```r
### copyright Dormann, C and Kaschner, K, additional material to MCED ###
#  when using, or when questions, please e-mail carsten.dormann@ufz.de, #
###             kristin.kaschner@biologie.uni-freiburg.de             ###

setwd("D:/Data/aktuell/2005_SpeciesDistributionAnalyses/marinemammals/")

require(MASS)

load("spermwale.Rdata") #SE
summary(SW)
attach(SW)


##############################################################################
#                        Example Analysis                                    #
##############################################################################
require(maptools)
data(wrld_simpl)
x11(width=12, height=4)
par(mar=c(0,0,0,0), oma=c(0,0,0,0))
plot(wrld_simpl, col="grey", ylim=c(-90,0), asp=1)  # only the Southern hemisphere
points(unique(SW[,c("CenterLong", "CenterLat")]), pch=15, cex=2.075, col=ifelse(SW$PAsperm == 0,
"grey90", "black")) # black is present!
dev.off()

# STEP 1: PRE-PROCESSING

## 1.1 Response variable: Binary, no further considerations.

## 1.2 Explanatory variables:
# Look at distribution of these variables and transform to maximise uniformity:
names(SW)
# [Comment: We know already that we want to use BRT and GLM. Hence we can transform the variables as a
first step. Using BRT alone would not require this step.]
# The first variable of interest is "AreaBelow100". The others before will not be used as such.
hist(AreaBelow100); summary(AreaBelow100)
# See what a Box-Cox-transformation would propose:
boxcox(lm(AreaBelow100+1 ~1)) #-1, i.e. the inverse:
hist((AreaBelow100^(0.8))) # does not look that much better
# -> stick to untransformed data

var <- Seamount
hist(var); summary(var)
boxcox(lm(var+2 ~1))
# all unmentioned variables remain untransformed
# "DepthMin"   -> consider splitting into <500m and >500m
# SSTAnMean -> log +1.8
# SalinityMean -> log
# PrimProdMean -> log
# LandDist -> sqrt
# Shelf -> log(x+1)
# Slope -> sqrt
# Abyssal -> ^0.25
# Seamount -> log(x+1)

detach(SW)
SW.t0 <- SW
# implement the transformations in new data set SW.t
SW.t0 <- SW
SW.t0$SSTAnMean <- log(SW$SSTAnMean + 1.8)
SW.t0$SalinityMean <- log(SW$SalinityMean)
SW.t0$PrimProdMean <- log(SW$PrimProdMean)
SW.t0$LandDist <- sqrt(SW$LandDist)
SW.t0$Shelf <- log(SW$Shelf + 1)
SW.t0$Slope <- sqrt(SW$Slope)
SW.t0$Abyssal <- SW$Abyssal^0.25
SW.t0$Seamount <- log(SW$Seamount + 1)

# Finally, we standardise all explanatory variables to a mean=0 and sd=1:
SW.t <- SW.t0
SW.t[,-c(1:4)] <- scale(SW.t0[,-c(1:4)])
```

```
## 1.3 Collinearity
# calculate a correlation matrix (round to 3 decimal places):
round(cor(SW.t[,-c(1:4)], use="complete.obs", method="kendall"),3)
# We can use a cluster representation to visualise collinearity:
require(Hmisc)
v <- as.formula(paste("~", names(SW.t)[-c(1,2)], collapse="+"))
par(mar=c(1,5,1,1))
plot(varclus(v, data=SW.t))  # correlation structure
# Several variables are unacceptably highly correlated (Spearman's rho^2 > 0.5). From those, we use
the one with higher ecological relevance/interpretability (or, if this is not possible, highest
importance in the randomForest model):
# DepthMean > DepthMax, Abyssal
# DepthMin > Shelf, IslandsNo, LandDist,
# SSTAnMean > IceConAnn, SSTMnMax, SSTAnSD, SSTMnRange, OceanArea, AreaBelow100
# SSTMnMin > SBTAnMean
# SalinityMean > SalinityMax


#This leads to a reduced data set:
SW.t.red <- SW.t[,c("PAsperm", "CsquareCode", "CenterLat", "CenterLong", "Slope", "DepthSD",
"Seamount", "DepthMean", "SalinityBMean", "DepthMin", "PrimProdMean", "SSTAnMean", "SSTMnMin",
