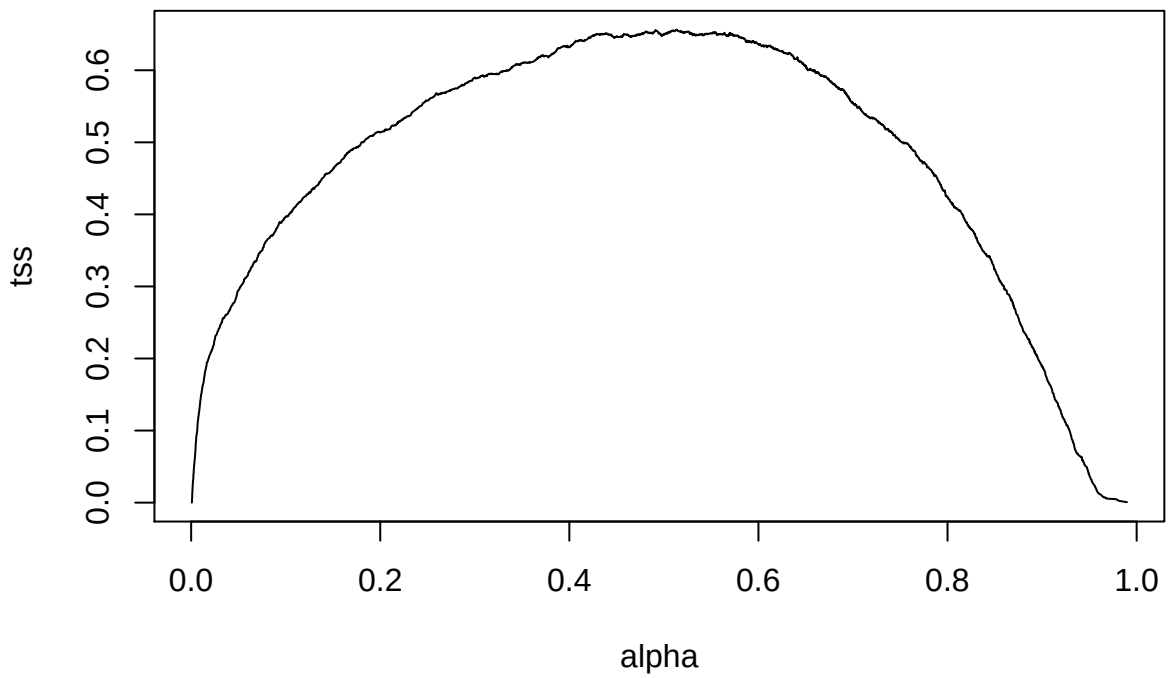
```
#> [1] 2 2 2 2 3 3 2 3 2 2 2 2 3 3 4 3 3 2
#> 3 3 4 1 4 2 2 3 2 4 3 5 5 2 3 2 3 2 3 1
#> 3 2 1 3 2 2 3 3 3 3 3 4 2 2 2 3 2 2 2 2
#> 2 3 1 2 2 4 2 2 2 2 1 2 3 3 2 3 2 3 2 2
#> 3 2 2 1 2 2 3 2 3 2 4 2 2 2 2 2 3 1 2 3
#> 4 1 3 2 1 2 2 2 3 2 2 2 3 2 4 2 2 2 3 4
#> 3 2 2 2 2 3 2 3 2 4 2 3 2 4 2 2 3 3 2 2
#> 2 2 2 3 2 3 2 1 2 2 3 2 3 2 2 4 3 2 3 4
#> 2 3 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 4
#> 3 2 2 3 2 2 3 2 2 2 3 3 3 3 3 2 2 2 3 1
#> 2 3
#>
#> Variable Usage, last iteration (var:count):
#> (1: 32) (2: 30) (3: 24) (4: 22) (5: 34)
#> (6: 18) (7: 27) (8: 30) (9: 22) (10: 18)
#> (11: 28)
#> DONE BART
```

That's well and good, but dbarts doesn't have great tools to evaluate what the models do, or how they're working as predictive tools. Plus, we can't see the spatial prediction, which makes it hard to know if it's even looking plausible. One quick trick is to use the *bart.auc* function in *embarcadero*, which uses the *dbarts* model object (or an *embarcadero* model object) and the vector of true data.

```
bart.auc(first.model, all.cov[,'tick'])
#> [1] "AUC = "
#> [1] 0.9108574
```



```
#> Press [enter] to continue
```

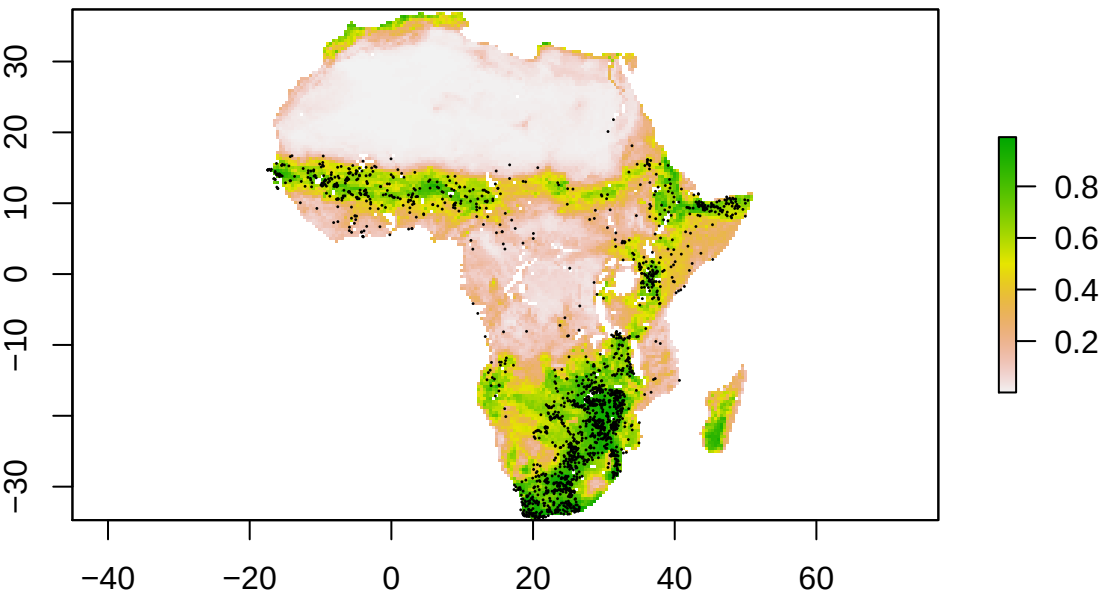



```
#> [1] "TSS threshold"
#> [1] 0.5132467
#> [1] "Type I error rate"
#> [1] 0.1425178
#> [1] "Type II error rate"
#> [1] 0.2012263
```

A high AUC value indicates our model performs well. The AUC function also returns an optimal threshold that maximizes the true skill statistic (TSS), and the sensitivity/specificity of the model at that cutoff (alpha).

What do the predictions look like? To make a predicted raster, we have to use *embarcadero*'s wrapper for the native *predict* function in *dbarts*.

```
pred.prelim <- predict.dbart.raster(model = first.model,
                                    inputstack = cov.big)
plot(pred.prelim)

points(SpatialPoints(ticks[,c('Longitude.X','Latitude.Y')]),
       pch=16, cex=0.2)
```



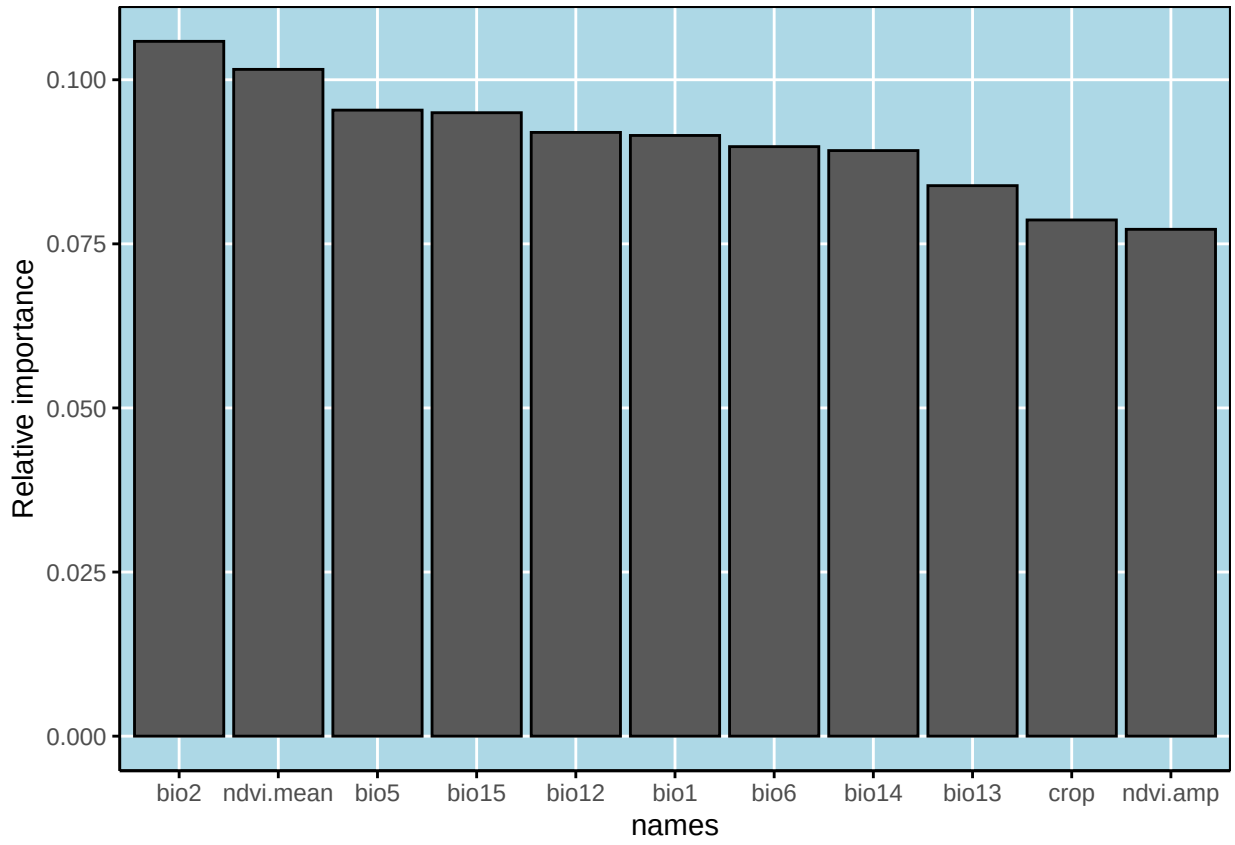
This model seems okay. We're getting predictions in places we don't have any records, like North Africa. That could be good if we think that's suitable climatic space (and if you know Hyalomma, you know there's definitely some species there, though posibly not truncatum), but with much of the inhabited area not being predicted, let's revisit that later.

