Mapping Hyalomma and CCHF in Africa

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. How spatial prediction works with the posterior distribution
5. Cool things you can do with partial dependence plots

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

```
library(embarcadero)
#> Loading required package: raster
#> Loading required package: sp
#>
#> Attaching package: 'raster'
#> The following object is masked from 'package:dplyr':
#>
#>     select
#> The following object is masked from 'package:tidyr':
#>
#>     extract
#> Loading required package: dbarts
#> Loading required package: Metrics
#> Loading required package: dismo
#> Loading required package: ROCR
#> Loading required package: gplots
#>
#> Attaching package: 'gplots'
#> The following object is masked from 'package:stats':
#>
#>     lowess
library(velox)
```

Doors are closing; please stand clear of the doors.

Data entry

Let's start by loading in the sample data and predictor set. A set of 11 pre-processed covariates are automatically provided: WorldClim variables BIO 1, 2, 5, 6, 12, 13, 14, and 15; NDVI mean and amplitude; and percent cropland.

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

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.

```
pres.cov <- raster::extract(covs, SpatialPoints(ticks[,c('Longitude.X','Latitude.Y')]))
head(pres.cov)
#>          bio1      bio12      bio13       bio14     bio15     bio2
#> [1,] 23.40137 0.014528484 0.01654123 0.011125606 0.1861420 12.46577
#> [2,] 27.40958 0.013695496 0.01698302 0.009386292 0.2613297 15.32046
#> [3,]       NA          NA         NA          NA        NA       NA
#> [4,] 26.30052 0.009260052 0.01398302 0.004924487 0.3049060 21.40000
#> [5,] 26.03661 0.012159383 0.01609366 0.007829520 0.2941052 18.69936
#> [6,] 21.67183 0.012053753 0.01467431 0.008025348 0.2407484 15.09896
#>          bio5     bio6  crop  ndvi.amp ndvi.mean
#> [1,] 32.50000 13.30386 0.085 0.1562955 0.4837922
#> [2,] 39.79676 16.74935 0.028 0.1792627 0.4234426
#> [3,]       NA       NA    NA        NA        NA
#> [4,] 42.00000 10.30000 0.040 0.1427580 0.3324099
#> [5,] 40.69899 13.80037 0.000 0.1877721 0.4223068
#> [6,] 35.37956 10.47286 0.036 0.1573135 0.4726681
```