"Coral", "Estuary", "SalinityMin", "SalinityMean")]

## 1.4 Dimensional reduction
require(randomForest)
f <- as.formula(paste("as.factor(PAsperm)~", paste(names(SW.t.red)[-c(1,2)], collapse="+"), sep=""))
rf <- randomForest(f, data=SW.t.red, na.action=na.omit, importance=T)
varImpPlot(rf) # top 5: "SSTAnMean" "SalinityBMean" "DepthSD" "SalinityMean" "DepthMean"
# Hastie et al. (2008, page 593) recommend using the Gini coefficient for importance
head(names(sort(importance(rf)[,4], decreasing=T)), 10) # gives the 10 most important variables

# Plot pairwise variable space (for easier interpretation, use untransformed data):
pair <- SW.t.red[,c("SSTAnMean", "SalinityBMean")]
plot(pair, pch="+", cex=2, col=rgb(.5,.5,.5,.5), las=1, tcl=0.5, main="Observational space of sea
surface temperature and salinity")
polygon(pair[chull(pair),], lwd=2, border="grey")
# The two are uncorrelated (r_kendall = 0.377), but clearly there are only few regions in the
parameter space that is actually covered, even inside the convex hull!

pair <- SW[,c("DepthSD", "DepthMean")] #(r_kendall= -0.238)
plot(pair, pch="+", cex=2, col=rgb(.5,.5,.5,.5), las=1, tcl=0.5, main="Observational space of mean
depth and its standard deviation", xlab="DepthSD [°C]")
polygon(pair[chull(pair),], lwd=2, border="grey")
# Now that looks much better!

pair <- SW[,c("SSTAnMean", "DepthMean")] #(r_kendall= 0.234)
par(mar=c(5,5,1,1))
plot(pair, pch="+", cex=2, col=rgb(.5,.5,.5,.5), las=1, tcl=0.5)
  #main="Observational space of mean depth and sea surface temperature"
polygon(pair[chull(pair),], lwd=2, border="grey")
# The majority of data points are from near Antarctica, hence sub-zero temperature. The trips to South
Africa, South America and Australia yield the high-temperature values.
# Notice that we log-transform SSTAnMean to spread the low values and shrink the high:
pair[,1] <- log(pair[,1]+1.8)
plot(pair, pch="+", cex=2, col=rgb(.5,.5,.5,.5), las=1, tcl=0.5,
 #main="Observational space of mean depth and log(sea surface temperature",
 xlab="log(SSTAnMean+1.8)")
polygon(pair[chull(pair),], lwd=2, border="grey")
# The effect is remarkable and should convince anyone sceptical of transformations of the explanatory
variable!

# [...] continue for other combinations


# STEP 2: MODELLING

## 2.1 Boosted Regression Trees
# first, include helper R-code from Elith et al. (2008):
source("brtfunctions.r")
# BRT offers various specifications; crucial ones are
## tree.complexity (don't use high values, e.g. > 5)
```

```
## learning.rate (the lower the slower; good values are 0.01 to 0.001)
## bag fraction (the proportion of data points used for fitting)
# For defaults type: fix(gbm.step) and read the paper!
set.seed(20202)
f.brt <- gbm.step(data=SW.t.red, gbm.x = 5:17, gbm.y = 1, family = "bernoulli",  tree.complexity = 3,
learning.rate = 0.01, bag.fraction = 0.5, verbose=TRUE, silent=FALSE, plot.main=TRUE)
# The graph returned depicts the development of residual variance in the validation data sets: the
lower, the better. At some point there are too many trees, and the BRT overfits the data (the lines go
up again). This point (indicated by a vertical green line) is the BRT-set used.
summary(f.brt)    # returns importances; last four are irrelevant
gbm.plot(f.brt, smooth=T, n.plots=8, write.title = F) # shows partial plots

## 2.2 GLM
table(SW.t.red$PAsperm)
# 0   1
#146 115
# Since we have an effective sample size of 115, and we want to have at least 10 data points support
per variable (events per variable, EPV=10), we allow only 10 effects into the GLM.
