# Appendix A for: A Novel Framework to Predict Relative Habitat Selection in Aquatic Systems: Applying Machine Learning and Resource Selection Functions to Acoustic Telemetry Data From Multiple Shark Species

via: Griffin LP, Casselberry GA, Hart KM, Jordaan A, Becker SL, Novak AJ, DeAngelis BM, Pollock CG, Lundgren I, Hillis-Starr Z, Danylchuk AJ and Skomal GB (2021) A Novel Framework to Predict Relative Habitat Selection in Aquatic Systems: Applying Machine Learning and Resource Selection Functions to Acoustic Telemetry Data From Multiple Shark Species. Front. Mar. Sci. 8:631262. doi: 10.3389/fmars.2021.63126

## Introduction

This vignette is to provide an overview and framework to perform resource selection functions (RSFs) with machine learning algorithms and acoustic telemetry data. This vignette is meant to illustrate each step needed to derive and predict relative selection values for a given species that was monitored with acoustic telemetry.

For this example, acoustic telemetry data from eight juvenile lemon sharks (*Negaprion brevirostris*) that were monitored between 2013-2019 has already been filtered to remove false detections[1] and converted into centers of activity (COAs)[2] using 90-minute bins. All individual tagging data has been removed to preclude any misuse of these data. Tagging information may be made available upon reasonable request[3]. All code and data presented within this vignette are publicly available at https://github.com/lucaspgriffin.

**Steps provided:**

**1 - Defining available resource units and presence/background points**

**2 - Aggregating habitat information**

**3 - Applying RSFs with machine learning**

**4 - Performance, interpretation, and prediction**

> *All above steps will use the complete juvenile lemon shark data between 2013-2019 that was used within the manuscript and that was compiled outside of this vignette. For brevity, steps 1-2 will be implemented for one year as to provide an example.*

[1]Simpfendorfer, C. A., C. Huveneers, A. Steckenreuter, K. Tattersall, X. Hoenner, R. Harcourt, and M. R. Heupel. 2015. Ghosts in the data: false detections in VEMCO pulse position modulation acoustic telemetry monitoring equipment. Animal Biotelemetry 3:55.

[2]Campbell, H. A., M. E. Watts, R. G. Dwyer, and C. E. Franklin. 2012. V-Track: Software for analysing and visualising animal movement from acoustic telemetry detections. Marine and Freshwater Research 63:815–820.

[3]Contact: Grace Casselberry at grace.casselberry@gmail.com

### Load packages and data

These analyses require many packages. You may need to install these on your local machine in order to run all code in this worksheet. Use the **pacman package** to: check, install, and load needed packages.

- Use **library(devtools)** and **install_github("trinker/pacman")** if the **pacman package** is needed on your local machine.
- Load needed data to run vignette. See Costa et al. (2012) for additional information on habitat classifications and data collection methods[4].
- Type ??*function* into your R console when more information is desired on any used function below.

```
library(pacman)
pacman::p_load(here, tidyr, dplyr, ggplot2, raster, rgeos, tmap, mlr, ranger, iml, pdp)

# Animal position data.
# Object: ObsPts. Constructed lemon shark COA SpatialPointsDataFrame. Steps 1-2.
load(here("data", "ObsPts.RData"))
# Object: rf_Lemon_juv. Fully compiled data frame for resource selection functions. Steps 3-4.
load(here("data", "rf_Lemon_juv.RData"))

# Station position data.
# Object: receiver_spdf. Station locations for 2013.
load(here("data", "receiver_spdf.RData"))

# Habitat, boundary, and study area data.
# Object: shapes. Habitat SpatialPolygonsDataFrame derived via Costa et al. (2012).
load(here("data", "shapes.RData"))
# Object: MPA_shape. Marine protected area SpatialPolygonsDataFrame SpatialPolygonsDataFrame.
load(here("data", "MPA_shape.RData"))
# Object: Land_shape. Land SpatialPolygonsDataFrame.
load(here("data", "Land.RData"))
# Object: r. Study area raster, 200 x 200 m pixels.
load(here("data", "r.RData"))
#Object: hab_r. Habitat raster brick, 200 x 200 m pixels.
load(here("data", "hab_r.RData"))
```

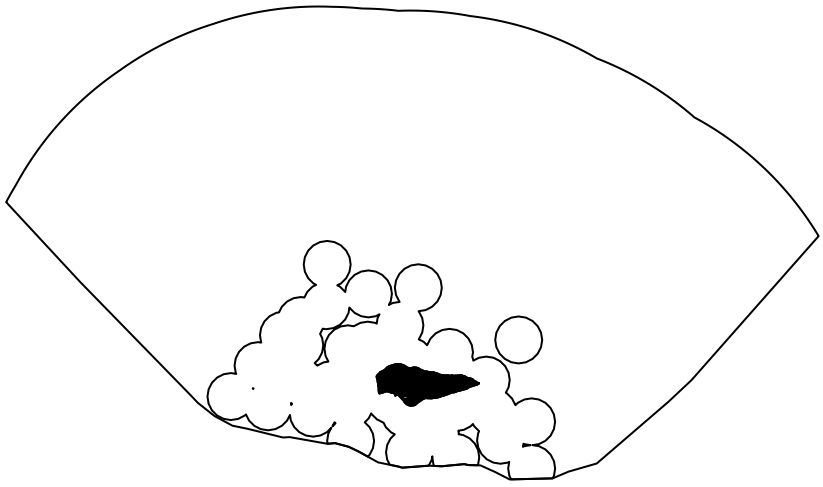
# 1 - Defining available resource units and presence/background points

