Supporting Information Appendix S2 - Model tuning, model evaluation and predictions

S. Derville, L. Torres, C. Iovan, C. Garrigue

22 mai, 2018

Supporting information S2 to "Finding the right fit: Comparative cetacean distribution models using multiple data sources and statistical approaches", 2018, Diversity and Distributions. Contact information: solene.derville@ird.fr

Runs on R version 3.3.2 (2016-10-31) Platform: x86_64-pc-linux-gnu (64-bit)

BRTs, **MAXENT** and **SVMs** were subject to a preliminary tuning stage ensuring optimal performance within the scope of our training datasets. For **BRTs**, different combinations of learning rates (0.005, 0.01, 0.05) and tree complexity (1, 2, 3) were tested. Folds were set at random and other parameters were left as default in the *gbm* R package (version 2.1.1). **MAXENT** models were built with the *dismo* R package (version 1.1-1): the regularization parameter beta was tested over 4 values (1, 4, 7, 10) and the features type was allowed to be linear only, linear or quadratic, and linear or quadratic or hinge. Finally, 2-class **SVMs** were built with the *e1071* R package (version 1.6-7) using the C-classification algorithm with gamma equal to the inverse of data dimension, and differential class-weighting. Linear, polynomial (degree 2 and 3) and RBF (radial) kernels were tested, with cost (i.e. constant of regularization) ranging 1e-5, 1e-2, 1 and 10. **GLMs** and **GAMs** were fitted using the *glm2* (version 1.1.2) and *mgcv* (version 1.8-12) R packages respectively, using a clog-log link function to account for zero-inflation. In the **GLMs**, each predictor was included as a cubic orthogonal polynomial with function poly(). In the **GAMs**, restricted maximum likelihood (REML) was used to optimize parameter estimates for the thin plate regression splines. Weights were applied in each statistical approach to account for differences in prevalence in each training dataset of the cross-validation (so that the sum of weights on presences equals the sum of weights on background points).

R Packages

```r
library(pROC)
library(dismo)
library(plyr)
library(ggplot2)
library(SDMTools)
library(gbm)
library(e1071)
library(mgcv)
library(caret)
library(glm2)
library(viridis)
library(tidyr)
```

Brief description of input data

```r
# path to folder containing input files
input_path <- "./myinput_path/"
# path to folder containing output files
```

```r
output_path <- "./myoutput_path/"

# training and evaluation data have the same structure: a list of 50 dataframes which,
# when paired, form the PAsurvey_df full dataframe. This dataset contains all presence and
# control points (0/1) and the associated lat-long positions, date, time and environmental
# variables extracted at these positions.
load(paste(input_path, "training.df.day.RData", sep=""))
load(paste(input_path, "evaluation.df.day.RData", sep=""))
load(paste(input_path, "PAsurvey_df.RData", sep=""))
fulldata <- PAsurvey_df
```

```r
head(PAsurvey_df)
```

```
##          day      lon       lat presence    mercX    mercY B0.5km_ras
## 1 2003-07-21 167.0120 -22.45187        1 223972.4 -2549552  -70.67655
## 2 2003-07-21 166.9472 -22.40007        1 216765.2 -2543350  -43.03870
## 3 2003-07-25 167.0327 -22.58678        1 226275.7 -2565718  -84.04722
## 4 2003-07-26 167.0784 -22.49272        1 231368.1 -2554445  -73.18062
## 5 2003-07-26 167.0086 -22.45628        1 223600.4 -2550082  -45.06274
## 6 2003-07-27 167.1988 -22.62930        1 244766.6 -2570816  -81.43746
##   S.COV5km_ras S.AVG5km_ras A.AVG5km_ras  dissurf Cpromean5km_ras
## 1    0.8654183   0.01289129     2.808440 1118.034    0.0013338733
## 2    1.4753360   0.02533339     3.104411 1414.214    0.0008401876
## 3    0.8410672   0.01347645     2.718718 9219.544    0.0003836328
## 4    1.1784818   0.01282853     2.858851 4242.641    0.0012118479
## 5    0.9357634   0.01181535     2.829403  500.000    0.0011356844
## 6    0.5864632   0.01760744     3.581625 7632.169   -0.0001628439
##         k490      sst julian month    sst_m     k490_m B0.5km_ras.s
## 1 0.07460079 22.90034    201     7 22.77276 0.07727450    0.2416297
## 2 0.07800070 23.00572    201     7 22.93007 0.07945000    0.3368806
## 3 0.05529805 22.76803    205     7 22.87389 0.06032233    0.1955490
## 4 0.06632276 22.84001    206     7 22.70535 0.07946300    0.2329996
## 5 0.07486253 22.89637    206     7 22.77276 0.07727450    0.3299049
## 6 0.05856973 22.80029    207     7 22.99939 0.05738700    0.2045433
##   S.COV5km_ras.s S.AVG5km_ras.s A.AVG5km_ras.s   dissurf.s
## 1    -0.11210209     -0.4138485     -0.3001811 -0.30771625
## 2     1.46444384     -0.1707735      0.1051968 -0.29455165
## 3    -0.17504603     -0.4024164     -0.4230699  0.05238013
## 4     0.69712023     -0.4150746     -0.2311360 -0.16883356
## 5     0.06972958     -0.4348685     -0.2714689 -0.33518666
## 6    -0.83315934     -0.3217115      0.7588129 -0.01817563
##   Cpromean5km_ras.s     k490.s      sst.s   julian.s   month.s      sst_m.s
## 1        0.55976353  0.1891624 -0.6038469 -1.1879022 -1.298995 -0.08360858
## 2        0.29149819  0.3136858 -0.4354744 -1.1879022 -1.298995  0.10768296
## 3        0.04340955 -0.5178111 -0.8152527 -1.0121515 -1.298995  0.03936444
## 4        0.49345577 -0.1140251 -0.7002390 -0.9682138 -1.298995 -0.16559033
## 5        0.45206910  0.1987488 -0.6101937 -0.9682138 -1.298995 -0.08360858
## 6       -0.25354197 -0.3979840 -0.7637059 -0.9242761 -1.298995  0.19199037
##     k490_m.s
## 1  0.2377648
## 2  0.3004586
## 3 -0.2507651
## 4  0.3008333
## 5  0.2377648
## 6 -0.3353559
```

```
# the eez_argos_df_eval dataframe contains all ARGOS tracking positions used as an
# external validation four the models
load(paste(input_path, "eez_argos_df_eval.RData", sep=""))
```

```
head(eez_argos_df_eval)
```

```
##            id                time lq     lat     lon       date year index
## 1 2007-24638 2007-08-20 03:14:08  3 -22.592 166.626 2007-08-20 2007     1
## 2 2007-24638 2007-08-20 05:15:26  A -22.546 166.659 2007-08-20 2007     2
## 3 2007-24638 2007-08-20 06:56:36  2 -22.537 166.730 2007-08-20 2007     3
## 5 2007-24638 2007-08-21 05:09:01  2 -22.883 166.940 2007-08-21 2007     5
## 7 2007-24638 2007-08-22 04:57:30  1 -22.554 167.043 2007-08-22 2007     7
## 8 2007-24638 2007-08-22 06:08:15  2 -22.538 167.080 2007-08-22 2007     8
##      mercX    mercY      dist         dt    speed julian week B0.5km_ras
## 1 181006.3 -2566343  0.000000  0.0000000      NaN    231   34 -11.500000
## 2 184678.7 -2560830  6.114244  2.0216667 3.024358    231   34 -26.451044
## 3 192582.2 -2559751  7.370843  1.6861111 4.371505    231   34 -21.421913
## 5 215960.1 -2601267  1.509194  0.2930556 5.149857    232   34  -9.202993
## 7 227425.6 -2561789 40.464162 22.3311111 1.812008    233   34 -69.537998
## 8 231544.7 -2559871  4.197299  1.1791667 3.559547    233   34 -75.652233
##   S.COV5km_ras S.AVG5km_ras A.AVG5km_ras  dissurf Cpromean5km_ras
## 1    0.9403521  0.001836791     2.160027 2500.000   -6.653685e-05
## 2    0.6214701  0.002817016     2.191043 3162.278    7.499037e-05
## 3    0.7810952  0.002986507     2.621770 3201.562   -1.116760e-04
## 5    0.5865140  0.008496768     1.807665  500.000   -3.349758e-04
## 7    0.8047059  0.009407016     3.555864 9340.771    4.748820e-04
## 8    1.1337432  0.008867233     3.122973 8500.000    8.960429e-04
##         k490      sst month    sst_m     k490_m  julian.s B0.5km_ras.s
## 1 0.06172557 22.80391     8 22.06209 0.05966250 0.1302280    0.4455753
## 2 0.06453889 22.67365     8 21.77970 0.05681525 0.1302280    0.3940481
## 3 0.08212616 22.69553     8 22.00562 0.08144000 0.1302280    0.4113805
## 5 0.06277473 22.71218     8 22.27289 0.05600775 0.1741657    0.4534917
## 7 0.05930430 22.78100     8 21.80074 0.05879833 0.2181033    0.2455535
## 8 0.05955499 22.82218     8 22.17971 0.06180233 0.2181033    0.2244815
##   S.COV5km_ras.s S.AVG5km_ras.s A.AVG5km_ras.s   dissurf.s
## 1     0.08159051     -0.6298146     -1.1882817 -0.24629055
## 2    -0.74267166     -0.6106644     -1.1458013 -0.21685360
## 3    -0.33006466     -0.6073531     -0.5558540 -0.21510748
## 5    -0.83302784     -0.4997020     -1.6708955 -0.33518666
## 7    -0.26903465     -0.4819189      0.7235300  0.05776841
## 8     0.58147769     -0.4924644      0.1306201  0.02039778
##   Cpromean5km_ras.s     k490.s      sst.s    sst_m.s   k490_m.s
## 1       -0.20120942 -0.2823997 -0.7579237 -0.9478347 -0.2697802
## 2       -0.12430454 -0.1793604 -0.9660458 -1.2912435 -0.3518327
## 3       -0.22573771  0.4647830 -0.9310909 -1.0165161  0.3578067
## 5       -0.34707724 -0.2439734 -0.9044876 -0.6914959 -0.3751033
## 7        0.09299374 -0.3710798 -0.7945261 -1.2656605 -0.2946839
## 8        0.32184960 -0.3618984 -0.7287336 -0.8048027 -0.2081142
```

```
# RasterStack of environmental variables
# with rasters of dynamic variables (SST, K490) averaged over a climatology 2003 to 2016
env_stack <- stack(paste(input_path, "env_stack.grd", sep=""))
```

```r
nb_cv <- 50 # total number of MonteCarlo-CV iterations to run
```