# Since we know (from the partial BRT plots) that variable effects are non-linear, we also include
quadratic terms, as well as interactions.
# However, we start with a more complex model and reduce it to the desired complexity:
source("COLL_allfunctions.r")
f <- formula.maker(SW.t.red[,c("PAsperm", "SSTAnMean", "SalinityBMean", "DepthSD", "SalinityMean",
"PrimProdMean")])
fm <- glm(f, data=SW.t.red, family=binomial)
anova(fm, test="Chisq")  # just a quick look at the first, overfitted model
# Since the function for SSTAnMean looked very non-linear in the BRT plots, lets add a third-order
polynomial:
fm <- update(fm, .~. + I(SSTAnMean^3))


## 2.3 Model simplification
fm.red <- stepAIC(fm, k=log(115))
anova(fm.red, test="Chisq")
# From the first parameter-space plot we know that the interaction of SSTAnMean and SalinityBMean
cannot be supported by data. We thus manually delete this interaction:
fm.red2 <- update(fm.red, .~. - SSTAnMean:SalinityBMean)
anova(fm.red2, test="Chisq")
# We cannot remove main effects when they are part of a significant interaction!
# A quadratic effect without the main effect is "allowed", but strange. Let's put the main effect in,
too:
fm.red3 <- update(fm.red2, .~.-I(DepthSD^2)+poly(DepthSD,2)) # not significant
anova(fm.red3, test="Chisq")
# Was it worth it?
anova(fm.red3, fm.red2, test="Chisq")
# No, models 4 and 5 are indistinguishable, so we use the simpler (4) as our FINAL MODEL.
# Let's do the same for SSTAnMean:
fm.red4 <- update(fm.red2, .~.-I(SSTAnMean^3)-SSTAnMean+poly(SSTAnMean,3)) # not significant
anova(fm.red4, test="Chisq")
# Was it worth it?
anova(fm.red4, fm.red2, test="Chisq")
# No evidence! Still, for reasons of elegance and beauty, let's stick to the "proper" polynomial
effect of SSTAnMean! (This is to show that there is always a level of arbitration in statistical
modelling!)


# 2.4 Model diagnostics

# Time to look at our models.
# First, we want to get a feeling whether the error distribution has been handled properly:
par(mfrow=c(1,2))
hist(residuals(f.brt))
hist(residuals(fm.red4))
# In both cases, residuals clump around two values (1/-1). This indicates that a substantial part of
the data were estimated as absent when present (leading to positive values) or vice versa (negative
values). The BRT is more "categorical" in its predictions, making the residuals larger.
# Note that these values are at the "link-scale", and hence a value of 1 represents a probability of
plogis(1) # 0.73.
# But: are they good?
# Model fit is given by the reduction in deviance:
summary(fm.red4)
# An intercept-only model ("null model") has a deviance of 64.3, while our final model reduces this to
```

```
57.6 - a moderate decrease. Expressing this as R2 is tricky, there are many different R2s for GLMs. A
pseudo-R2 (also named D2) can easily be calculated as:
(358.1-284.8)/358.1   # 0.204, i.e. 20% explained deviance.
# In GLMs, explained deviance in a good model rarely exceeds 0.3, but 0.2 is not a yet a really good
model.
# Often we use AUC to express the ability of a model to disciminate between 0s and 1s.
require(verification)
# For the GLM, AUC is:
roc.area(obs=SW.t.red$PAsperm, pred=predict(fm.red4, type="response")) #0.78
# For the BRT, we can calculate the same (but need to give some more information):
roc.area(obs=SW.t.red$PAsperm, pred=predict(f.brt, newdata=SW.t.red, type="response", n.trees=200))
#0.883
# However, BRT also returns this value when fitting (scroll in your R-window or repeat the BRT
analysis and check).