Variable Selection

Next, let's try some automated variable selection. There's a few different component pieces that do this in *embarcadero*.

First, let's look at the variable contributions in the existing model:

```
varimp(model=first.model,
       names=names(covs),
       plots = TRUE)
```




```
#>        names    varimps
#> 1       bio1 0.09151130
#> 2      bio12 0.09197496
#> 3      bio13 0.08386658
#> 4      bio14 0.08920886
#> 5      bio15 0.09497791
#> 6       bio2 0.10585038
#> 7       bio5 0.09537216
#> 8       bio6 0.08981008
#> 9       crop 0.07863812
#> 10  ndvi.amp 0.07721162
#> 11 ndvi.mean 0.10157804
```

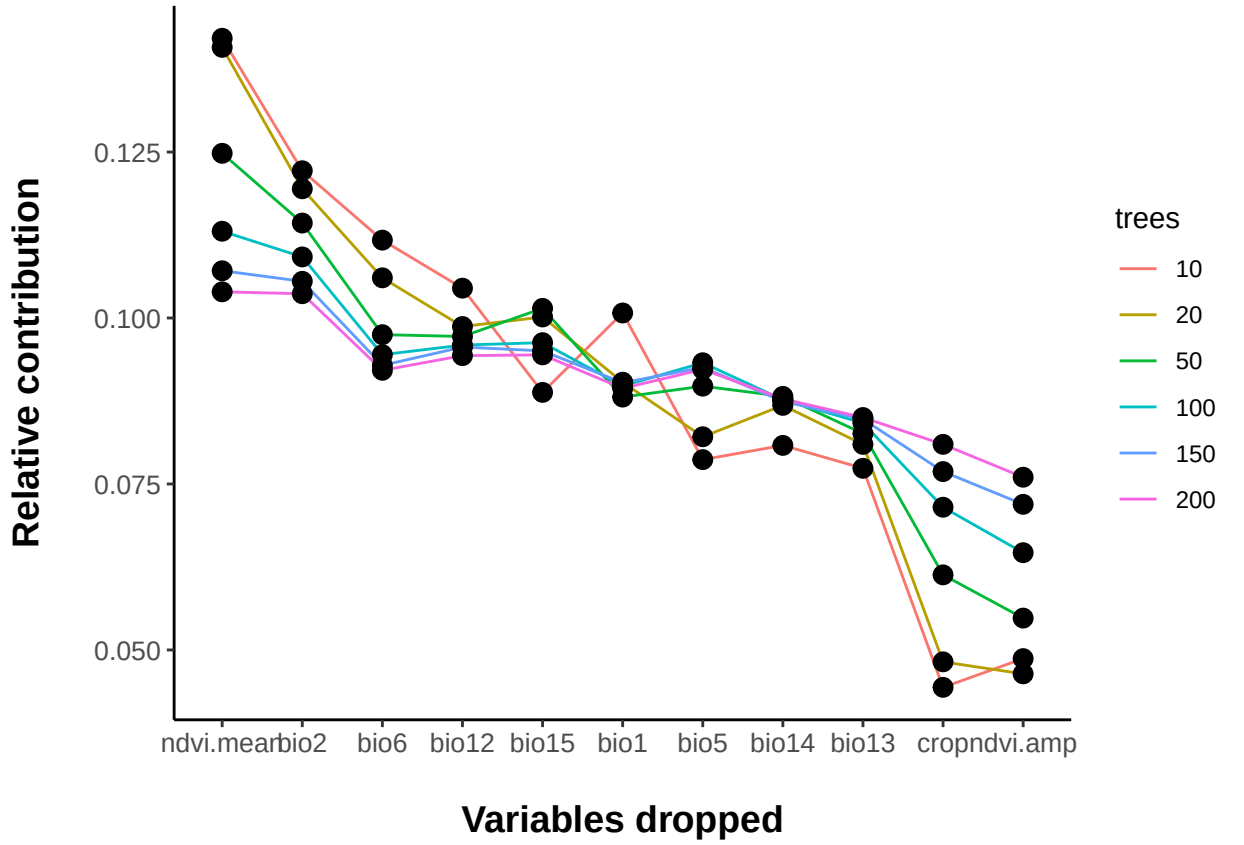
This tells us roughly how the variables contribute so far, but it doesn't tell us who to eliminate first - we don't want to eliminate based on a single run.

Previously, it's been suggested that the best variable diagnostic for BART is to run models with progressively smaller numbers of trees - and as you get down to 10 or 20 trees per model, the contributions of bad or irrelevant variables will drop out. This is because like most CART methods, BART has the ability to overfit on variables with low information content.

What *varimp.plot* does is run variable importance for hundreds of models at different tree levels. Let's say 10 models per combination is enough. This will print each level of models it's run, and then it will make a plot that shows us variable importance across runs.

```
varimp.plot(x.data=all.cov[,1:11],
            y.data=all.cov[,'tick'],
            iter=100)
```

```
#> [1] 10
#> [1] 20
#> [1] 50
#> [1] 100
#> [1] 150
#> [1] 200
```



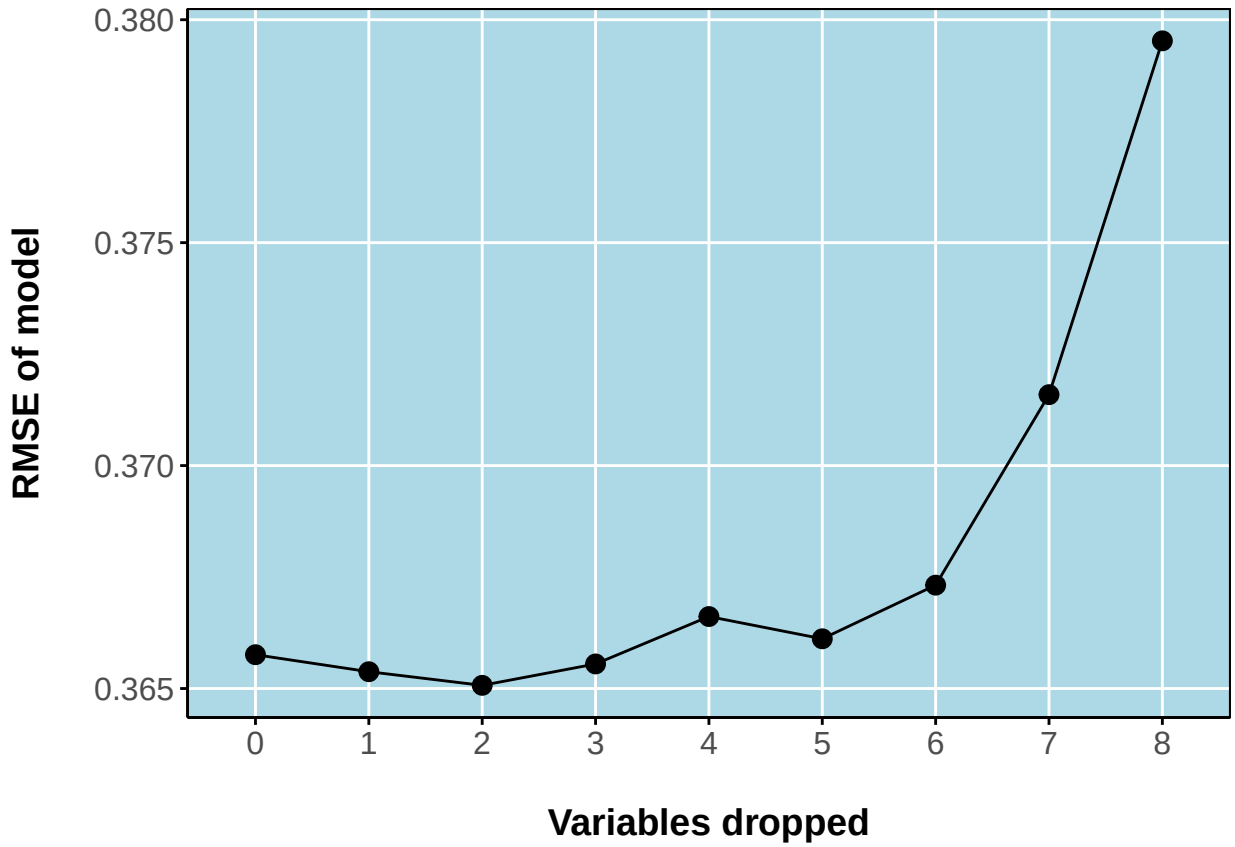
Out of all the variables, "crop" seems to be especially undesirable as a predictor. A few other variables seem like they might not be helping the model either, but we probably need a systematic way to deal with that.

Automated stepwise reduction isn't the best way to do things in machine learning, but it's consistent over a high number of iterations, and is the current stopgap in the package. *variable.step* will automate the process, starting with the full feature set, fitting *iter* models with *n.trees* each (use a small value - 10 or 20), and reducing stepwise based on the variable with the lowest importance each iteration. Then, it'll make a recommendation for a feature set based on root mean square error (RMSE). That's not perfect, and you can take or leave it as an approach.