1- Model Tuning and evaluation

1.1 - BRT

```r
##########################
###CREATE MODELS
lr <- c(0.005, 0.01, 0.05)
names(lr) <- as.character(lr)
model_tun <- adply(lr, .margins = 1, .id = "lr", function(lr) {
  mod_tc <- adply(c(1,2,3), .margins = 1, .id = "tc", function(tc) {
    mod_cv <- adply(seq(1, nb_cv, 1), .margins = 1, .id = "cv.run", function(i) {
      m <- gbm.step(training.df.day[[i]],
                    gbm.x = c(19, 20, 21, 23, 27, 29, 30),
                    gbm.y = 7,
                    tree.complexity = tc,
                    learning.rate = lr,
                    family = "bernoulli",
                    n.trees = 50,
                    bag.fraction = 0.75,
                    plot.main = T,
                    verbose = F,
                    site.weights = training.df.day[[i]]$weights,
                    silent=T)
      save(m, file = paste(output_path,"survey_simple_brt.m_lr",lr,"_tc",tc,"_",i,
                          ".RData", sep=""))

      ### calculate evaluation metrics for each model and rbind in dataframe
      df <- data.frame(trees = c(NA), int.AUC = c(NA), ext.AUC = c(NA), diff.AUC = c(NA),
                    thresh = c(NA), TSS = c(NA), sensitivity_argos = c(NA))
      df$trees <- m$gbm.call$best.trees
      # predictions to evaluation and training
      pred.ext <- predict(m, evaluation.df.day[[i]], n.trees = m$gbm.call$best.trees,
                         type = "response")
      pred.int <- predict(m, training.df.day[[i]], n.trees = m$gbm.call$best.trees,
                         type = "response")
      # AUC = Threshold-independent metric of performance
      df$int.AUC <- round(roc(training.df.day[[i]]$presence, pred.int)$auc,3)
      df$ext.AUC <- round(roc(evaluation.df.day[[i]]$presence, pred.ext)$auc,3)
      df$diff.AUC <- round(df$int.AUC - df$ext.AUC, 3)
      # TSS = threshold-dependent metric calculated on confusion matrix from pred.ext
      thresh <- SDMTools::optim.thresh(obs = evaluation.df.day[[i]]$presence,
                pred = pred.ext, threshold = 1001)$`max.sensitivity+specificity`[1]
      df$thresh <- thresh
      mat <- SDMTools::confusion.matrix(obs = evaluation.df.day[[i]]$presence,
                                        pred = pred.ext, threshold = thresh)
      sen <- SDMTools::sensitivity(mat)
      spe <- SDMTools::specificity(mat)
      df$TSS <- sen + spe - 1
      # Using the treshold calculated above on pred.ext, predict sensitivity
      # of predictions to argos positions
      pred.argos <- predict(m, eez_argos_df_eval, n.trees = m$gbm.call$best.trees,
                           type = "response")
      # calculate percent well classified as presences
      df$sensitivity_argos <- round(length(pred.argos[pred.argos > thresh])/
                                   length(pred.argos)*100,2)

      rm(m)
      return(df)
    })
    return(mod_cv)
  })
  return(mod_tc)
})
save(model_tun, file = paste(output_path, "model_brt_tun.RData", sep = ""))

######################################
### AVERAGE PERFORMANCE ACROSS CV RUNS

summary_tun <- ddply(model_tun, . (lr, tc), function(mod){
  mean <- apply(mod[,c(4:ncol(mod))], 2, function(x) {mean(x, na.rm=T)})
  sd <- apply(mod[,c(4:ncol(mod))], 2, function(x) {sd(x, na.rm=T)})
  model_summary <- data.frame(rbind(mean, sd))
  model_summary <- round(model_summary, 3)
  model_summary$value <- c("mean","sd")
  return(model_summary)
})
write.csv(summary_tun, file = paste(output_path, "summary_brt_tun.csv", sep = ""),
          row.names = F)

######################################
### VARIABLE CONTRIBUTIONS

########################################### BEST STABLE MODEL
# load each model one by one and extract the contribution % for each predictor
load(paste(output_path, "survey_simple_brt.m_lr0.005_tc1_1.RData", sep=""))
contribution <- m$contributions # contributions are already set to sum to 100%
for (i in c(2:nb_cv)){
  load(paste(output_path, "survey_simple_brt.m_lr0.005_tc1_",i,".RData", sep=""))
  contribution <- merge(contribution, m$contributions, by = "var") # ignore warnings
}

# mean contribution of each predictor
contribution <- ddply(contribution, ~var, function(d) {
  dd <- data.frame(mean = round(apply(d[,2:ncol(d)], 1, function(w) {mean(w, na.rm=T)}),3),
                   sd = round(apply(d[,2:ncol(d)], 1, function(w) {sd(w, na.rm=T)}), 3))
  return(dd)
})
contribution$model <- "stable_model"
```

```
########################################### BEST PREDICTIVE MODEL
# load each model one by one and extract the contribution % for each predictor
load(paste(output_path, "survey_simple_brt.m_lr0.005_tc3_1.RData", sep=""))
contribution2 <- m$contributions
for (i in c(2:nb_cv)){
  load(paste(output_path, "survey_simple_brt.m_lr0.005_tc3_",i,".RData", sep=""))
  contribution2 <- merge(contribution2, m$contribution, by = "var") # ignore warnings
}

# mean contribution of each predictor
contribution2 <- ddply(contribution2, ~var, function(d) {
  dd <- data.frame(mean = round(apply(d[,2:ncol(d)], 1, function(w) {mean(w, na.rm=T)}),3),
                   sd = round(apply(d[,2:ncol(d)], 1, function(w) {sd(w, na.rm=T)}), 3))
  return(dd)
})
contribution2$model <- "predictive_model"

contribution <- rbind(contribution, contribution2)

write.csv(contribution, file = paste(output_path,
                        "mean_simple_contributions_brt.csv", sep = ""), row.names = F)
```