# training data ROC score = 0.883
# Additionally, it give the average cross-validation AUC-value, which is a much better indication of
the model's predictive performance (and usually much lower):
# cv ROC score = 0.677 ; se = 0.037
# Is is also extractable using:
mean(f.brt$cv.roc.matrix)
# This drop in AUC from training to cross-validation is typical for weak models.

# Another thing to inspect binomial (and poisson) GLMs for is overdispersion. An estimate of
dispersion is calculated by dividing the residual deviance by the residual degrees of freedom:
284.8/252 #1.13
#High values (say, > 4 or so) indicate a problem. Choose "quasibinomial" as family (which yields no
AIC and can hence not be used with stepAIC).
# To summarise model fits:
# * Models are not spectactularly good, but not all that bad either.
# * Predictive performance for models with an AUC around 0.8 is usually deemed "moderate".
# * BRT: Annual mean sea surface temperature and salinity at the bottom are most relevant.
# * GLM: Annual mean sea surface temperature and also bottom salinity are most relevant.
# At least the two models are somewhat consistent!

## Spatial Autocorrelation
require(ncf)
# use the untransformed lat-long-data:
# To get a feeling for the distances, make a histogram of Euclidean distances (which is wrong, because
the earth is a sphere where -175° is close to +175° and no distance can be greater than 180*sqrt(2).
We are here only interested in the spacing and the min distance.)
hist(d <- dist(SW[,3:4]))
min(d)
# shortest distance between two cells is 5, the histogram proposes steps of 20
cor.fglm <- correlog(x=SW$CenterLong, y=SW$CenterLat, z=residuals(fm.red4), increment=20, resamp=999,
latlon=T) # takes a while (1 min or so)
plot(cor.fglm)
abline(h=0)
# Notice that lat-long-data are converted into km! (internally, using gcdist)
# This plot is a bit cluttered and "long". Distance-classes over about 400 units (i.e. 400 * 20 = 8000
km) contain only few data points:
cor.fglm$n
#Moreover, it is unlikely that biological mechanisms lead to separation or aggregation of sperm wales
at distances of more than, say, 1000 km. We can thus truncate the plot at 1000/20 = 50.
plot(cor.fglm$mean.of.class[1:50], cor.fglm$correlation[1:50], type="n", lwd=2, col="grey",
xlab="distance [km]", ylab="Moran's I", las=1, tcl=0.5)
abline(h=0)
lines(cor.fglm$mean.of.class[1:50], cor.fglm$correlation[1:50], lwd=2, col="grey")
points(cor.fglm$mean.of.class[1:50], cor.fglm$correlation[1:50], pch=16, col=ifelse(cor.fglm$p<0.05,
"black", "grey90"), cex=1.5)
# What we see is a significant positive spatial autocorrelation at 200km and again at around 500,
620-640, ... and a negative SAC at 820 km.
# We shall now try to account for this pattern. From experience, we would not expect to remove all of
this pattern, but primarily the peak at 200 km, and perhaps those around 600-800 km.
# The method we use is called "spatial eigenvector mapping" or "principal coordinates of neighbourhood
matrix".
require(spdep)
# First, we construct a list, which contains the neighbours' IDs for each data point. There are
several ways to do so. Check, e.g., ?knearneigh, ?dnearneigh, ?gabrielneigh
# Note that coords must be given as long-lat, not lat-long! (Which is consistent with plotting, but
not with common usage.) d1/2 is in km!
SW.nb <- dnearneigh(as.matrix(SW[,4:3]), d1=0, d2=2000, longlat=T)
summary(SW.nb)
```

```
ME.fit <- ME(fm.red4$formula, listw=nb2listw(SW.nb), data=SW.t.red, alpha=0.5)
SW.t.red <- cbind(SW.t.red, fitted(ME.fit))
f <- as.formula(paste("PAsperm ~ ", fm.red4$formula[3], "+ vec1+vec3", sep=""))
fm.red4.me <- glm(f, data=SW.t.red, family=binomial)
anova(fm.red4.me)
# So, the spatial dimension has almost as much relevance as the SSTAnMean-effect.
# Let's compare model coefficients to see whether these were affected by spatial autocorrelation:
summary(fm.red4)
summary(fm.red4.me)