Expert knowledge about variable importance and cautious inclusion will *always* be better, epistemologically, than automated stepwise feature set reduction.

```
varlist <- variable.step(x.data=all.cov[,1:11],
              y.data=all.cov[,'tick'],
              n.trees=10)
#> [1] Number of variables included: 11
#> [1] Dropped:
#> [1]
#> [1] ---------------------------------------
```

```
#> [1] Number of variables included: 10
#> [1] Dropped:
#> [1] crop
#> [1] ---------------------------------------
#> [1] Number of variables included: 9
#> [1] Dropped:
#> [1] crop     ndvi.amp
#> [1] ---------------------------------------
#> [1] Number of variables included: 8
#> [1] Dropped:
#> [1] crop     ndvi.amp bio5
#> [1] ---------------------------------------
#> [1] Number of variables included: 7
#> [1] Dropped:
#> [1] crop     ndvi.amp bio5     bio14
#> [1] ---------------------------------------
#> [1] Number of variables included: 6
#> [1] Dropped:
#> [1] crop     ndvi.amp bio5     bio14    bio13
#> [1] ---------------------------------------
#> [1] Number of variables included: 5
#> [1] Dropped:
#> [1] crop     ndvi.amp bio5     bio14    bio13    bio6
#> [1] ---------------------------------------
#> [1] Number of variables included: 4
#> [1] Dropped:
#> [1] crop     ndvi.amp bio5     bio14    bio13    bio6     bio15
#> [1] ---------------------------------------
#> [1] Number of variables included: 3
#> [1] Dropped:
#> [1] crop     ndvi.amp bio5     bio14    bio13    bio6     bio15    bio2
#> [1] ---------------------------------------
```

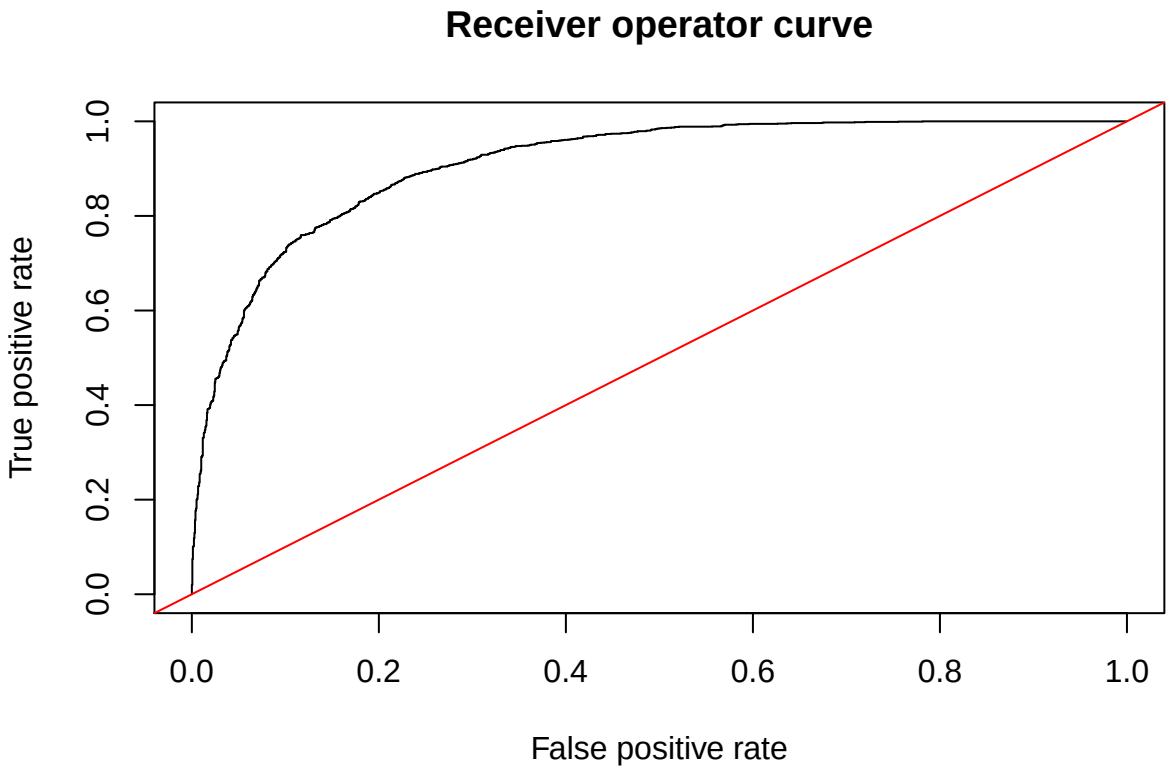



```
#> [1] ---------------------------------------
#> [1] Final recommended variable list
#> [1] bio1      bio12     bio13     bio14     bio15     bio2      bio5
#> [8] bio6      ndvi.mean
```

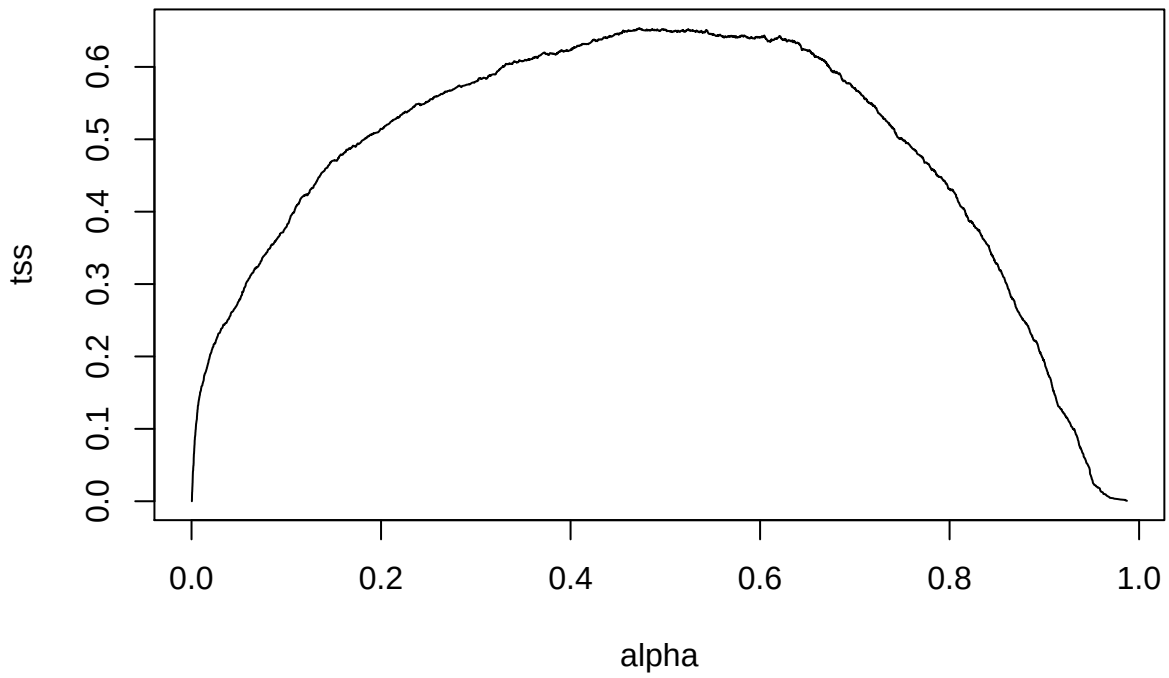
Our stepwise reduction only cuts crop - that's great. Normally this step cuts a few variables - it's probably a good sign about our *a priori* variable selection that not much got dropped. Let's run a "good" model with that predictor cut.

```
# Rerun the model
good.model <- bart(all.cov[,varlist], all.cov[,'tick'], keeptrees=TRUE)
#>
#> Running BART with binary y
#>
#> number of trees: 200
#> number of chains: 1, number of threads 1
#> Prior:
#>  k: 2.000000
#>  power and base for tree prior: 2.000000 0.950000
#>  use quantiles for rule cut points: false
#> data:
#>  number of training observations: 3478
#>  number of test observations: 0
#>  number of explanatory variables: 9
#>
#> Cutoff rules c in x<=c vs x>c
#> Number of cutoffs: (var: number of possible c):
```

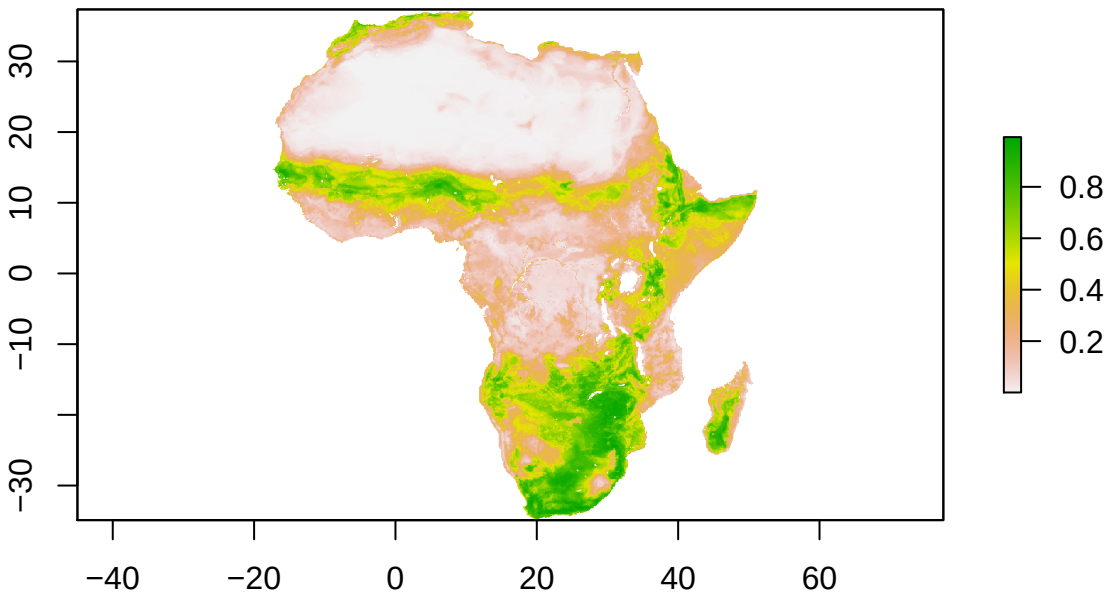
```
#> (1: 100) (2: 100) (3: 100) (4: 100) (5: 100)
#> (6: 100) (7: 100) (8: 100) (9: 100)
#>
#> offsets:
#>  reg : 0.00 0.00 0.00 0.00 0.00
#> Running mcmc loop:
#> iteration: 100 (of 1000)
#> iteration: 200 (of 1000)
#> iteration: 300 (of 1000)
#> iteration: 400 (of 1000)
#> iteration: 500 (of 1000)
#> iteration: 600 (of 1000)
#> iteration: 700 (of 1000)
#> iteration: 800 (of 1000)
#> iteration: 900 (of 1000)
#> iteration: 1000 (of 1000)
#> total seconds in loop: 7.169468
#>
#> Tree sizes, last iteration:
#> [1] 3 2 2 2 2 3 2 2 3 2 3 3 4 1 1 2 3 2
#> 2 2 3 3 2 2 2 2 3 4 2 2 2 4 6 4 4 3 3 2
#> 4 3 3 3 2 3 3 2 3 2 1 3 3 2 2 3 2 3 2 3
#> 3 3 3 2 2 3 2 2 2 3 2 3 4 2 3 2 3 3 2 2
#> 4 2 2 3 2 2 2 1 2 1 2 3 2 3 3 1 3 5 2 3
#> 2 5 2 3 2 2 2 2 2 3 2 2 2 2 3 1 2 2 2 5 2
#> 2 2 2 2 1 2 1 2 4 4 2 3 2 1 1 2 2 2 3 2
#> 3 2 2 2 3 5 2 3 3 2 4 1 2 2 2 3 2 2 3 1
#> 2 4 4 5 2 2 2 2 2 3 3 2 4 2 2 2 2 5 2 2
#> 2 2 2 3 3 3 3 2 2 2 4 3 2 2 2 2 2 3 2 3
#> 3 3
#>
#> Variable Usage, last iteration (var:count):
#> (1: 31) (2: 39) (3: 23) (4: 34) (5: 24)
#> (6: 40) (7: 36) (8: 39) (9: 32)
#> DONE BART
# Check the AUC
bart.auc(good.model, all.cov[,'tick'])
#> [1] "AUC = "
#> [1] 0.9114245
```




```
#> Press [enter] to continue
```




```
#> [1] "TSS threshold"
#> [1] 0.4722557
#> [1] "Type I error rate"
#> [1] 0.1187648
#> [1] "Type II error rate"
#> [1] 0.2279822
# Do the spatial prediction
hytr.layer <- predict.dbart.raster(model = good.model,
                                   inputstack = covs[[varlist]])
# How's it look?
plot(hytr.layer)
```



Mapping CCHF

Alright. Now let's get back to business by building the CCHF map. We're going to use the same predictors as we used for *H. truncatum* plus the suitability layer.