1.2 - MAXENT

```
##########################
###CREATE MODELS
features <- list(lqh = c('linear=true','quadratic=true','hinge=true'),
                 lq = c('linear=true','quadratic=true','hinge=false'),
                 l = c('linear=true','quadratic=false','hinge=false'))
betas <- list(beta1=c('betamultiplier=1'),
              beta4=c('betamultiplier=4'),
              beta7=c('betamultiplier=7'),
              beta10=c('betamultiplier=10'))
model_tun <- adply(names(features), .margins = 1, .id = "features", function(ft) {
  mod_beta <- adply(names(betas), .margins = 1, .id = "beta", function(b) {
    mod_cv <- adply(seq(1, nb_cv, 1), .margins = 1, .id = "cv.run", function(i) {
      m <- dismo::maxent(x = training.df.day[[i]][c(19, 20, 21, 23, 27, 29, 30)],
                         p = training.df.day[[i]]$presence, removeDuplicates=F,
                         args=c(betas[[b]],
                        'pictures=false',
                        features[[ft]],
                        'product=false',
                        'threshold=false',
                        'threads=2',
                        'responsecurves=false',
                        'jackknife=false',
                        'askoverwrite=false'
      ))
      save(m, file = paste(output_path, "survey_simple_max.m_ft",ft,"_b",b,"_",i,
                           ".RData", sep=""))

      ### calculate evaluation metrics for each model and rbind in dataframe
      df <- data.frame(int.AUC = c(NA), ext.AUC = c(NA), diff.AUC = c(NA),
                       thresh = c(NA), TSS = c(NA), sensitivity_argos = c(NA))
      # predictions to evaluation and training
      pred.ext <- predict(m, evaluation.df.day[[i]], type = "response")
      pred.int <- predict(m, training.df.day[[i]], type = "response")
      # AUC = Threshold-independent metric of performance
      df$int.AUC <- round(roc(training.df.day[[i]]$presence, pred.int)$auc,3)
      df$ext.AUC <- round(roc(evaluation.df.day[[i]]$presence, pred.ext)$auc,3)
      df$diff.AUC <- round(df$int.AUC - df$ext.AUC, 3)
      # TSS = threshold-dependent metric calculated on confusion matrix from pred.ext
      thresh <- SDMTools::optim.thresh(obs = evaluation.df.day[[i]]$presence,
                      pred = pred.ext, threshold = 1001)$`max.sensitivity+specificity`[1]
      df$thresh <- thresh
      mat <- SDMTools::confusion.matrix(obs = evaluation.df.day[[i]]$presence,
                                        pred = pred.ext, threshold = thresh)
      sen <- SDMTools::sensitivity(mat)
      spe <- SDMTools::specificity(mat)
      df$TSS <- sen + spe - 1
      # Using the treshold calculated above on pred.ext,
      # I predict sensitivity of predictions to argos positions
      pred.argos <- predict(m, eez_argos_df_eval, type = "response")
      # calculate percent well classified as presences
      df$sensitivity_argos <- round(length(pred.argos[pred.argos > thresh])/
                                      length(pred.argos)*100,2)

      rm(m)
      return(df)
    })
    return(mod_cv)
  })
  return(mod_beta)
})
save(model_tun, file = paste(output_path,"model_max_tun.RData", sep=""))

######################################
### AVERAGE PERFORMANCE ACROSS CV RUNS

summary_tun <- ddply(model_tun, . (features, beta), function(mod){
  mean <- apply(mod[,c(4:ncol(mod))], 2, function(x) {mean(x, na.rm=T)})
  sd <- apply(mod[,c(4:ncol(mod))], 2, function(x) {sd(x, na.rm=T)})
  model_summary <- data.frame(rbind(mean, sd))
  model_summary <- round(model_summary, 3)
  model_summary$value <- c("mean","sd")
  return(model_summary)
})
write.csv(summary_tun, file = paste(output_path,"summary_max_tun.csv",sep=""),
          row.names = F)


######################################
### VARIABLE CONTRIBUTIONS

########################################## BEST STABLE MODEL
```

```r
# load each model one by one and extract the contribution % for each predictor
load(paste(output_path, "survey_simple_max.m_ftl_bbeta1_1.RData", sep=""))
var.contrib <- plot(m)
contribution <- data.frame(var = names(var.contrib), rel.inf = var.contrib)
for (i in c(2:nb_cv)){
  load(paste(output_path, "survey_simple_max.m_ftl_bbeta1_",i,".RData", sep=""))
  var.contrib <- plot(m)
  newcontribution <- data.frame(var = names(var.contrib), rel.inf = var.contrib)
  contribution <- merge(contribution, newcontribution, by = "var") # ignore warnings
}

# mean contribution of each predictor
contribution <- ddply(contribution, ~var, function(d) {
  dd <- data.frame(mean = round(apply(d[,2:ncol(d)], 1, function(w) {mean(w, na.rm=T)}),3),
                   sd = round(apply(d[,2:ncol(d)], 1, function(w) {sd(w, na.rm=T)}), 3))
  return(dd)
})
contribution$model <- "stable_model"

########################################### BEST PREDICTIVE MODEL

# load each model one by one and extract the contribution % for each predictor
load(paste(output_path, "survey_simple_max.m_ftlqh_bbeta1_1.RData", sep=""))
var.contrib <- plot(m)
contribution2 <- data.frame(var = names(var.contrib), rel.inf = var.contrib)
for (i in c(2:nb_cv)){
  load(paste(output_path, "survey_simple_max.m_ftlqh_bbeta1_",i,".RData", sep=""))
  var.contrib <- plot(m)
  newcontribution <- data.frame(var = names(var.contrib), rel.inf = var.contrib)
  contribution2 <- merge(contribution2, newcontribution, by = "var") # ignore warnings
}

# mean contribution of each predictor
contribution2 <- ddply(contribution2, ~var, function(d) {
  dd <- data.frame(mean = round(apply(d[,2:ncol(d)], 1, function(w) {mean(w, na.rm=T)}),3),
                   sd = round(apply(d[,2:ncol(d)], 1, function(w) {sd(w, na.rm=T)}), 3))
  return(dd)
})
contribution2$model <- "predictive_model"

contribution <- rbind(contribution, contribution2)

write.csv(contribution, file = paste(output_path,
                        "mean_complex_contributions_brt.csv", sep = ""), row.names = F)
```