# Indeed! Errors became smaller and parameters changed by tens of percent!
# Finally, let us investigate whether this new model has lower spatial autocorrelation in its
residuals:
cor.fglm.me <- correlog(x=SW$CenterLong, y=SW$CenterLat, z=residuals(fm.red4.me), increment=20,
resamp=999, latlon=T) # takes a while (1 min or so)
# add line to previous correlogram:
lines(cor.fglm.me$mean.of.class[1:50], cor.fglm.me$correlation[1:50], lwd=1, col="black")
points(cor.fglm.me$mean.of.class[1:50], cor.fglm.me$correlation[1:50], pch=16, col=ifelse(cor.fglm
$p<0.05, "black", "grey90"), cex=1)
# Little difference overall, main effect at 200 and 600 km and at the funny peak at 1400 km.
# Just for fun, let's have a look what the eigenvector 1 looks like on a map:
require(lattice)

levelplot(SW.t.red$vec1 ~ SW$CenterLong+SW$CenterLat, aspect="iso")
# This eigenvector thus codes for some effect that is high around Australia (+/-180°) and low around
South Africa (around 0°).
levelplot(SW.t.red$vec3 ~ SW$CenterLong+SW$CenterLat, aspect="iso")
# Vec 3 in contrasts codes a much more fine-scaled pattern of unknown origin.


# 2.5 Interpretation

# Make plots of functional relationships.
# 1. Back-transform predictor variables (un-standardise, then un-transform):
# SSTAnMean:
SSTAnMean.t <- function(x) exp(x * sd(SW.t0$SSTAnMean, na.rm=T) + mean(SW.t0$SSTAnMean, na.rm=T)) - 1.8

x11(width=15, height=8)
par(oma=c(0,5,1,0), mar=c(5,2,0,.5), mfrow=c(1,2))
# SSTAnMean - transformed:
  xlims=c(-2.25,2)
  # BRT
  partial <- plot.gbm(f.brt, i.var="SSTAnMean", return.grid=T)
  partial.t <- partial
  partial.t[,1] <- SSTAnMean.t(partial[,1])
  plot(partial.t, type="n", main="", ylab="", xlim=xlims, xlab="transformed(SSTAnMean)", ylim=c(0,1),
yaxs="i", las=1, xaxs="r", cex.lab=1.55, tcl=0.5)
  newx <- seq(min(SW.t.red$SSTAnMean, na.rm=T), max(SW.t.red$SSTAnMean, na.rm=T), len=100)
  lines(newx, predict.gbm(f.brt, newdata=newdata2, n.trees=250, type="response"), col="grey40", lwd=2,
type="s")
  # GLM
  newdata1 <- as.data.frame(t(apply(SW.t.red[,-c(1:4)], 2, mean, na.rm=T)))
  newdata2 <- cbind.data.frame(SSTAnMean=newx, newdata1[,-8], vec1=0, vec3=0,vec259=0)
  p <- predict.glm(fm.red4.me, newdata=newdata2, se.fit=T)
  matlines(newx, plogis(cbind(p$fit, p$fit-2*p$se.fit, p$fit+2*p$se.fit)), col="grey", lwd=2, lty=c
(1,2,2))
  rug(SW.t$SSTAnMean[SW.t$PAsperm==0], col=rgb(.4,.4,.4,.4))
  rug(SW.t$SSTAnMean[SW.t$PAsperm==1], side=3, col=rgb(.4,.4,.4,.4))


# SSTAnMean - backtransformed:
  xlims=c(-2,25)
  # BRT
  partial <- plot.gbm(f.brt, i.var="SSTAnMean", return.grid=T)
  partial.t <- partial
  partial.t[,2] <- plogis(partial[,2]) # back-transform to response scale
  partial.t[,1] <- SSTAnMean.t(partial[,1])
  plot(partial.t, type="n", main="", ylab="", xlim=xlims, xlab="SSTAnMean [°C]", ylim=c(0,1),
yaxs="i", las=1, xaxs="r", cex.lab=1.55, tcl=0.5)
  newx <- seq(min(SW.t.red$SSTAnMean, na.rm=T), max(SW.t.red$SSTAnMean, na.rm=T), len=100)
  lines(SSTAnMean.t(newx), predict.gbm(f.brt, newdata=newdata2, n.trees=250, type="response"),
col="grey40", lwd=2, type="s")
  # GLM
```

```
  newdata1 <- as.data.frame(t(apply(SW.t.red[,-c(1:4)], 2, mean, na.rm=T)))
  newdata2 <- cbind.data.frame(SSTAnMean=newx, newdata1[,-8])
  p <- predict.glm(fm.red4, newdata=newdata2, se.fit=T)
  matlines(SSTAnMean.t(newx), plogis(cbind(p$fit, p$fit-2*p$se.fit, p$fit+2*p$se.fit)), col="grey",
lwd=2, lty=c(1,2,2))
  rug(SW$SSTAnMean[SW$PAsperm==0], col=rgb(.4,.4,.4,.4))
  rug(SW$SSTAnMean[SW$PAsperm==1], side=3, col=rgb(.4,.4,.4,.4))

mtext(side=2, "probability of occurrence", cex=1.5, outer=T, line=2)