Next, let's generate an equal number of pseudoabsences around Africa to the number of presences we have.

```
#Generate the data
absence <- randomPoints(covs,5000) #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 23.40137 0.014528484 0.01654123 0.011125606 0.1861420 12.46577 32.50000
#> 2 27.40958 0.013695496 0.01698302 0.009386292 0.2613297 15.32046 39.79676
#> 3       NA          NA         NA          NA        NA       NA       NA
#> 4 26.30052 0.009260052 0.01398302 0.004924487 0.3049060 21.40000 42.00000
#> 5 26.03661 0.012159383 0.01609366 0.007829520 0.2941052 18.69936 40.69899
#> 6 21.67183 0.012053753 0.01467431 0.008025348 0.2407484 15.09896 35.37956
#>       bio6  crop  ndvi.amp ndvi.mean tick
#> 1 13.30386 0.085 0.1562955 0.4837922    1
#> 2 16.74935 0.028 0.1792627 0.4234426    1
#> 3       NA    NA        NA        NA    1
#> 4 10.30000 0.040 0.1427580 0.3324099    1
#> 5 13.80037 0.000 0.1877721 0.4223068    1
#> 6 10.47286 0.036 0.1573135 0.4726681    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

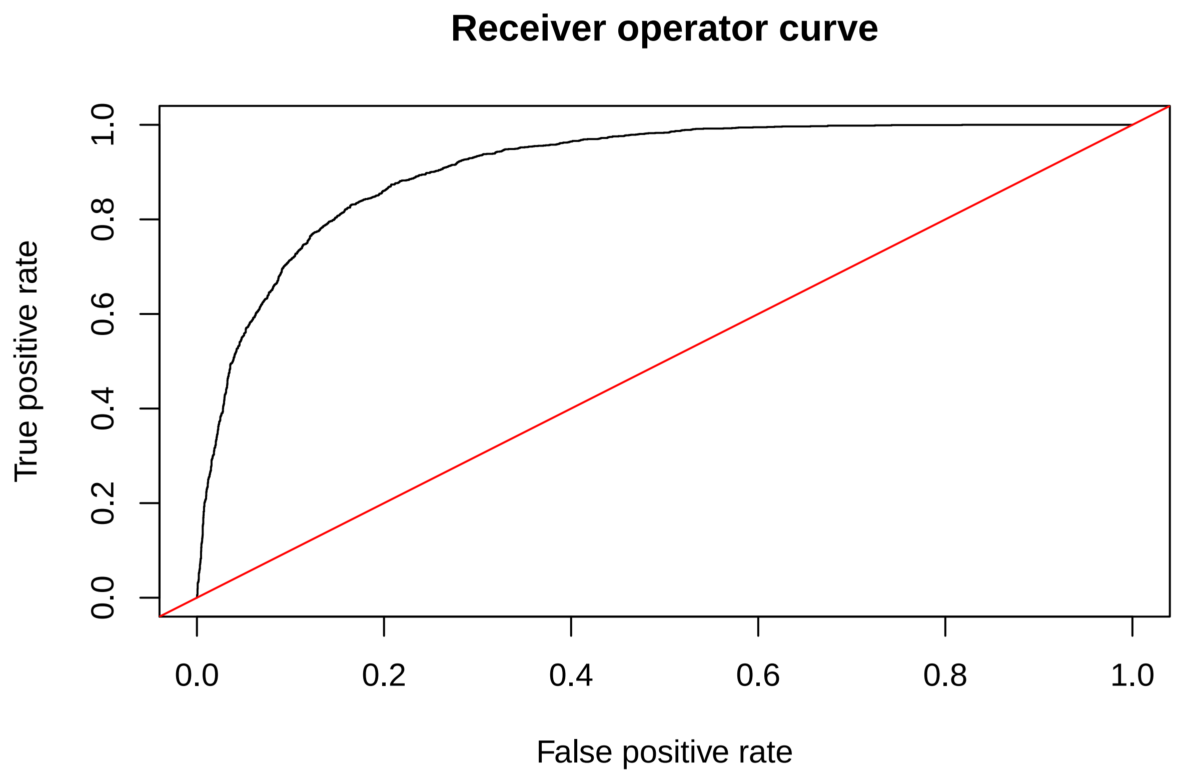
We could try something really simple on defaults, right out the gate. The *bart* function in *dbarts* can just be run on defaults:

```
bad.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: 6759
#>  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: 41.281994
#>
#> Tree sizes, last iteration:
```

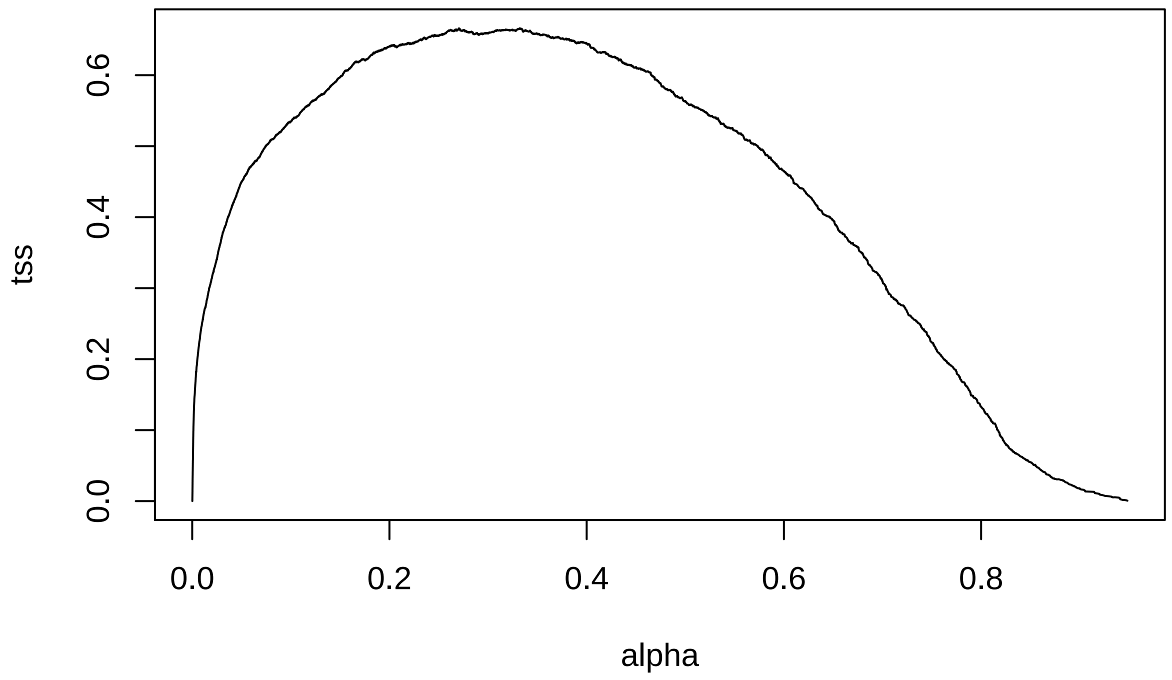
```
#> [1] 4 3 3 4 3 2 2 2 4 4 2 2 3 3 3 4 2 2
#> 3 2 4 3 3 3 2 2 3 2 3 2 2 5 2 2 2 3 2 2
#> 3 2 4 3 2 2 2 2 3 3 2 2 3 3 3 4 2 3 3 1
#> 2 2 2 3 3 2 2 2 2 3 2 3 3 3 2 3 2 2 2 4
#> 3 2 2 2 2 3 2 2 2 2 2 1 2 2 2 3 3 3 2 2
#> 2 2 3 2 2 3 2 2 4 2 3 2 2 3 4 4 2 2 2 2
#> 2 2 3 2 3 2 3 3 2 2 6 3 3 4 3 2 4 3 2 2
#> 4 3 2 3 2 3 3 2 2 2 2 2 2 2 3 3 2 3 2 3
#> 2 3 2 3 3 3 2 2 2 2 2 2 2 3 3 5 2 2 3 1
#> 2 2 2 2 2 3 2 2 1 2 2 3 5 1 5 2 2 2 5 1
#> 2 1
#>
#> Variable Usage, last iteration (var:count):
#> (1: 25) (2: 31) (3: 26) (4: 28) (5: 23)
#> (6: 35) (7: 18) (8: 35) (9: 20) (10: 24)
#> (11: 40)
#> 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(bad.model, all.cov[,'tick'])
#> [1] "AUC = "
#> [1] 0.9109731
```



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

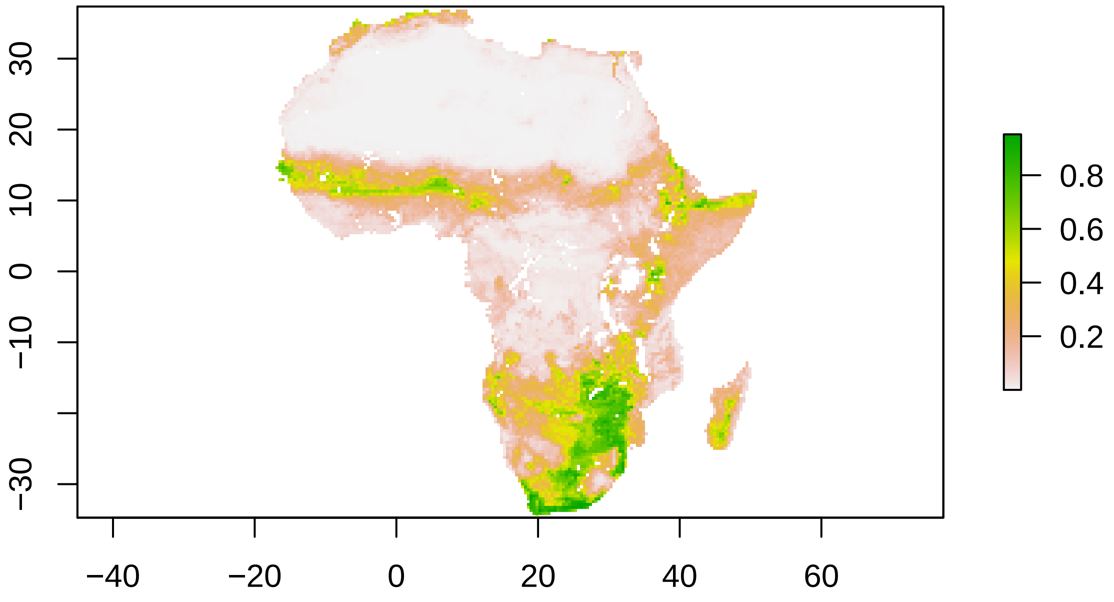