1.3 - SVM

```r
###########################
###CREATE MODELS
kernel <- c("linear", "polynomial", "polynomial","radial")
names(kernel) <- c("linear", "polynomial2", "polynomial3","radial")
```

```r
degree <- c(0, 2, 3, 0)

d <- 0

cost <- c(1e-5, 1e-2, 1, 10)
names(cost) <- as.character(cost)

model_tun <- adply(kernel, .margins = 1, .id = "kernel", function(k) {
  d <<- d + 1
  mod_k <- adply(cost, .margins = 1, .id = "cost", function(co) {
    mod_cv <- adply(seq(1, nb_cv, 1), .margins = 1, .id = "cv.run", function(i) {
      print(i)
      m <- e1071::svm(presence ~ S.COV5km_ras.s +
                                    sst_m.s +
                                    k490_m.s +
                                    S.AVG5km_ras.s +
                                    B0.5km_ras.s +
                                    dissurf.s +
                                    julian.s,
              probability = T, type = "C-classification",
              class.weights = c("0" = training.df.day[[i]][
                    training.df.day[[i]]$presence == 0,"weights"][1], "1" = 1),
              data=training.df.day.svm[[i]],
              kernel = k, degree = degree[d], cost = co, gamma = (1/9))

      save(m, file = paste(output_path,"survey_simple_svm.m_", k, degree[d],"_cost",
                          co,"_",i, ".RData", sep=""))

      ### calculate evaluation metrics for each model and rbind in dataframe
      df <- data.frame(int.AUC = c(NA), ext.AUC = c(NA), diff.AUC = c(NA),
                      thresh = c(NA), TSS = c(NA), sensitivity_argos = c(NA))
      # predictions to evaluation and training
      pred.ext <- predict(m, evaluation.df.day[[i]], probability = T)
      pred.ext <- attr(pred.ext, "probabilities")[,2]
      pred.int <- predict(m, training.df.day[[i]], probability = T)
      pred.int <- attr(pred.int, "probabilities")[,2]
      # AUC = Threshold-independent metric of performance
      df$int.AUC <- round(roc(training.df.day[[i]]$presence, pred.int)$auc,3)
      df$ext.AUC <- round(roc(evaluation.df.day[[i]]$presence, pred.ext)$auc,3)
      df$diff.AUC <- round(df$int.AUC - df$ext.AUC, 3)
      # TSS = threshold-dependent metric calculated on confusion matrix from pred.ext
      thresh <- SDMTools::optim.thresh(obs = evaluation.df.day[[i]]$presence,
                  pred = pred.ext, threshold = 1001)$`max.sensitivity+specificity`[1]
      df$thresh <- thresh
      mat <- SDMTools::confusion.matrix(obs = evaluation.df.day[[i]]$presence,
                                    pred = pred.ext, threshold = thresh)
      sen <- SDMTools::sensitivity(mat)
      spe <- SDMTools::specificity(mat)
      df$TSS <- sen + spe - 1
      # Using the treshold calculated above on pred.ext,
      # I predict sensitivity of predictions to argos positions
      pred.argos <- predict(m, eez_argos_df_eval,  probability = T)
      pred.argos <- attr(pred.argos, "probabilities")[,2]
      # calculate percent well classified as presences
      df$sensitivity_argos <- round(length(pred.argos[pred.argos > thresh])/
                                    length(pred.argos)*100,2)

      rm(m)
      return(df)
    })
    return(mod_cv)
  })
  return(mod_k)
})

save(model_tun, file = paste(output_path, "model_svm_tun.RData",
                             sep = ""))

######################################
### AVERAGE PERFORMANCE ACROSS CV RUNS

summary_tun <- ddply(model_tun, . (kernel, cost), function(mod){
  mean <- apply(mod[,c(4:ncol(mod))], 2, function(x) {mean(x, na.rm=T)})
  sd <- apply(mod[,c(4:ncol(mod))], 2, function(x) {sd(x, na.rm=T)})
  model_summary <- data.frame(rbind(mean, sd))
  model_summary <- round(model_summary, 3)
  model_summary$value <- c("mean","sd")
  return(model_summary)
})
write.csv(summary_tun, file = paste(output_path, "summary_svm_tun.csv",
                                    sep = ""), row.names = F)

######################################
### VARIABLE CONTRIBUTIONS

# only assessed for the best "stable model" where the kernel is linear

# load each model one by one and extract the contribution % for each predictor
load(paste(output_path, "survey_simple_svm.m_linear0_cost0.01_1.RData", sep=""))
contribution <-(t(m$coefs) %*% m$SV)^2
contribution <- contribution/sum(contribution)*100 # rescale to percent
contribution <- as.data.frame(t(contribution))
contribution$var <- row.names(contribution)
for (i in c(2:nb_cv)){
  load(paste(output_path, "survey_simple_svm.m_linear0_cost0.01_",i,".RData", sep=""))
  newcontribution <- (t(m$coefs) %*% m$SV)^2 #method RFE by Guyon
  newcontribution <- newcontribution/sum(newcontribution)*100 # rescale to percent
  newcontribution <- as.data.frame(t(newcontribution))
  newcontribution$var <- row.names(newcontribution)
  contribution <- merge(contribution, newcontribution, by = "var") # ignore warnings
}

# mean contribution of each predictor
contribution <- ddply(contribution, ~var, function(d) {
  dd <- data.frame(mean = round(apply(d[,2:ncol(d)], 1, function(w) {mean(w, na.rm=T)}),8),
                   sd = round(apply(d[,2:ncol(d)], 1, function(w) {sd(w, na.rm=T)}), 8))
  return(dd)
})

write.csv(contribution, file = paste(output_path,
                        "mean_simple_contributions_svm.csv", sep = ""), row.names = F)
```