# As an illustration that variables in an interaction should not be studied on their own!
# SalinityBMean
# define the unstandardisation function for convenience:
SBMunscale <- function(x) x*sd(SW$SalinityBMean, na.rm=T) + mean(SW$SalinityBMean, na.rm=T)
par(mar=c(5,5,4,1))
#  xlims=c(-7.7,5.4)
  # BRT
  partial <- plot.gbm(f.brt, i.var="SalinityBMean", return.grid=T)
  partial.t <- partial
  partial.t[,2] <- plogis(partial[,2]) # back-transform to response scale
  partial.t[,1] <- SBMunscale(partial.t[,1])# un-standardise
  #partial.t[,1] <- SSTAnMean.t(partial[,1])
  plot(partial.t, type="n", ylab="probability of occurrence", xlab="SalinityBMean [\u2030]", ylim=c
(0,1), yaxs="i", las=1, xaxs="r", cex.lab=1.55, tcl=0.5, main="Example for a step change")
  newx <- seq(min(SW.t.red$SalinityBMean, na.rm=T), max(SW.t.red$SalinityBMean, na.rm=T), len=100)
  newdata1 <- as.data.frame(t(apply(SW.t.red[,-c(1:4)], 2, mean, na.rm=T)))
  newdata2 <- cbind.data.frame(SalinityBMean=newx, newdata1[,-5], vec1=0, vec3=0,vec259=0)
  lines(SBMunscale(newx), predict.gbm(f.brt, newdata=newdata2, n.trees=250, type="response"),
col="grey40", lwd=2, type="s")
  # GLM
  p <- predict.glm(fm.red4.me, newdata=newdata2, se.fit=T)
  matlines(SBMunscale(newx), plogis(cbind(p$fit, p$fit-2*p$se.fit, p$fit+2*p$se.fit)), col="grey",
lwd=2, lty=c(1,2,2))
  rug(SBMunscale(SW.t.red$SalinityBMean[SW.t$PAsperm==0]), col=rgb(.4,.4,.4,.4))
  rug(SBMunscale(SW.t.red$SalinityBMean[SW.t$PAsperm==1]), col=rgb(.4,.4,.4,.4), side=3)


# Same in the interaction with PrimProdMean:
# define the unstandardisation-untransformation function for PrimProd for convenience:
PPunscale <- function(x) exp(x*sd(SW.t0$PrimProdMean, na.rm=T) + mean(SW.t0$PrimProdMean, na.rm=T))

newx <- seq(min(SW.t.red$SalinityBMean, na.rm=T), max(SW.t.red$SalinityBMean, na.rm=T), len=50)
newx2 <- seq(min(SW.t.red$PrimProdMean, na.rm=T), max(SW.t.red$PrimProdMean, na.rm=T), len=50)
newdata1a <- expand.grid(SalinityBMean=newx, PrimProdMean=newx2)
newdata1b <- as.data.frame(t(apply(SW.t.red[,-c(1:4)], 2, mean, na.rm=T)))
newdata2 <- cbind.data.frame(newdata1a, newdata1[,-c(5,7)], vec1=0, vec3=0,vec259=0)

x11(width=15, height=8)
par(oma=c(5,5,1,0), mar=c(2,2,1,1), mfrow=c(1,2))
# for BRT:
pred.fun.brt <- function(x,y){
    predict.gbm(f.brt, newdata=cbind.data.frame(SalinityBMean=x, PrimProdMean=y, newdata1b[,-c(5,7)]),
type="response", n.trees=300)
}
z.brt <- outer(newx, newx2, pred.fun.brt)
image(SBMunscale(newx), PPunscale(newx2), z.brt, xlab="", ylab="", las=1)
contour(SBMunscale(newx), PPunscale(newx2), z.brt, add=T)
polygon(SW[chull(cbind(SW$SalinityBMean, SW$PrimProdMean)),c("SalinityBMean","PrimProdMean")], lwd=2,