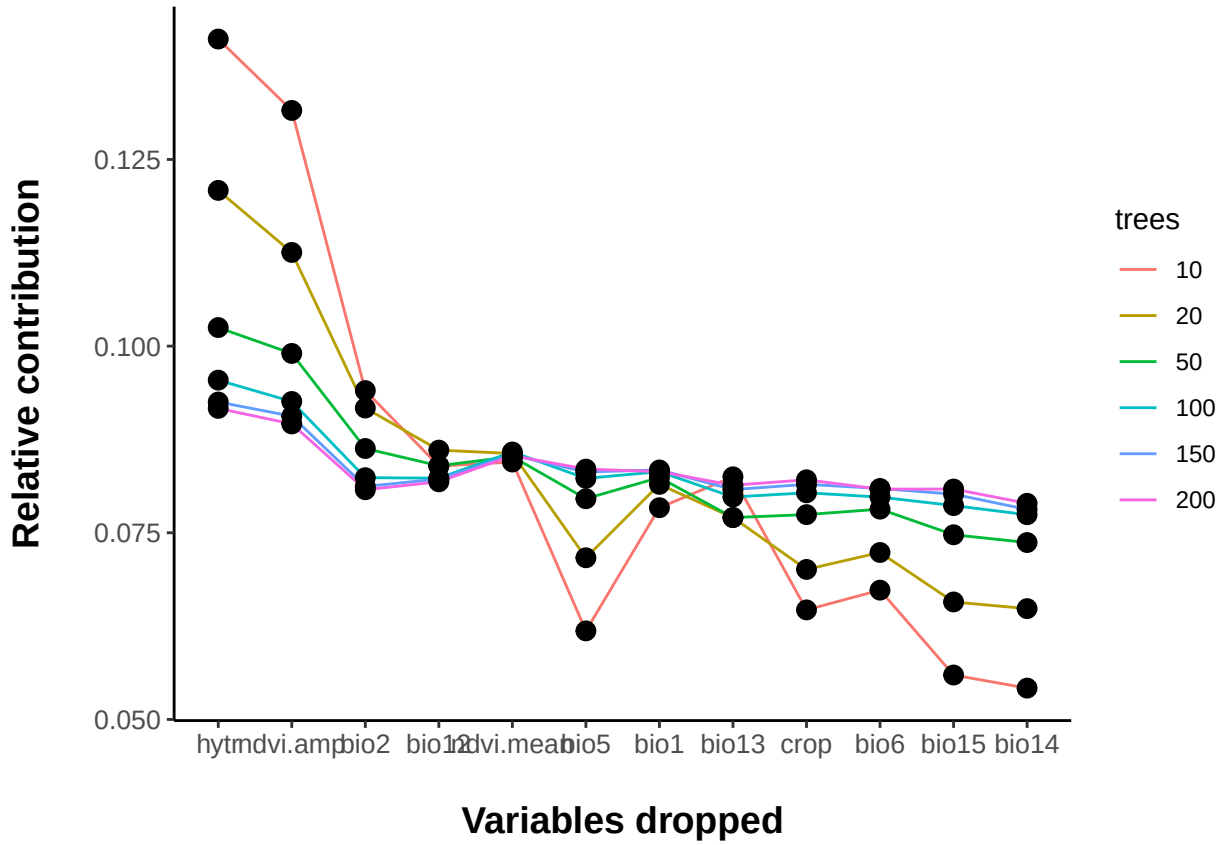
Running the CCHF model

This time, let's just do the variable selection up front. And instead of running each piece separately, we can run a full-service model using *bart.var* from *embarcadero*. That runs each of the steps we did above, including running the reduced-feature model, and returns a list object with a model and a variable set.

```
# Update those pesky covariates
covs <- stack(covs, hytr.layer)
names(covs)[12]='hytr'

# Read in the data
cchf <- read.csv('~/Github/embarcadero/vignettes/CCHF_1953_2012_Messina.csv')
head(cchf)
#>   OCCURRENCE_ID LOCATION_TYPE ADMIN_LEVEL GAUL_AD1 GAUL_AD2
#> 1             1         point        -999     1282    16397
#> 2             2         point        -999     1282    16397
#> 3             3         point        -999     1282    16397
#> 4             4         point        -999     1278    16376
```

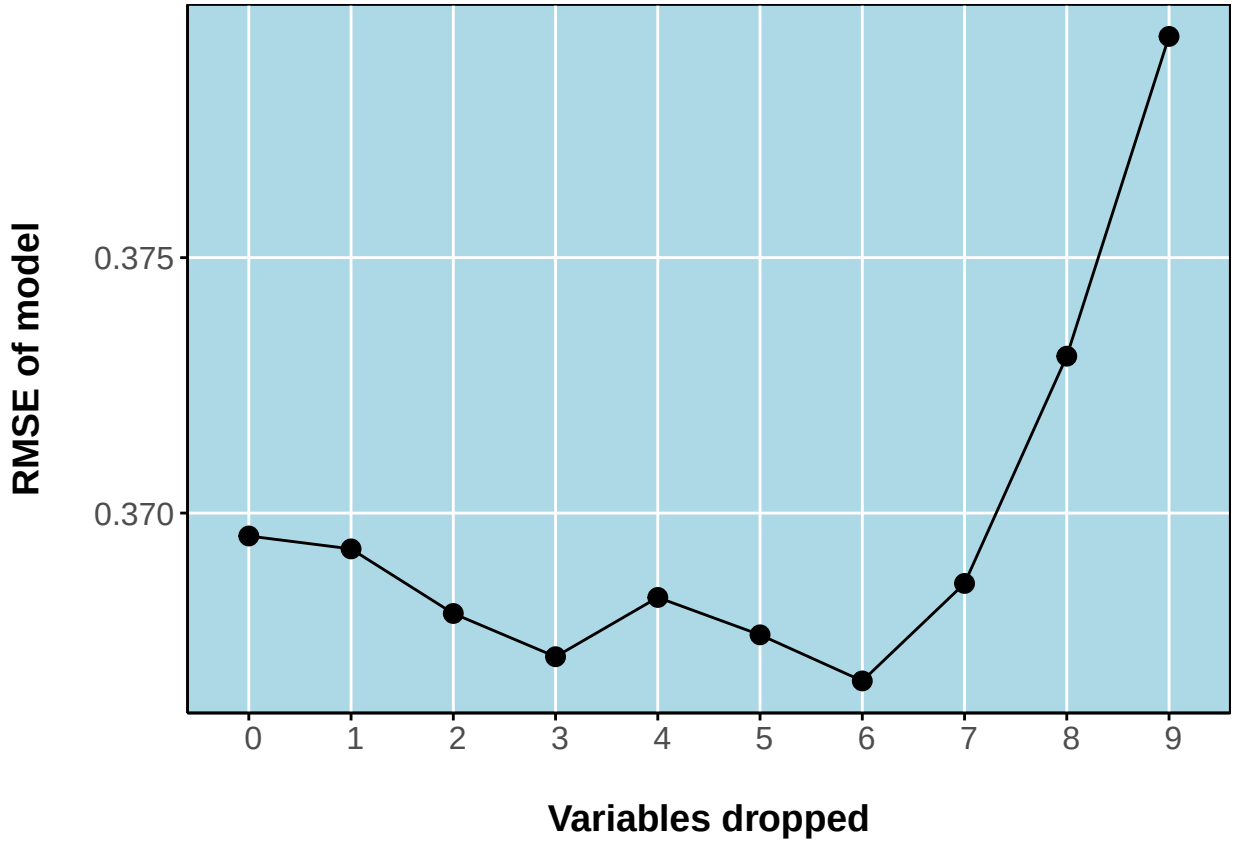
```
#> 5             5         point        -999     1278    16376
#> 6             6         point        -999     1278    16376
#>   UNIQUE_LOCATION YEAR LATITUDE LONGITUDE    COUNTRY REGION
#> 1             535 1953  38.0944   69.3321 Tajikistan   Asia
#> 2            1178 1953  37.6570   69.6272 Tajikistan   Asia
#> 3             620 1954  42.4129   20.7944     Serbia   Asia
#> 4            1182 1954  37.2350   69.0988 Tajikistan   Asia
#> 5            1165 1954  37.4917   69.4029 Tajikistan   Asia
#> 6            1178 1954  37.6570   69.6272 Tajikistan   Asia
nrow(cchf)
#> [1] 1721

# Spatial thinning checks; this also limits it to African points
cchf <- cchf[,c('LONGITUDE','LATITUDE')]; cchf$Presence = 1
cchf <- SpatialPointsDataFrame(cchf[,1:2],data.frame(Presence=cchf[,3]))
tmp=rasterize(cchf, covs[[1]], field="Presence", fun="min")
pts.sp1=rasterToPoints(tmp, fun=function(x){x>0})
nrow(pts.sp1)
#> [1] 147

# Extract presence values
pres.cov <- raster::extract(covs, pts.sp1[,1:2])
pres.cov <- na.omit(pres.cov)
head(pres.cov)
#>          bio1       bio12      bio13       bio14     bio15     bio2
#> [1,] 31.40000 0.007572869 0.01082603 0.004382429 0.2038342 23.12232
#> [2,] 31.49514 0.007635116 0.01092475 0.004430649 0.2060203 23.20000
#> [3,] 28.84297 0.010640300 0.01573716 0.006981144 0.2984491 14.74204
#> [4,] 30.74783 0.009813542 0.01538886 0.005333871 0.3309597 20.64350
#> [5,] 31.60000 0.008458333 0.01432100 0.003642234 0.3755587 21.21700
#> [6,] 31.40000 0.008174079 0.01485319 0.003165020 0.4296599 20.50000
#>          bio5      bio6  crop   ndvi.amp  ndvi.mean       hytr
#> [1,] 47.89513  9.859759 0.046 0.18322578 0.03276619 0.04027415
#> [2,] 47.90000  9.895139 0.009 0.10102680 0.02260730 0.05302135
#> [3,] 40.83361 16.904684 0.000 0.03136669 0.06945100 0.22671964
#> [4,] 45.59567 14.652166 0.036 0.01889407 0.13301405 0.36604065
#> [5,] 46.84039 14.483000 0.002 0.02256186 0.12429252 0.10653951
#> [6,] 46.68309 13.300000 0.048 0.02511384 0.13301110 0.15971777

#Generate pseudoabsences
absence <- randomPoints(covs,nrow(pres.cov))
#> Warning in couldBeLonLat(mask): CRS is NA. Assuming it is longitude/
#> latitude
abs.cov <- raster::extract(covs, absence)

#Code the response
pres.cov <- data.frame(pres.cov); pres.cov$cchf <- 1
abs.cov <- data.frame(abs.cov); abs.cov$cchf <- 0

# And one to bind them
all.cov <- rbind(pres.cov, abs.cov)
all.cov <- all.cov[complete.cases(all.cov),]; nrow(all.cov)
#> [1] 184
```

```
head(all.cov)
#>       bio1       bio12      bio13       bio14     bio15     bio2     bio5
#> 1 31.40000 0.007572869 0.01082603 0.004382429 0.2038342 23.12232 47.89513
#> 2 31.49514 0.007635116 0.01092475 0.004430649 0.2060203 23.20000 47.90000
#> 3 28.84297 0.010640300 0.01573716 0.006981144 0.2984491 14.74204 40.83361
#> 4 30.74783 0.009813542 0.01538886 0.005333871 0.3309597 20.64350 45.59567
#> 5 31.60000 0.008458333 0.01432100 0.003642234 0.3755587 21.21700 46.84039
#> 6 31.40000 0.008174079 0.01485319 0.003165020 0.4296599 20.50000 46.68309
#>        bio6  crop   ndvi.amp  ndvi.mean       hytr cchf
#> 1  9.859759 0.046 0.18322578 0.03276619 0.04027415    1
#> 2  9.895139 0.009 0.10102680 0.02260730 0.05302135    1
#> 3 16.904684 0.000 0.03136669 0.06945100 0.22671964    1
#> 4 14.652166 0.036 0.01889407 0.13301405 0.36604065    1
#> 5 14.483000 0.002 0.02256186 0.12429252 0.10653951    1
#> 6 13.300000 0.048 0.02511384 0.13301110 0.15971777    1

# This part automates the variable selection and returns the model
cchf.model <- bart.var(xdata=all.cov[,1:12],
                       ydata=all.cov['cchf'],
                       iter.step = 100,
                       tree.step = 10,
                       iter.plot = 100)
#> [1] 10
#> [1] 20
#> [1] 50
#> [1] 100
#> [1] 150
#> [1] 200
```