1.4 - GAM

```
##########################
###CREATE MODELS
mod_cv <- adply(seq(1, nb_cv, 1), .margins = 1, .id = "cv.run", function(i) {
  print(i)
  m <- mgcv::gam(presence ~ s(sst_m.s) +
                   s(k490_m.s) +
                   s(julian.s) +
                   s(B0.5km_ras.s) +
                   s(S.COV5km_ras.s) +
                   s(S.AVG5km_ras.s) +
                   s(dissurf.s),
                 weights = training.df.day[[i]]$weights,
                 family = binomial(link="cloglog"), method = "REML",
                 data = training.df.day[[i]])
  save(m, file = paste(output_path,"survey_simple_gam.m_", i, ".RData", sep = ""))

  ### calculate evaluation metrics for each model and rbind in dataframe
  df <- data.frame(int.AUC = c(NA), ext.AUC = c(NA), diff.AUC = c(NA),
                   thresh = c(NA), TSS = c(NA), sensitivity_argos = c(NA))
  # predictions to evaluation and training
  pred.ext <- predict(m, evaluation.df.day[[i]], type = "response")
  pred.int <- predict(m, training.df.day[[i]], type = "response")
  # AUC = Threshold-independent metric of performance
  df$int.AUC <- round(roc(training.df.day[[i]]$presence, pred.int)$auc,3)
  df$ext.AUC <- round(roc(evaluation.df.day[[i]]$presence, pred.ext)$auc,3)
  df$diff.AUC <- round(df$int.AUC - df$ext.AUC, 3)
  # TSS = threshold-dependent metric calculated on confusion matrix from pred.ext
  thresh <- SDMTools::optim.thresh(obs = evaluation.df.day[[i]]$presence,
              pred = pred.ext, threshold = 1001)$`max.sensitivity+specificity`[1]
  df$thresh <- thresh
  mat <- SDMTools::confusion.matrix(obs = evaluation.df.day[[i]]$presence,
                                    pred = pred.ext, threshold = thresh)
  sen <- SDMTools::sensitivity(mat)
  spe <- SDMTools::specificity(mat)
  df$TSS <- sen + spe - 1
  # Using the treshold calculated above on pred.ext,
  # I predict sensitivity of predictions to argos positions
  pred.argos <- predict(m, eez_argos_df_eval, type = "response")
  # calculate percent well classified as presences
  df$sensitivity_argos <- round(length(pred.argos[pred.argos > thresh])/
                                length(pred.argos)*100,2)


  rm(m)
```

```r
  return(df)
})
save(mod_cv, file = paste(output_path, "model_gam_cv.RData", sep = ""))

######################################
### AVERAGE PERFORMANCE ACROSS CV RUNS

mean <- apply(mod_cv[2:ncol(mod_cv)], 2, function(x) {mean(x, na.rm=T)})
sd <- apply(mod_cv[2:ncol(mod_cv)], 2, function(x) {sd(x, na.rm=T)})
model_summary <- data.frame(rbind(mean, sd))
model_summary <- round(model_summary, 3)
model_summary$value <- c("mean","sd")

write.csv(model_summary, file = paste(output_path, "summary_gam_cv.csv",
                                      sep = ""), row.names = F)

######################################
### VARIABLE CONTRIBUTIONS

# load each model one by one and extract the contribution % for each predictor
load(paste(output_path, "survey_simple_gam.m_1.RData", sep=""))
contribution <- caret::varImp(m, scale = TRUE)
contribution <- contribution/sum(contribution$Overall)*100
contribution$var <- row.names(contribution)
for (i in c(2:nb_cv)){
  load(paste(output_path, "survey_simple_gam.m_",i,".RData", sep=""))
  newcontribution <- caret::varImp(m, scale = TRUE)
  newcontribution <- newcontribution/sum(newcontribution$Overall)*100
  newcontribution$var <- row.names(newcontribution)
  contribution <- merge(contribution, newcontribution, by = "var")
}

save(contribution, file = paste(output_output_path, "full_contributions_gam.RData", sep=""))

# mean contribution of each predictor
contribution <- ddply(contribution, ~var, function(d) {
  dd <- data.frame(mean = round(apply(d[,2:ncol(d)], 1, function(w) {
                            mean(w, na.rm=T)}),1),
                   sd = round(apply(d[,2:ncol(d)], 1, function(w) {
                            sd(w, na.rm=T)}), 1))
  return(dd)
})

write.csv(contribution, file = paste(output_path, "mean_contributions_gam.csv",
                                     sep = ""), row.names = F)
```

1.5 - GLM

```r
##########################
###CREATE MODELS
mod_cv <- adply(seq(1, nb_cv, 1), .margins = 1, .id = "cv.run", function(i) {
      print(i)
      m <- glm2::glm2(presence ~ poly(sst_m.s,3) +
                           poly(k490_m.s,3) +
                           poly(julian.s,3) +
                           poly(BO.5km_ras.s,3) +
                           poly(S.COV5km_ras.s,3) +
                           poly(S.AVG5km_ras.s,3) +
                           poly(dissurf.s,3),
                           weights=training.df.day[[i]]$weights,
                     family=binomial(link="cloglog"), data = training.df.day[[i]])

      save(m, file = paste(output_path,"survey_simple_glm.m_", i, ".RData", sep = ""))

      ### calculate evaluation metrics for each model and rbind in dataframe
      df <- data.frame(int.AUC = c(NA), ext.AUC = c(NA), diff.AUC = c(NA),
                   thresh = c(NA), TSS = c(NA), sensitivity_argos = c(NA))
      # predictions to evaluation and training
      pred.ext <- predict(m, evaluation.df.day[[i]], type = "response")
      pred.int <- predict(m, training.df.day[[i]], type = "response")
      # AUC = Threshold-independent metric of performance
      df$int.AUC <- round(roc(training.df.day[[i]]$presence, pred.int)$auc,3)
      df$ext.AUC <- round(roc(evaluation.df.day[[i]]$presence, pred.ext)$auc,3)
      df$diff.AUC <- round(df$int.AUC - df$ext.AUC, 3)
      # TSS = threshold-dependent metric calculated on confusion matrix from pred.ext
      thresh <- SDMTools::optim.thresh(obs = evaluation.df.day[[i]]$presence,
              pred = pred.ext, threshold = 1001)$`max.sensitivity+specificity`[1]
      df$thresh <- thresh
      mat <- SDMTools::confusion.matrix(obs = evaluation.df.day[[i]]$presence,
                               pred = pred.ext, threshold = thresh)
      sen <- SDMTools::sensitivity(mat)
      spe <- SDMTools::specificity(mat)
      df$TSS <- sen + spe - 1
      # Using the treshold calculated above on pred.ext,
      # I predict sensitivity of predictions to argos positions
      pred.argos <- predict(m, eez_argos_df_eval, type = "response")
      # calculate percent well classified as presences
      df$sensitivity_argos <- round(length(pred.argos[pred.argos > thresh])/
                              length(pred.argos)*100,2)
      rm(m)
  return(df)
})
save(mod_cv, file = paste(output_path, "model_glm_cv.RData", sep = ""))

######################################
### AVERAGE PERFORMANCE ACROSS CV RUNS

mean <- apply(mod_cv[2:ncol(mod_cv)], 2, function(x) {mean(x, na.rm=T)})
sd <- apply(mod_cv[2:ncol(mod_cv)], 2, function(x) {sd(x, na.rm=T)})
model_summary <- data.frame(rbind(mean, sd))
model_summary <- round(model_summary, 3)
model_summary$value <- c("mean","sd")

write.csv(model_summary, file = paste(output_path, "summary_glm_cv.csv",
                                   sep = ""), row.names = F)
```

```r
######################################
### VARIABLE CONTRIBUTIONS

# load each model one by one and extract the contribution % for each predictor
load(paste(output_path, "survey_simple_glm.m_1.RData", sep=""))
contribution <- caret::varImp(m, scale = TRUE)
contribution$var <- row.names(contribution)
contribution$var <- aaply(contribution$var, .margins = 1, function(x){
  strsplit(substr(x, 6, nchar(x)), ",")[[1]][1] })
contribution_sum <- ddply(contribution, ~var, function(d){
  sum(d$Overall)
})
contribution_sum$V1 <- contribution_sum$V1/sum(contribution_sum$V1)*100
for (i in c(2:nb_cv)){
  load(paste(output_path, "survey_simple_glm.m_",i,".RData", sep=""))
  contribution <- caret::varImp(m, scale = TRUE)
  contribution$var <- row.names(contribution)
  contribution$var <- aaply(contribution$var, .margins = 1, function(x){
    strsplit(substr(x, 6, nchar(x)), ",")[[1]][1] })
  newcontribution <- ddply(contribution, ~var, function(d){
    sum(d$Overall)
  })
  newcontribution$V1 <- newcontribution$V1/sum(newcontribution$V1)*100
  contribution_sum <- merge(contribution_sum, newcontribution, by = "var")
}

save(contribution_sum, file = paste(output_path, "full_contributions_glm.RData", sep=""))

# mean contribution of each predictor
contribution <- ddply(contribution_sum, ~var, function(d) {
  dd <- data.frame(mean = round(apply(d[,2:ncol(d)], 1, function(w) {
                              mean(w, na.rm=T)}),3),
                   sd = round(apply(d[,2:ncol(d)], 1, function(w) {
                              sd(w, na.rm=T)}), 3))
  return(dd)
})

write.csv(contribution, file = paste(output_path, "mean_contributions_glm.csv",
                                     sep = ""), row.names = F)
```