border="grey")
points(SBMunscale(SW.t$SalinityBMean), PPunscale(SW.t$PrimProdMean), pch=ifelse(SW.t$PAsperm==1, "+",
"x"), cex=1.5, col=rgb(.4,.4,.4,.6))
rug(SBMunscale(SW.t$SalinityBMean[SW.t$PAsperm==0]), col=rgb(.5,.6,.7,.4), side=3)
rug(PPunscale(SW.t$PrimProdMean[SW.t$PAsperm==0]), col=rgb(.5,.6,.7,.4), side=4)
legend("topright", "BRT", cex=2, bty="n")

# same for GLM:
pred.fun.glm <- function(x,y){
    predict(fm.red4.me, newdata=cbind.data.frame(SalinityBMean=x, PrimProdMean=y, newdata1b[,-c(5,7)]),
type="response")
}
```

```
z.glm <- outer(newx, newx2, pred.fun.glm)
image(SBMunscale(newx), PPunscale(newx2), z.glm, xlab="", ylab="", las=1)
contour(SBMunscale(newx), PPunscale(newx2), z.glm, add=T)
polygon(SW[chull(cbind(SW$SalinityBMean, SW$PrimProdMean)),c("SalinityBMean","PrimProdMean")], lwd=2,
border="grey")
points(SBMunscale(SW.t$SalinityBMean), PPunscale(SW.t$PrimProdMean), pch=ifelse(SW.t$PAsperm==1, "+",
"x"), cex=1.5, col=rgb(.4,.4,.4,.6))
rug(SBMunscale(SW.t$SalinityBMean[SW.t$PAsperm==0]), col=rgb(.5,.6,.7,.4), side=3)
rug(PPunscale(SW.t$PrimProdMean[SW.t$PAsperm==0]), col=rgb(.5,.6,.7,.4), side=4)
legend("topright", "GLM", cex=2, bty="n")

mtext(side=1, text="sea bottom salinity [\u2030]", cex=1.5, outer=T, line=2)
mtext(side=2, text=expression(paste("Primary Productivity [", mgC*m^{-2} * d^{-1}, "]", sep="")),
cex=1.5, outer=T, line=2)


# plot model predictions from BRT and GLM:
brt.preds <- predict.gbm(f.brt, newdata=SW.t,n.trees=300, type="response")
glm.preds <- predict(fm.red4.me, type="response")

x11(width=12, height=8)
par(mfrow=c(2,1), mar=c(0,0,0,0), oma=c(1,1,1,1))
# BRT
plot(wrld_simpl, col="grey", ylim=c(-90,0), asp=1)  # only the Southern hemisphere
points(unique(SW[,c("CenterLong", "CenterLat")]), pch=16, cex=0.5,
  col=ifelse(SW$PAsperm == 0, "grey80", "black"))
points(unique(SW[,c("CenterLong", "CenterLat")]), pch=15, cex=2.075,
  col=rgb(0,0,0,brt.preds))
legend("topleft", "BRT", cex=2, bty="n")

# GLM
plot(wrld_simpl, col="grey", ylim=c(-90,0), asp=1)  # only the Southern hemisphere
points(unique(SW[,c("CenterLong", "CenterLat")]), pch=16, cex=0.5,
  col=ifelse(SW$PAsperm == 0, "grey80", "black"))
points(unique(SW[,c("CenterLong", "CenterLat")]), pch=15, cex=2.075,
  col=rgb(0,0,0,glm.preds))
legend("topleft", "GLM", cex=2, bty="n")

dev.off()



# Final comments:
# 1. If you have many steps in your analysis, and you are uncertain if they may lead to spurious
results, use a null model to find out. In its simplest form, simply shuffle the response variable
(sample(SW$PAsperm) in our case) and re-run the analysis 1000 times. Obviously you then have to
automatise the entire procedure.