```
#> [1] "TSS threshold"
#> [1] 0.3322533
#> [1] "Type I error rate"
#> [1] 0.1688459
#> [1] "Type II error rate"
#> [1] 0.165
```

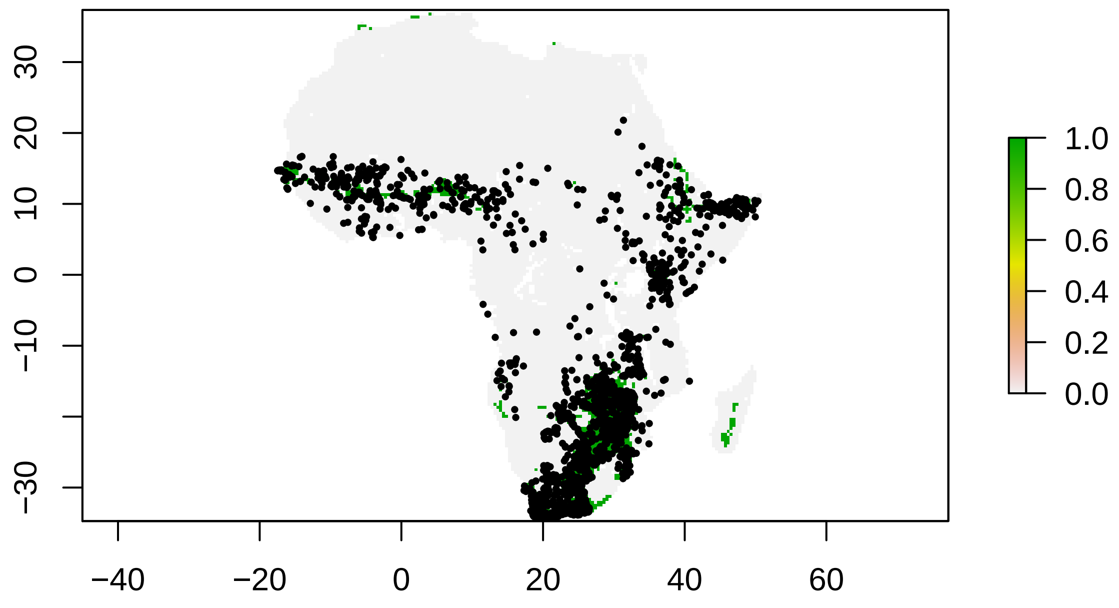
A high AUC value indicates our model performs well. The AUC function also returns an optimal threshold that maximizes the true skill statistic (TSS), and the sensitivity/specificity of the model at that cutoff (alpha).

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*. First, we'll aggregate the predictors a bit so it'll predict faster.

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



```
# What does it look like with the threshold?

 plot(pred.prelim > 0.54)
 points(SpatialPoints(ticks[,c('Longitude.X','Latitude.Y')]),
        pch=16, cex=0.5)
```



This doesn't seem to be behaving great - it's definitely overfitting. We're also 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, but with much of the inhabited area not being predicted, let's revisit that later.

Maybe we need to thin out some of this occurrence data to one point per pixel - this is SUPER dense, and it's swamping out the minor points in the prediction.

Next, let's try some automated variable selection.

How's our model looking now? Do we need a different threshold?