2- Partial dependence plots

This function calls all CV runs of a set of models (specified in stats_names, e.g., GLM, GAM, BRT etc.), then it averages the response to each predictor (with all other predictors fixed to their means) and calculates the first and third quantiles. The loop returns a list of dataframes (one per statistical approach), in which each element of the list is a dataframe containing the mean fit of the model across 50 cv runs and the quantiles.

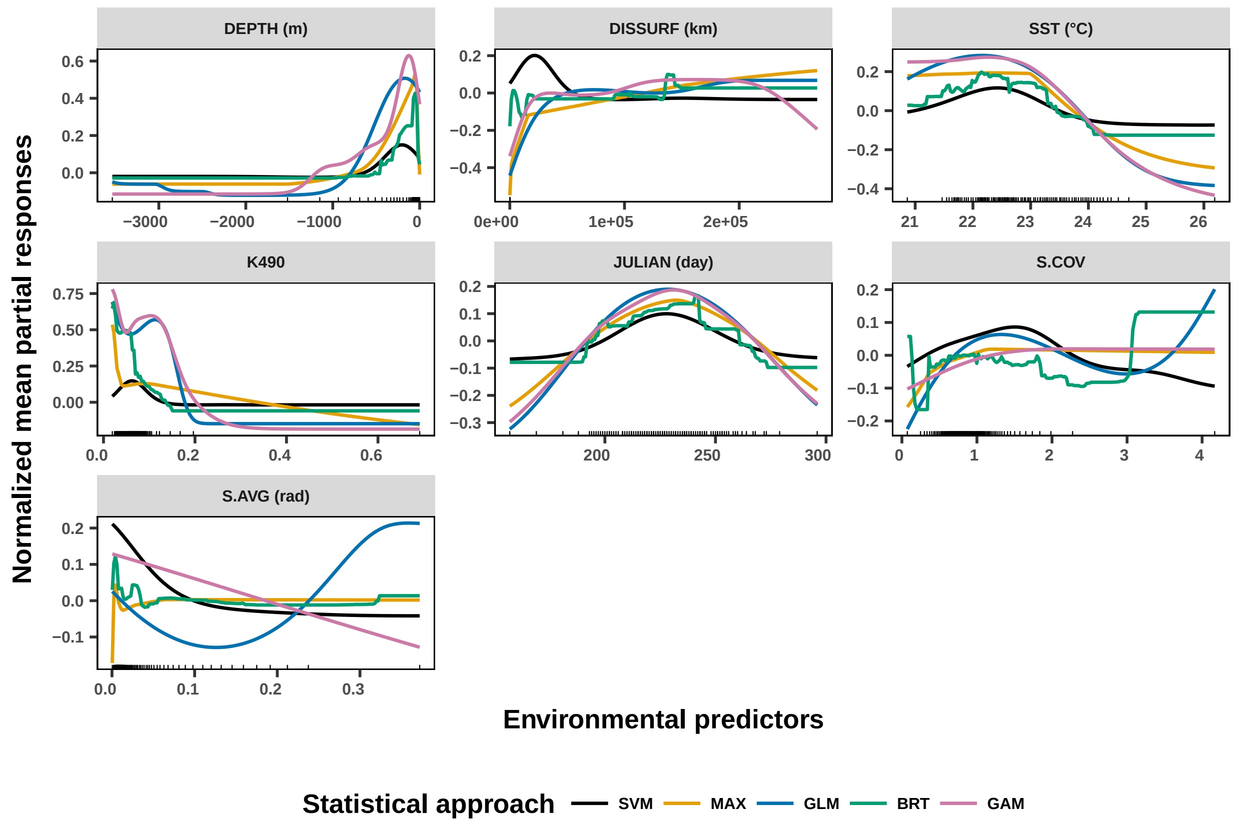
Calculating fitted responses

```r
# list the names of the models we are interested in
stats_names <- list("survey_simple_brt.m_lr0.005_tc3_",
                    "survey_simple_brt.m_lr0.005_tc1_",
                    "survey_simple_max.m_ftlqh_bbeta1_",
                    "survey_simple_max.m_ftl_bbeta1_",
                    "survey_simple_glm.m_",
                    "survey_simple_gam.m_",
                    "survey_simple_svm.m_radial0_cost10_",
                    "survey_simple_svm.m_linear0_cost0.01_")
names(stats_names) <- c("brt_pred","brt_stable", "max_pred", "max_stable",
                        "glm", "gam", "svm_pred", "svm_stable")

Fun_response <- function(model_name, fulldata){
  pred_cv <- adply(c(1:50), .margins = 1, .id = "cv", function(i){
    print(i)
    load(paste(output_path, model_name, i, ".RData", sep=""))
    # responses are calculated over the entire dataset (not just the training in each cv)
    mean_pred_df <- data.frame(sst_m.s = rep(mean(fulldata$sst_m.s),200),
                               julian.s = mean(fulldata$julian.s),
                               B0.5km_ras.s = mean(fulldata$B0.5km_ras.s),
                               S.COV5km_ras.s = mean(fulldata$S.COV5km_ras.s),
                               S.AVG5km_ras.s = mean(fulldata$S.AVG5km_ras.s),
                               dissurf.s = mean(fulldata$dissurf.s),
                               k490_m.s = mean(fulldata$k490_m.s))
    # for each predictor, calculate response with model m
    new_pred <- adply(names(mean_pred_df), .margin = 1, function (var){
      mean_pred_df[var] <- seq(min(fulldata[var]), max(fulldata[var]), length.out = 200)
      if (class(m)[1] == "gbm") {
                pred <- predict(m, mean_pred_df, type = "response",
                                n.trees = m$gbm.call$best.trees) }
      if(class(m)[1] == "svm.formula") {
                pred <- predict(m, mean_pred_df, probability = T)
                pred <- attr(pred, "probabilities")[,"1"] }
      if(class(m)[1] != "svm.formula" & class(m)[1] != "gbm") {
                pred <- dismo::predict(m, mean_pred_df, type = "response",
                                se.fit = F) }
      new_pred_df <- data.frame(x = mean_pred_df[,var])
      new_pred_df$x <- new_pred_df$x *
        sd(fulldata[,strsplit(var, split=".s", fixed=T)[[1]][1]]) +
        mean(fulldata[,strsplit(var, split=".s", fixed=T)[[1]][1]])
      new_pred_df$predvar <- var
      new_pred_df$fit <- pred
      return(new_pred_df)
    })
    return(new_pred)
  })
  av_pred_df <- ddply(pred_cv, .(x, predvar), function(d){
    dd <- data.frame(Q1rst = quantile(d$fit, na.rm=T)[2],
                     Q3rst = quantile(d$fit, na.rm=T)[4],
                     meanfit = mean(d$fit, na.rm=T),
                     medianfit = median(d$fit, na.rm=T))
    return(dd)
  })
  return(av_pred_df)
}


# the function Fun_response is run on each statistical approach:
# BRT, MAXENT, GLM, SVM and GAM and all responses dataframes are
# returned in a single list
all_responses <- llply(stats_names, function(s){
  responses <- Fun_response(s, PAsurvey_df)
  return(responses)
})

names(all_responses) <- c("brt_pred","brt_stable", "max_pred", "max_stable",
                          "glm", "gam", "svm_pred", "svm_stable")
```