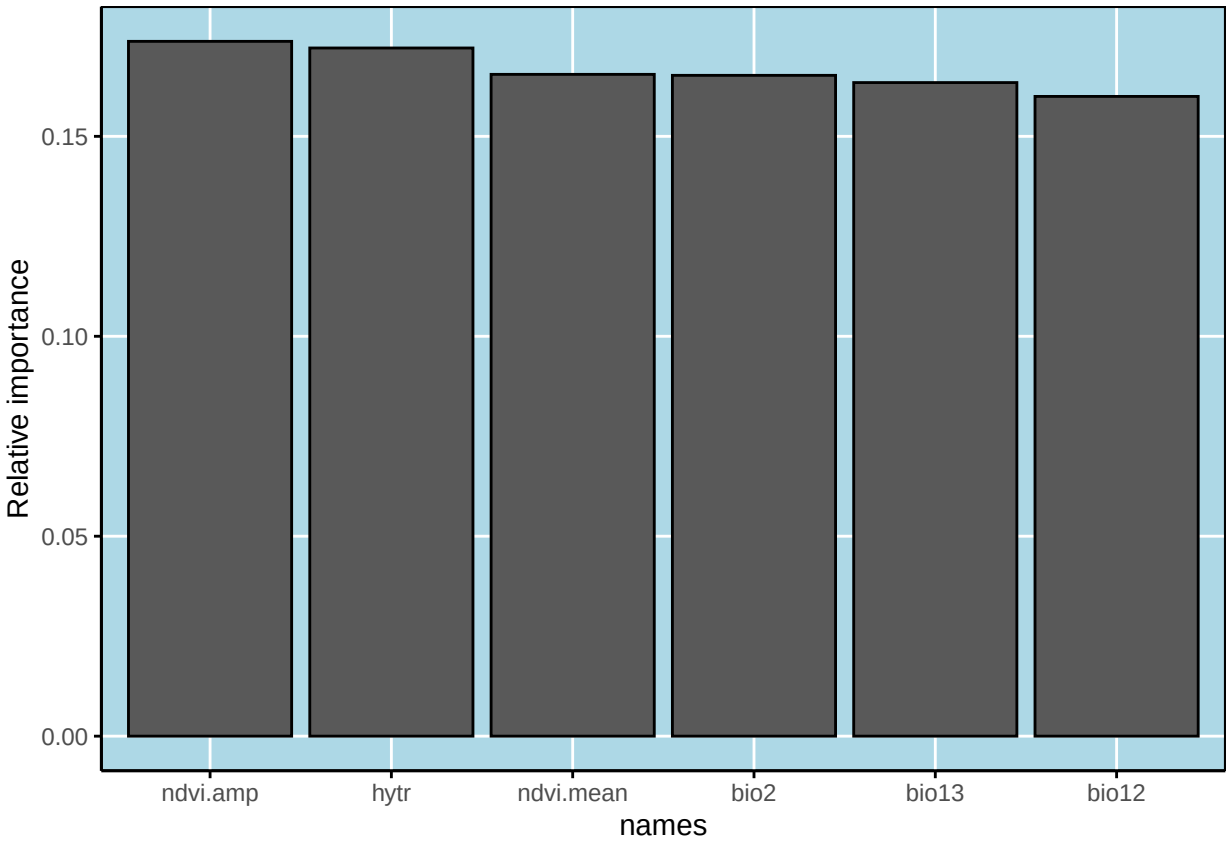

```
#> [1] Number of variables included: 12
#> [1] Dropped:
#> [1]
#> [1] ---------------------------------------
#> [1] Number of variables included: 11
#> [1] Dropped:
#> [1] bio15
#> [1] ---------------------------------------
#> [1] Number of variables included: 10
#> [1] Dropped:
#> [1] bio15 bio14
#> [1] ---------------------------------------
#> [1] Number of variables included: 9
#> [1] Dropped:
#> [1] bio15 bio14 bio5
#> [1] ---------------------------------------
#> [1] Number of variables included: 8
#> [1] Dropped:
#> [1] bio15 bio14 bio5  crop
#> [1] ---------------------------------------
#> [1] Number of variables included: 7
#> [1] Dropped:
#> [1] bio15 bio14 bio5  crop  bio6
#> [1] ---------------------------------------
#> [1] Number of variables included: 6
#> [1] Dropped:
```

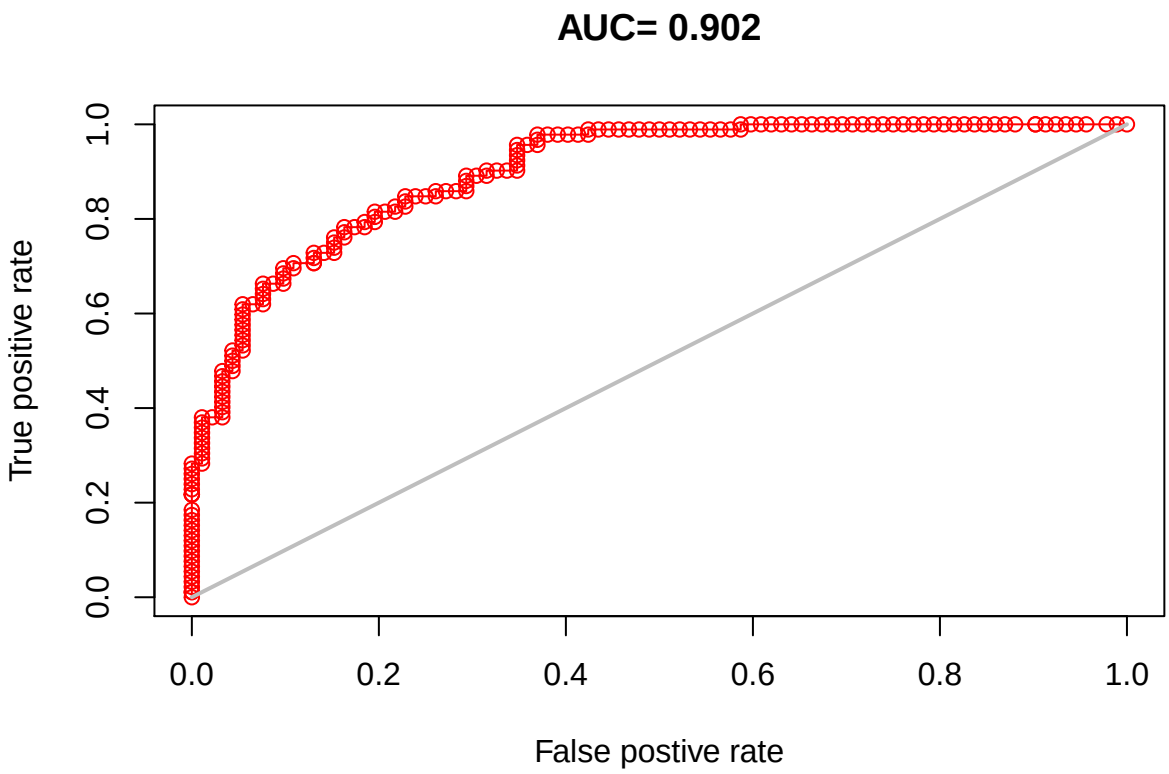
```
#> [1] bio15 bio14 bio5  crop  bio6  bio1
#> [1] ---------------------------------------
#> [1] Number of variables included: 5
#> [1] Dropped:
#> [1] bio15     bio14     bio5      crop      bio6      bio1      ndvi.mean
#> [1] ---------------------------------------
#> [1] Number of variables included: 4
#> [1] Dropped:
#> [1] bio15     bio14     bio5      crop      bio6      bio1      ndvi.mean
#> [8] bio13
#> [1] ---------------------------------------
#> [1] Number of variables included: 3
#> [1] Dropped:
#> [1] bio15     bio14     bio5      crop      bio6      bio1      ndvi.mean
#> [8] bio13     bio12
#> [1] ---------------------------------------
```