# 2. In a more realistic setting, you may want to shuffle the response but maintain the spatial
structure. This can be done, too: see Beale et al. (2008) for methods and R-code.
# 3. If you want to extrapolate beyond the geographic region or the parameter space, everything
becomes rather uncertain. Note the 95% CI in the above plots, and how the wide towards the lower and
upper limits of the variable's range! The spatial eigenvectors (SV) are contructed in a way that their
mean effect is 0. When you want to intra- or extrapolate, you have to make a "map" of the eigenvectors
and use kriging to extrapolate in space to unobserved sites.




#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------

#source("brtbootstrap.r")
#test <- gbm.bootstrap(f.brt, n.reps=100)  # approx. 2 minutes
#gbm.plot.cis(f.brt, bootpreds=test$function.dataframe, plot.layout=c(2,3))


# do some real bootstrapping:
nrep <- 30
bs.list <- list()
for (i in 1: nrep){
```

```
 f.brt.bs <- try(gbm.step(data=SW.t.red[sample(NROW(SW.t.red), NROW(SW.t.red), replace=T),], gbm.x =
5:17, gbm.y = 1, family = "bernoulli",  tree.complexity = 3, learning.rate = 0.01, bag.fraction = 0.5,
verbose=TRUE, silent=FALSE, plot.main=TRUE))
  if (class(f.brt.bs) == "try-error") next
  bs.list[[i]] <- f.brt.bs
  cat("This is replicate number ", i, "\n")
  save(bs.list, file="bs.list.Rdata")
  rm(f.brt.bs)
}

brt.ext <- function(x){
  ss <- summary(x)
  oo <- order(ss[,1])
  out <- ss[oo,2]
  names(out) <- ss[oo,1]
  out
}
brt.ext(bs.list[[1]])
aa <- sapply(bs.list, brt.ext)
summary(t(aa))
boxplot(t(aa))


CI.mat <- matrix(ncol=nrep, nrow=100)
for (i in 1:nrep) CI.mat[,i] <- predict.gbm(bs.list[[i]], newdata=newdata2, n.trees=2500,
type="response")
CIs <- apply(CI.mat, 1, quantile, c(1-0.683, 0.5, 0.683), na.rm=T)

  xlims=c(-2.25,1.8)
  # BRT
  partial <- plot.gbm(f.brt, i.var="SSTAnMean", return.grid=T)
  partial.t <- partial
  partial.t[,1] <- SSTAnMean.t(partial[,1])
  plot(partial.t, type="n", main="", ylab="", xlim=xlims, xlab="transformed(SSTAnMean)", ylim=c(0,1),
yaxs="i", las=1, xaxs="r", cex.lab=1.55, tcl=0.5)
  newx <- seq(min(SW.t.red$SSTAnMean, na.rm=T), max(SW.t.red$SSTAnMean, na.rm=T), len=100)
  #bootstrap lines:
  matlines(newx, t(CIs), col=rgb(.2,.2,.2,.4), lwd=2, type="s", lty=c(2,1,2))
#  lines(newx, predict.gbm(f.brt, newdata=newdata2, n.trees=2500, type="response"), col="grey30",
lwd=2, type="s")

  # GLM
  newdata1 <- as.data.frame(t(apply(SW.t.red[,-c(1:4)], 2, mean, na.rm=T)))
  newdata2 <- cbind.data.frame(SSTAnMean=newx, newdata1[,-8])
  p <- predict.glm(fm.red4, newdata=newdata2, se.fit=T)
  matlines(newx, plogis(cbind(p$fit, p$fit-p$se.fit, p$fit+p$se.fit)), col="grey", lwd=2, lty=c(1,2,2))
  rug(SW.t$SSTAnMean[SW.t$PAsperm==0], col=rgb(.4,.4,.4,.4))
  rug(SW.t$SSTAnMean[SW.t$PAsperm==1], side=3, col=rgb(.4,.4,.4,.4))
```