Creating mean Partial Dependence Plots

```r
# Convert the list of response dataframes to one large dataframe
all_responses_df <- ldply(all_responses, function(d) {return(d)})

# Normalize responses so that they appear all on the same scale
all_responses_df <- ddply(all_responses_df, .(.id, predvar), function(d){
            d$meanfit_normalized <- (d$meanfit - mean(d$meanfit))
            return(d)
})

# For the plot, remove the STABLE tunings (focus on 'PREDICTIVE' models tunings)
all_responses_df <- all_responses_df[!(all_responses_df$.id %in%
                                         c("svm_stable", "brt_stable", "max_stable")),]

# this will change labels of facets
labels <- c(julian.s = "JULIAN (day)", B0.5km_ras.s = "DEPTH (m)",
            dissurf.s = "DISSURF (km)", S.AVG5km_ras.s = "S.AVG (rad)",
            S.COV5km_ras.s = "S.COV", sst_m.s = "SST (°C)", k490_m.s = "K490")

# this will change the order in which elements are plotted
all_responses_df$predvar <- factor(all_responses_df$predvar, levels =
                               c("B0.5km_ras.s", "dissurf.s", "sst_m.s", "k490_m.s",
                                 "julian.s", "S.COV5km_ras.s", "S.AVG5km_ras.s"))
all_responses_df$.id <- factor(all_responses_df$.id, levels =
                                c("svm_pred", "max_pred",
                                  "glm", "brt_pred", "gam"))

# distribution of values displayed in the rug plots
rugs <- tidyr::gather(PAsurvey_df[c("dissurf.s","B0.5km_ras.s", "sst_m.s", "k490_m.s",
                                  "S.COV5km_ras.s", "julian.s", "S.AVG5km_ras.s")],
                      key = predvar,
                      value = x,
                      dissurf.s, B0.5km_ras.s, sst_m.s, k490_m.s,
                                  S.COV5km_ras.s, julian.s, S.AVG5km_ras.s)
rugs$predvar <- factor(rugs$predvar, levels =
                                c("B0.5km_ras.s", "dissurf.s", "sst_m.s", "k490_m.s",
                                 "julian.s", "S.COV5km_ras.s", "S.AVG5km_ras.s"))
```

```r
rugs <- ddply(rugs, ~ predvar, function(dd){
    if (dd$predvar[1] == "logdissurf"){
      dd$values <- (exp(dd$values) - 1)/1000 #in km
    }
    if (dd$predvar[1] == "logS.AVG5km_ras"){
      dd$values <- exp(dd$values)
    }
    if (dd$predvar[1] == "logB0.5km_ras"){
      dd$values <- 1- exp(dd$values)
    }
    return(dd)
})

# calculate centiles over the full dataset of presence/control points
rugs_centiles <- ddply(rugs, ~ predvar, function(d){
  q <- quantile(d$x, probs = seq(0, 1, 0.01))
  dd <- data.frame(x = c(q), predvar = rep(d$predvar[1], length(q)))
  return(dd)
})

# rescale variables to their original values
PAsurvey_df$day <- NULL
rugs_centiles <- ddply(rugs_centiles, ~ predvar, function(d){
    var <- strsplit(as.character(d$predvar[1]), split = ".s", fixed = T)[[1]][1]
    pa <- PAsurvey_df[, var]
    d$x <- d$x * sd(pa) + mean(pa)
    return(d)
})

# rescale dissurf to km
rugs_centiles[rugs_centiles$predvar == "dissurf.s", ]$x <-
  rugs_centiles[rugs_centiles$predvar == "dissurf.s", ]$x/1000

# ggplot theme for plot
mon_theme <- theme(axis.text.x  = element_text(angle = 0, vjust = 0.8, hjust = 0.8),
                  #orientate the labels on the side when they are dates
        panel.border = element_rect(size=0.5,color="black", fill="transparent"),
                  #white background
        plot.margin=unit(c(2,2,2,2),"mm"),
                  #small margins, first number is top then go clockwise
        panel.background = element_rect(fill = 'white'),
        text=element_text(face="bold", size=8),
                  #general size for all text.looks small here but ok once using ggsave
                  #with 600 dpi and default pointsize
        title = element_text(size=rel(1.2)))
                  # all titles will be a little larger than labels

ggplot(all_responses_df, aes(x, meanfit_normalized)) +
      geom_line(aes(col = .id), size = 0.6) +
      scale_colour_manual(values = c("black", "#E69F00", "#0072B2", "#009E73", "#CC79A7"),
                         name = "Statistical approach",
                         labels =  c("SVM","MAX", "GLM", "BRT", "GAM")) +
      geom_rug(data = rugs_centiles, aes(x = x), sides="b", inherit.aes = F,
               colour = "black", size = 0.2) +
      facet_wrap(~ predvar, scales = "free", ncol = 3,
                 labeller = labeller(predvar = labels)) +
      xlab("Environmental predictors") +
      ylab("Normalized mean partial responses") +
      mon_theme +
      theme(legend.box.just = "bottom",
            legend.position = "bottom",
            legend.direction = "horizontal",
            legend.key = element_blank())
```