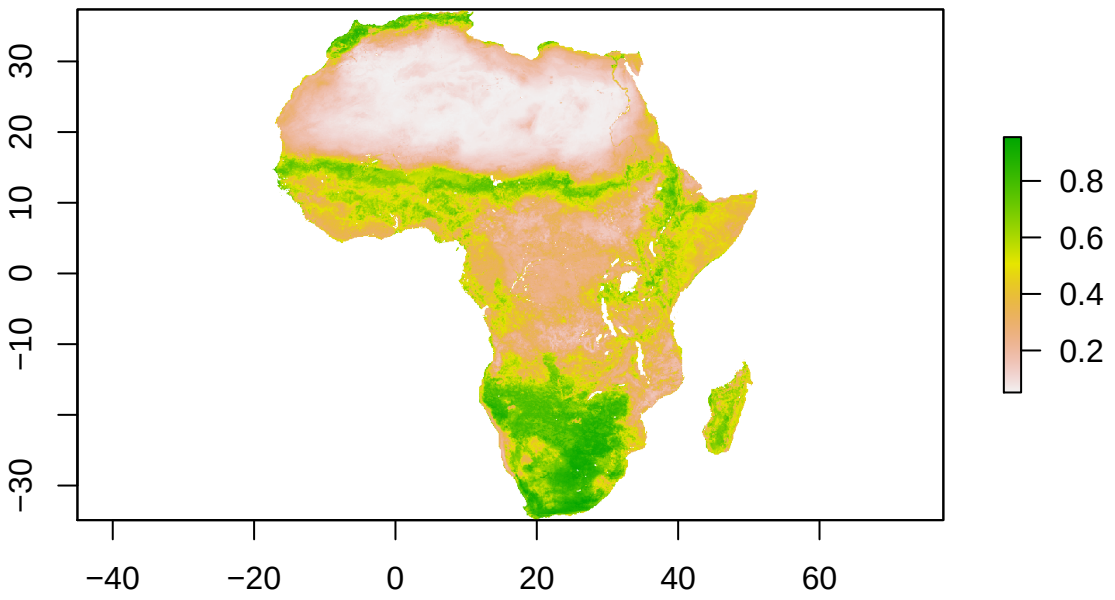

```
#> [1] ---------------------------------------
#> [1] Final recommended variable list
#> [1] bio12     bio13     bio2      ndvi.amp  ndvi.mean hytr
#>
#> Running BART with binary y
#>
#> number of trees: 200
#> number of chains: 1, number of threads 1
#> Prior:
#>  k: 2.000000
```

```
#>  power and base for tree prior: 2.000000 0.950000
#>  use quantiles for rule cut points: false
#> data:
#>  number of training observations: 184
#>  number of test observations: 0
#>  number of explanatory variables: 6
#>
#> 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)
#>
#> offsets:
#>  reg : 0.00 0.00 0.00 0.00 0.00
#> 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: 0.991744
#>
#> Tree sizes, last iteration:
#> [1] 3 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2
#> 2 2 2 2 2 3 3 2 1 2 3 4 2 2 3 3 2 2 6 3
#> 3 2 2 3 3 1 2 3 2 2 2 2 3 2 2 2 3 2 3 2
#> 3 3 2 2 3 1 4 2 2 3 2 5 3 2 2 3 3 2 3 2
#> 4 2 2 2 2 2 3 3 2 2 2 2 2 4 5 2 2 3 2 2
#> 3 2 2 2 2 1 2 3 2 2 3 1 2 2 4 2 2 3 2 2
#> 2 2 2 2 2 4 2 2 3 2 2 2 2 3 3 3 2 2 4 4
#> 2 2 3 2 2 2 2 2 2 2 4 2 1 2 3 2 2 2 2 2
#> 4 2 2 2 2 2 2 2 2 2 2 2 3 3 2 3 2 3 2 2
#> 3 2 3 3 3 2 3 2 3 2 2 2 2 3 3 2 1 4 2 2
#> 3 4
#>
#> Variable Usage, last iteration (var:count):
#> (1: 38) (2: 46) (3: 44) (4: 47) (5: 46)
#> (6: 55)
#> DONE BART
```

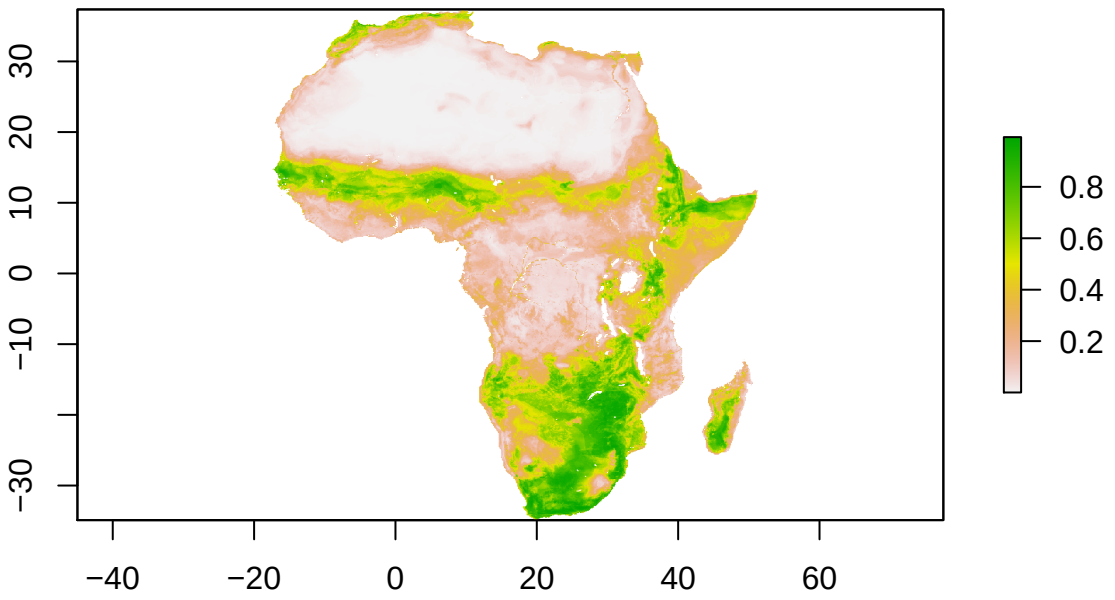

Relative importance
0.15
0.10
0.05
0.00
ndvi.amp
hytr
ndvi.mean
bio2
bio13
bio12
names



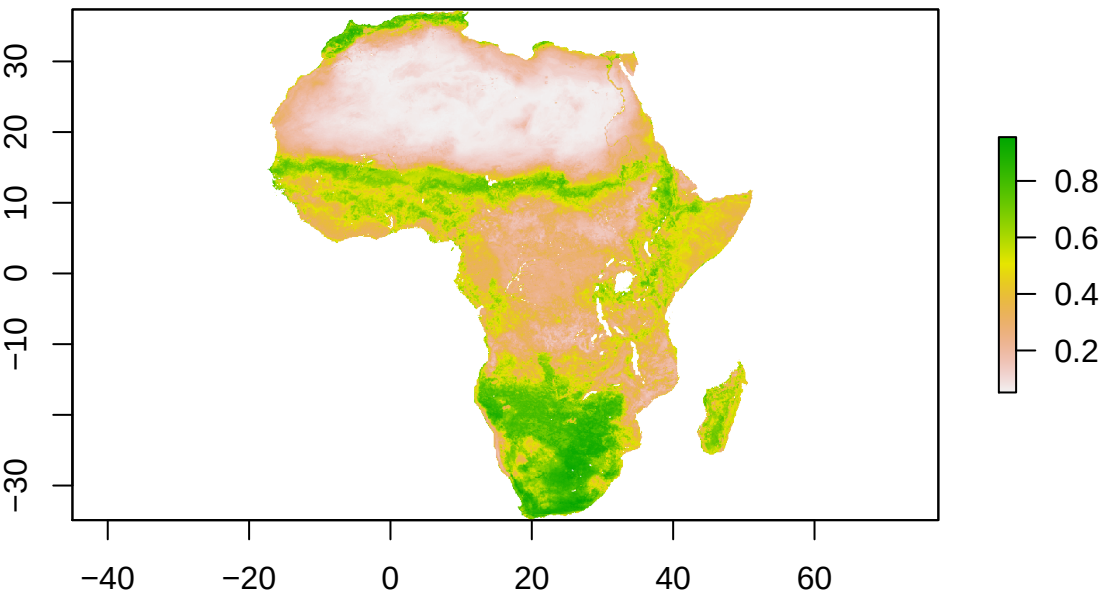

```
# Do the spatial prediction
cchf.map <- predict.dbart.raster(model = cchf.model$Model.object,
                                 inputstack = covs[[cchf.model$Variables]])
```




```
# How's it look?
plot(hytr.layer)
```




```
plot(cchf.map)
```

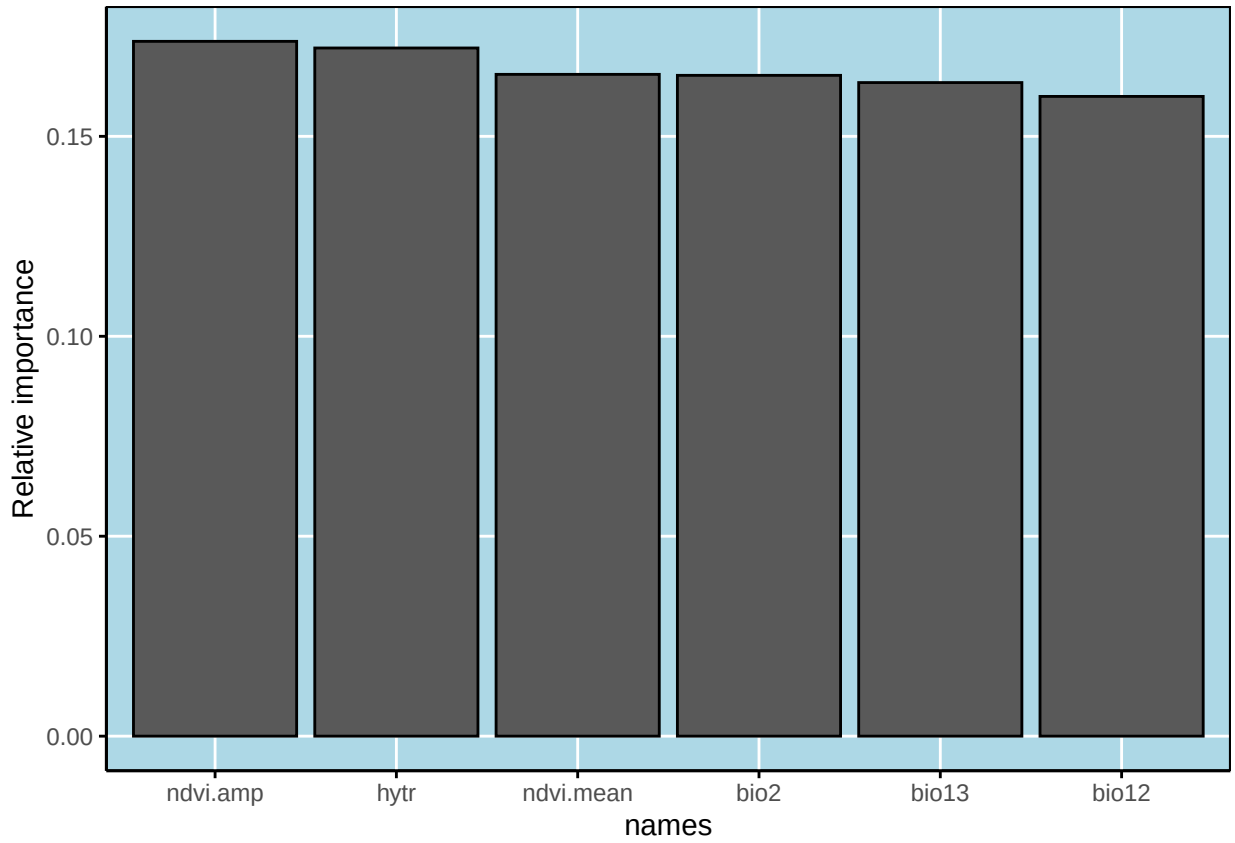


OK. Nice model! Let's see what we can do to unpack it.