3- Maps of predicted habitat suitability

Producing mean predicted maps

Averaged over the 50 cross validation runs of each statistical approach

First, the environmental rasters have to be scaled and centered on the same scale (mean and sd) than the presence/control dataset on which the models were built. For dynamic variables (sst and k490), the average rasters calculated over all austral winters between 2003 and 2016 are used as predictors.

```
var <- list("sst", "k490", "BO.5km_ras", "S.COV5km_ras", "S.AVG5km_ras", "dissurf")

env <- env_stack
env_PAsurvey_scaled <- llply(var, function(x){
    # for the climatological variables we need to rename them to the names _m
    # 'known' by the models
    if (x == "sst" | x == "k490"){x2 <- paste(x, "_m", sep="")}
    # for all other variables we will simply add a .s at the end of the name
    else { x2 <- x}
    # rescale raster
    r <- (env[[x]] - mean(PAsurvey_df[,x2]))/sd(PAsurvey_df[,x2])
    names(r) <- paste(x2, ".s", sep="")
    return(r)
})
# stack all rasters back together
env_PAsurvey_scaled <- stack(env_PAsurvey_scaled)
# add julian day rescaled
env_PAsurvey_scaled$julian.s <- (240 - mean(PAsurvey_df$julian))/sd(PAsurvey_df$julian)
# aggregate to speed up the predictions
env_PAsurvey_scaled <- aggregate(env_PAsurvey_scaled, fact = 2)
```

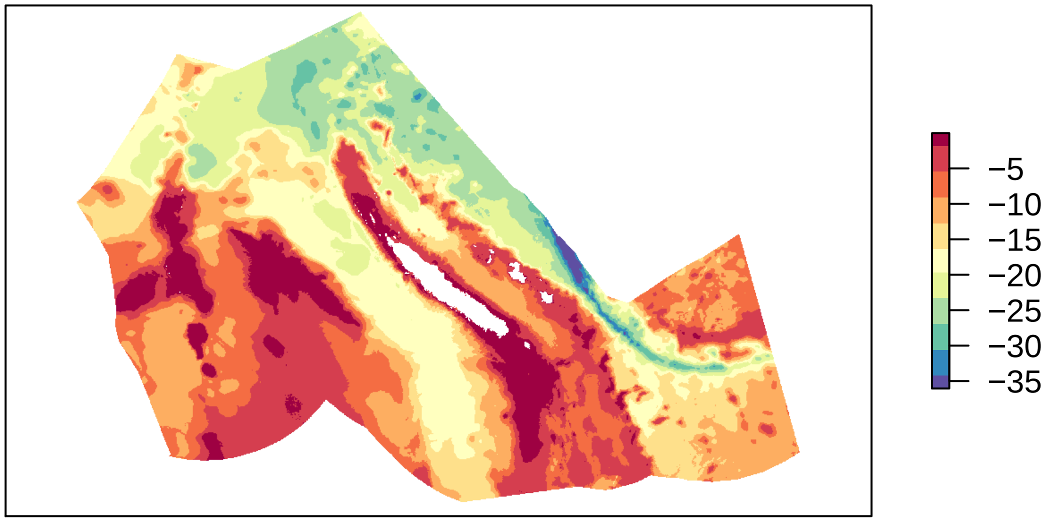
The probability of presence is predicted directly on the rescaled rasters for each CV run. Then all predicted layers are averaged per statistical algorithm. Example for GAM:

```
load(paste(output_path, "survey_simple_gam.m_1.RData", sep=""))
pred_ras <- list(predict(env_PAsurvey_scaled, m, type = "response"))
for (i in c(2:nb_cv)){
  load(paste(output_path, "survey_simple_gam.m_",i,".RData", sep=""))
  pred_ras <- c(pred_ras, list(predict(env_PAsurvey_scaled, m, type = "response")))
}

pred_stack <- stack(pred_ras)
pred_mean=calc(pred_stack, function(x){mean(x, na.rm=T)})

writeRaster(pred_mean, filename = paste(output_path, "pred_mean_gam_simple.grd",
                                        sep = ""), overwrite=T)
```

The **pred_mean** rasters produced for each of the statistical approach may then be plotted within R or exported to another format. For instance, for the GAM map of habitat suitability:

```
plot(pred_mean_gam_simple, col = rev(brewer.pal(11, name = "Spectral")),
     main = "GAM mean predicted habitat suitability (log-scale)", axes=FALSE)
```

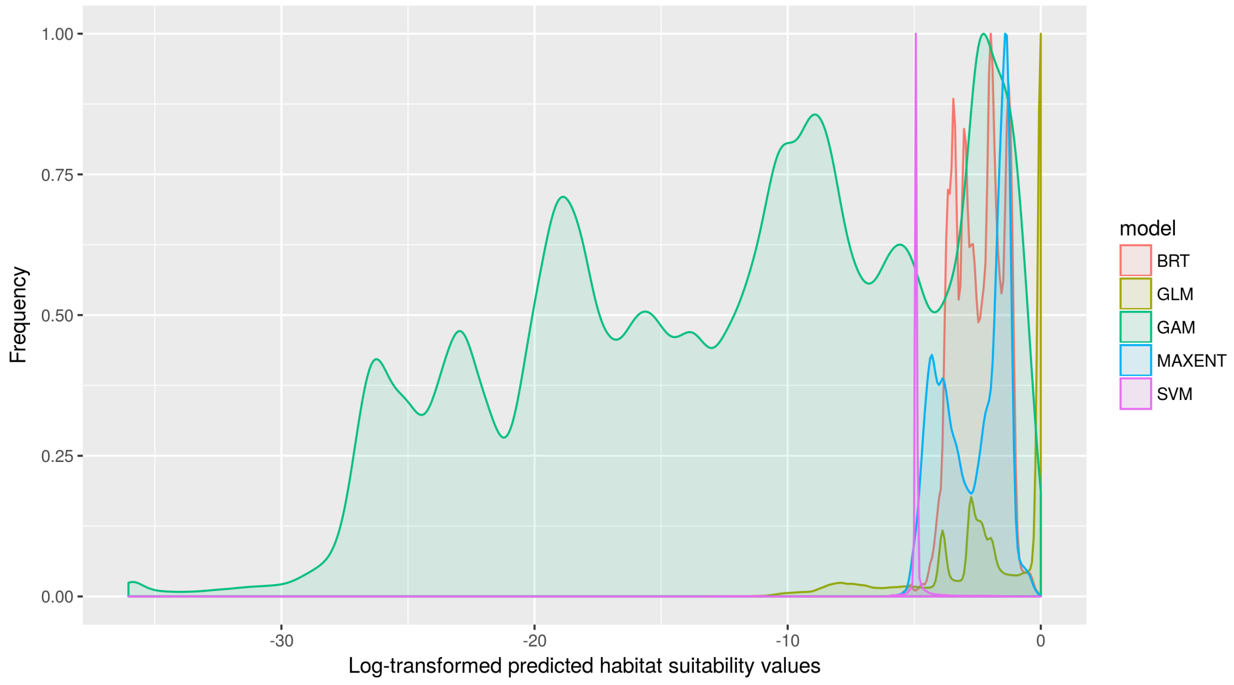
GAM mean predicted habitat suitability (log−scale)



Distribution of predicted values

Over the five maps of mean predicted habitat suitability

This plot allows us to visualize the probability distribution of predicted values (log-transformed) for each of the five statistical approaches and compare them together.



The predicted values for the SVM algorithm are extremely skewed.