Analytics

Let's look at the variable contributions:

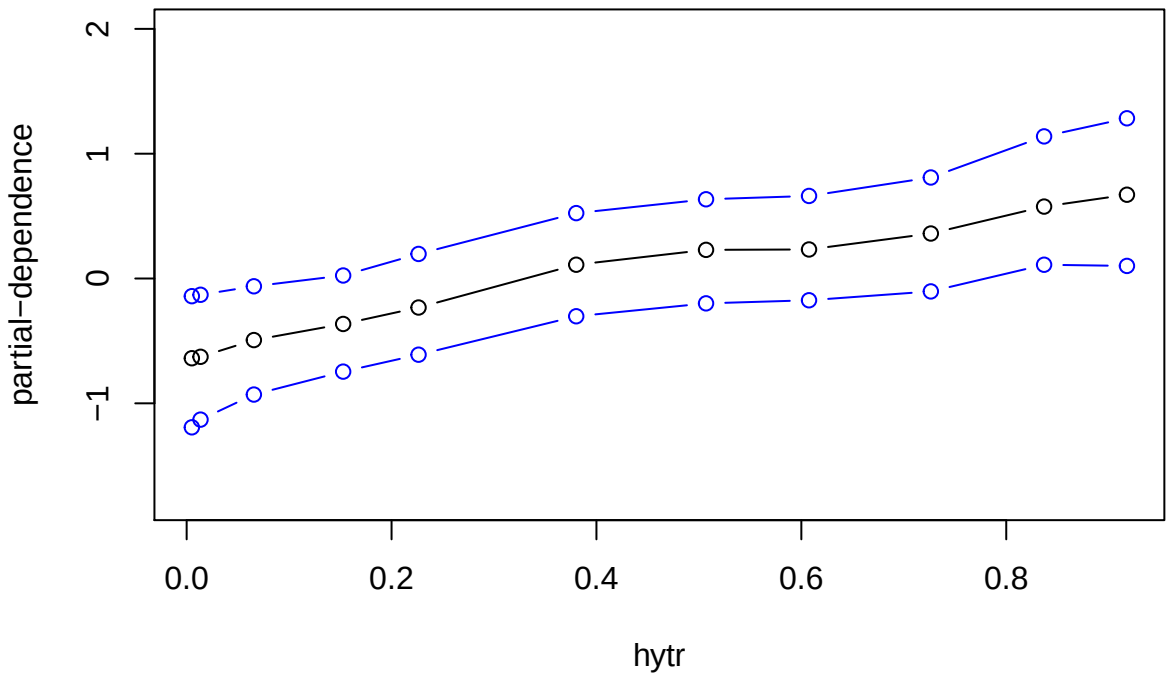
```
varimp(cchf.model$Model.object, cchf.model$Variables, plots=TRUE)
```



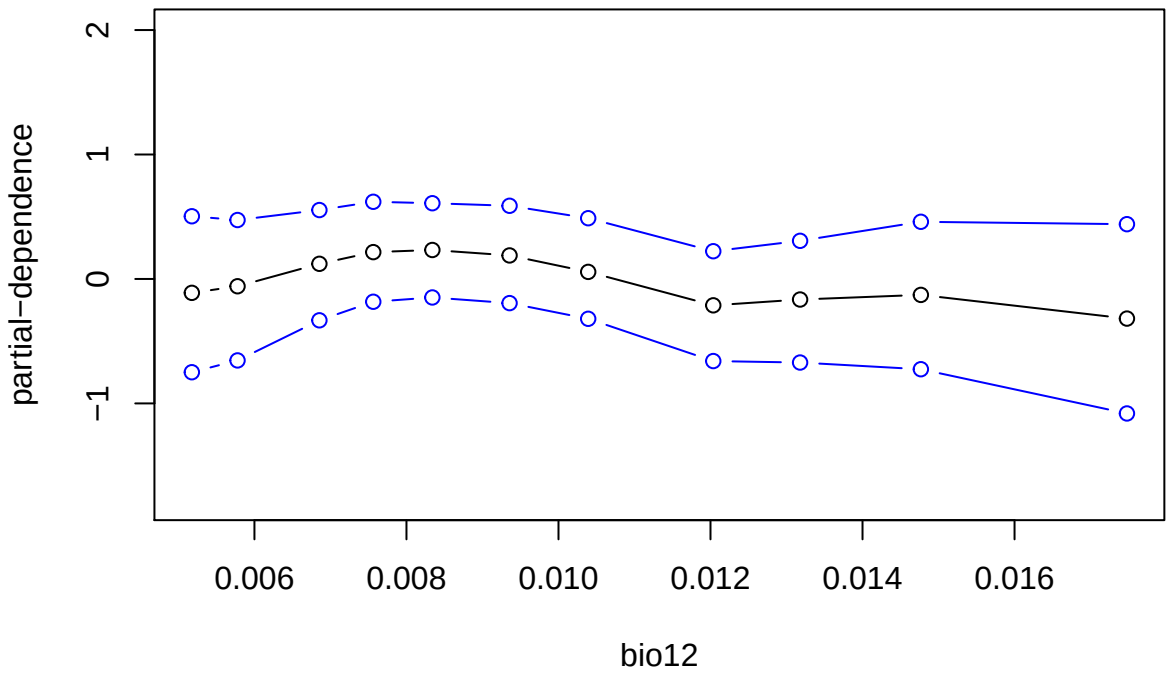

```
#>       names   varimps
#> 1     bio12 0.1599811
#> 2     bio13 0.1634399
#> 3      bio2 0.1652454
#> 4  ndvi.amp 0.1737535
#> 5 ndvi.mean 0.1654918
#> 6      hytr 0.1720883
```

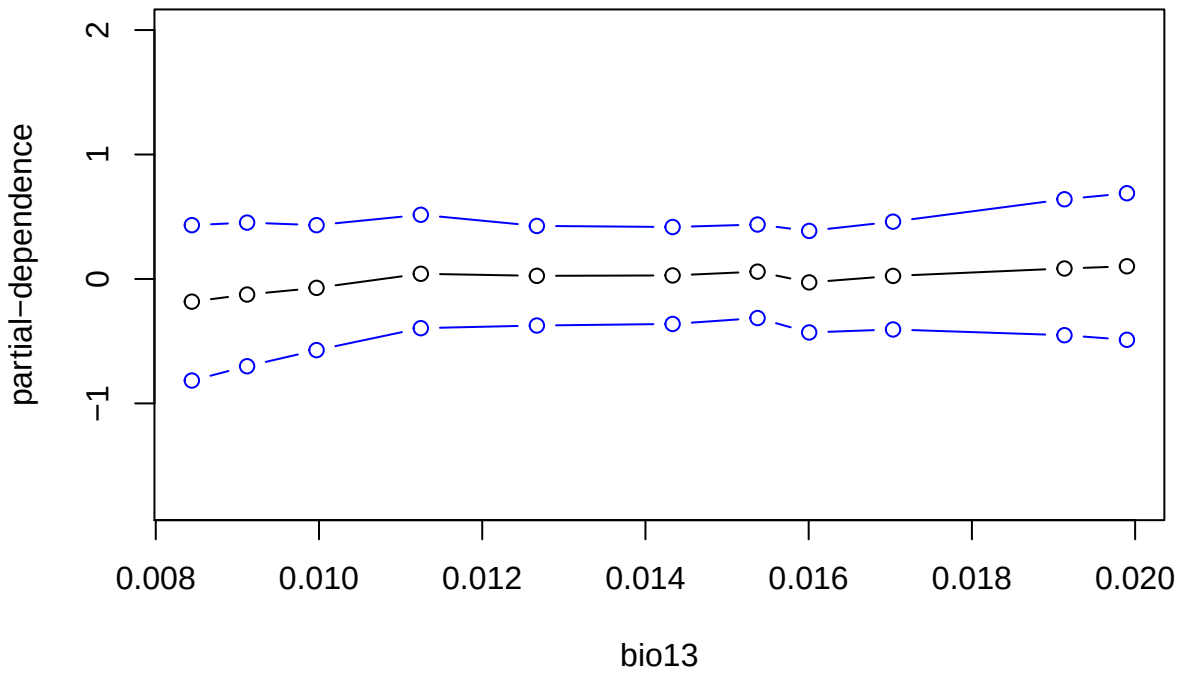
NDVI amplitude and the tick vector come out on top; bio1 and mean NDVI also contribute a lot. Let's look at the response functions a little bit. First, let's look at the partial dependence plots for a couple individual variables:

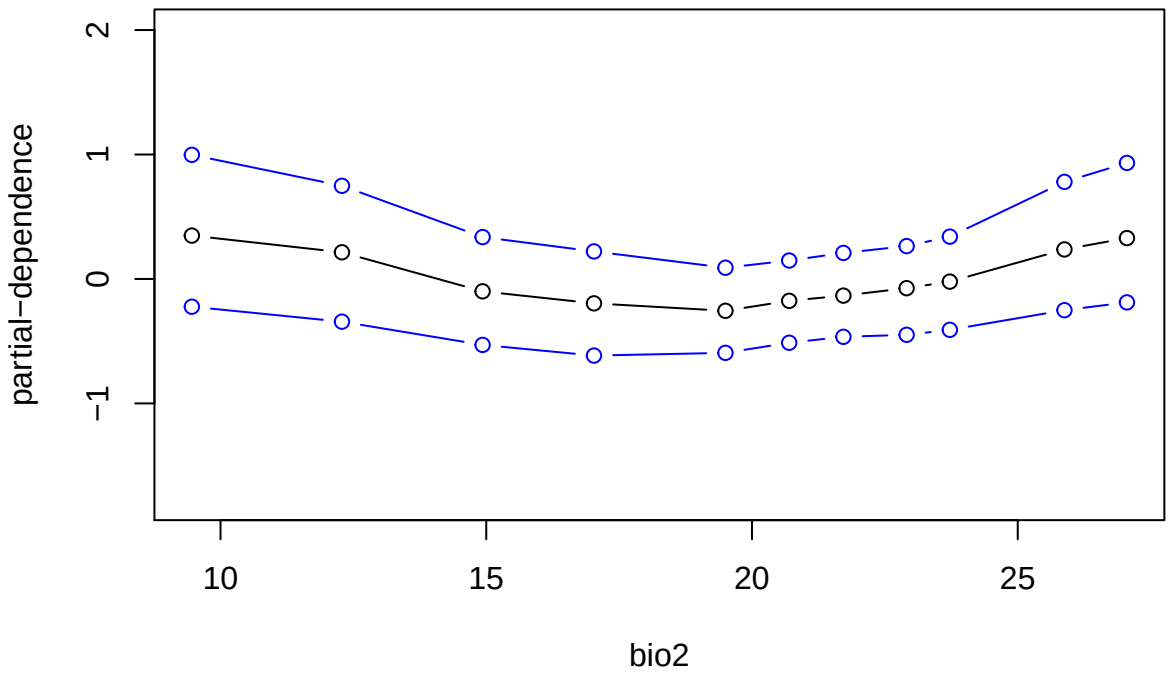
```
# Let's do one variable
p <- pdbart(cchf.model$Model.object, xind=hytr, pl=TRUE)
```

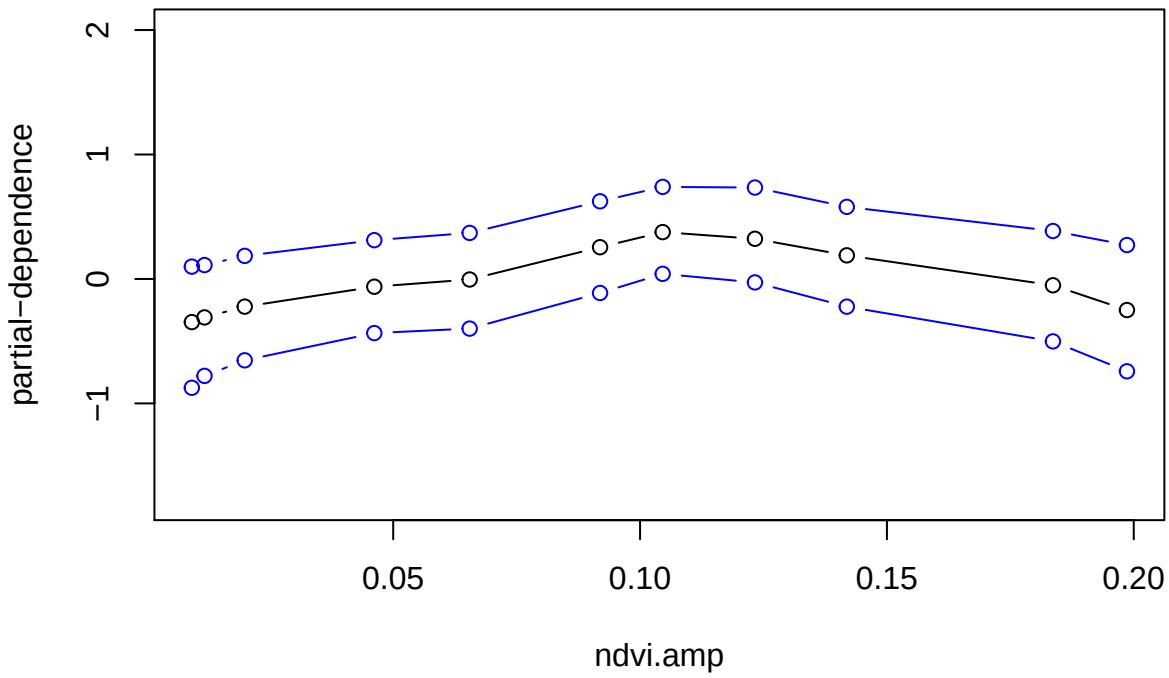


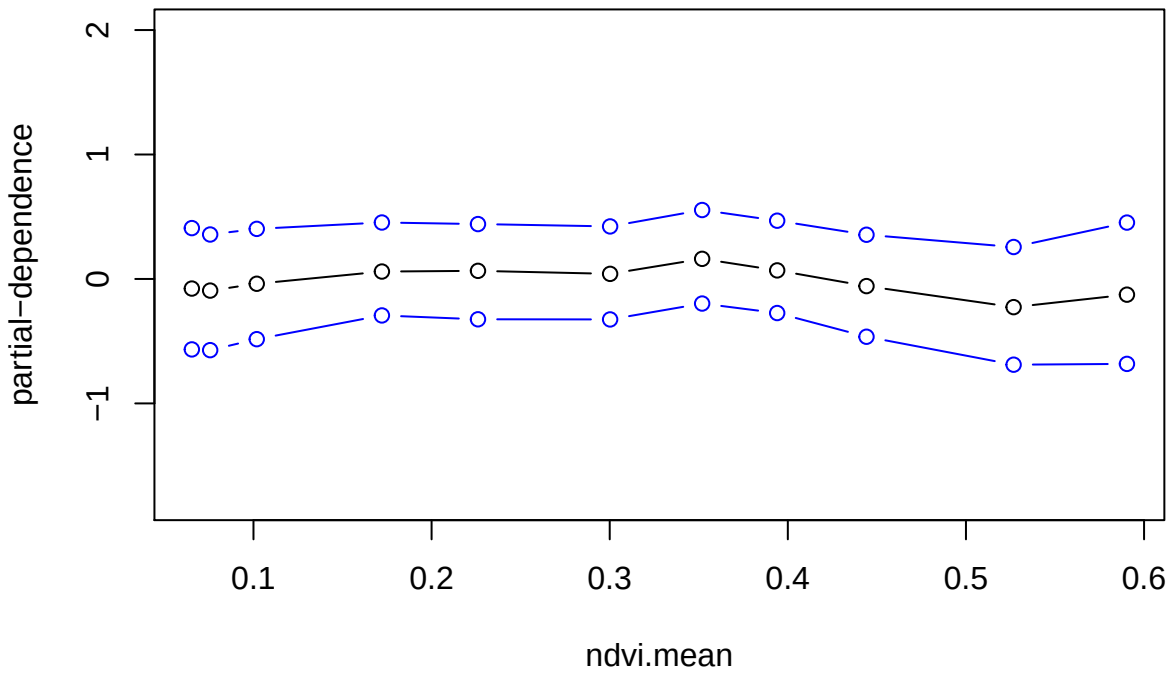

```
# Let's do em all
p <- pdbart(cchf.model$Model.object, pl=TRUE)
```

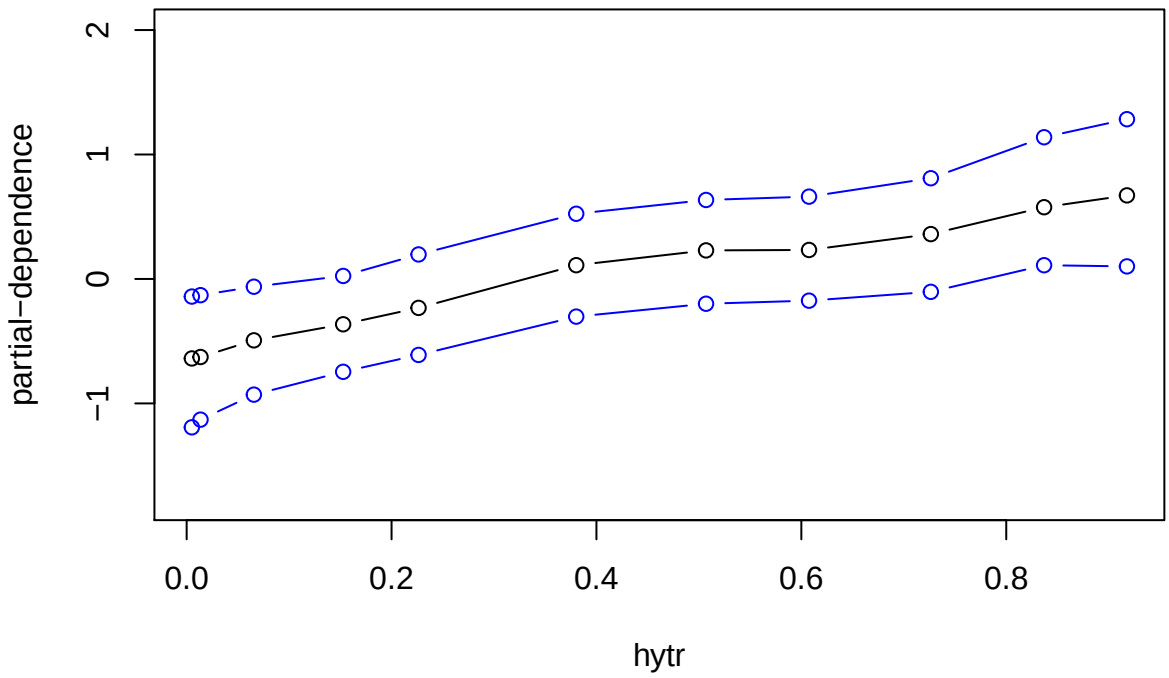

partial−dependence
2
1
0
−1
0.006
0.008
0.010
0.012
0.014
0.016
bio12


partial−dependence
2
1
0
−1
0.008
0.010
0.012
0.014
0.016
0.018
0.020
bio13


partial−dependence
2
1
0
−1
10
15
20
25
bio2


partial−dependence
2
1
0
−1
0.05
0.10
0.15
0.20
ndvi.amp


partial−dependence
2
1
0
−1
0.1
0.2
0.3
0.4
0.5
0.6
ndvi.mean



Some of these patterns are pretty clear - suitability declines above 20 degrees C, and increases with the probability of the tick. NDVI is a little less intuitive to the human mind, but a great feature of BART is that we can pretty easily do two-dimensional partial dependence plots, and we can pretty easily visualize the optimum:

#{r} #p <- pd2bart(cchf.model$Model.object, xind=c('ndvi.mean', 'ndvi.amp'), #pl=TRUE) # # Come back to this when you know which variables make it in

One downside of the BART plots is that they just spit out a lot of text. That's why the objects are being saved to "p".