Define available resource units (i.e., 400 m receiver buffers) for each year and extract data within boundaries.

```
# Designate overlapping 400 m areas around each station in 2013.

# Generate buffer around each station.
receiver_spdf_buffer <- gBuffer(receiver_spdf, width = 400)
# Merge buffers if overlapping / touching.
receiver_spdf_buffer2 <- gUnaryUnion(receiver_spdf_buffer)
# Crop buffer areas outside study area (i.e., outside of Buck
```

[4]Costa, B. M., S. Tormey, and T. A. Battista. 2012. Costa, B.M., S. Tormey, and T.A. Battista. Benthic Habitats of Buck Island Reef National Monument. NOAA Technical Memorandum NOS NCCOS 142. Prepared by the NCCOS Center for Coastal Monitoring and Assessment Biogeography Branch. Silver Spring, MD. 64.

```
# Island National Reef Monument).
receiver_spdf_buffer3 <- gIntersection(receiver_spdf_buffer2, MPA_shape,
    byid = TRUE)
# Remove land from buffer areas to produce the final set of
# available resource units for 2013.
avail_resource <- gDifference(receiver_spdf_buffer3, Land_shape)

# Plot the available resource units for 2013.
plot(MPA_shape, col = "white")
plot(avail_resource, add = TRUE)
plot(Land_shape, add = TRUE, col = "black")
```



```
# Use the over function to extract data position estimates
# (i.e., COAs) within the available resource unit boundary.Also,
# remove all points outside the defined available resource unit
# area usng the command '!is.na'.
ObsPts <- ObsPts[!is.na(over(ObsPts, avail_resource)), ]
```

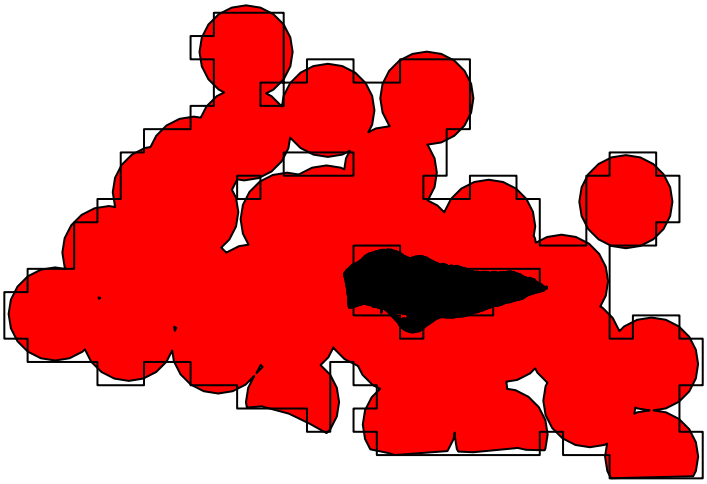
Prior to using a loop to randomly distribute background points (i.e., pseudo-absences), generate the needed list that compiles an equal to the number of observed COAs per individual and year across all available resource units.

```
# Generate a list for indexing within the loop: number of COAs
# derived for each transmitter and diel period during 2013.
```

```r
COAs_Trans_Diel_2013_list <- as.data.frame(ObsPts) %>% group_by(Transmitter,
    Year, Diel) %>% # Calculate the number of occurrences with this transmitter,
# year, diel combination.
summarise(count = length(Transmitter)) %>% ungroup() %>% # Combine the columns as unique ID.
mutate(TYD = paste0(Transmitter, "_", Year, "_", Diel)) %>% # Select only 2013.
filter(Year == 2013) %>% # Make into list based on TYD.
group_split(TYD)

# Generate a data frame with number of observed COAs across each
# date per transmitter, year, and diel period combination. This
# will be used to assign dates across the background points.
# Within the loop, it will sample from this distribution of
# dates and assign dates to the random points.  This step is
# relevant when examining RSFs with additional temporal
# components.

COAs_Trans_Diel_Date_2013 <- as.data.frame(ObsPts) %>% group_by(Transmitter,
    Year, Diel, DayDate) %>% # Calculate the number of occurrences with this transmitter,
# year, diel, date combination.
summarise(count = length(Transmitter)) %>% ungroup() %>% # Select only 2013.
filter(Year == 2013)

# Define available resource units for background point
# randomization.  Generate a SpatialPolygonsDataFrame from the
# 2013 array grid matching the study area raster (r).
r
```

```
## class      : RasterLayer
## dimensions : 34, 61, 2074  (nrow, ncol, ncell)
## resolution : 200, 200  (x, y)
## extent     : 322957.5, 335157.5, 1965746, 1972546  (xmin, xmax, ymin, ymax)
## crs        : NA
## source     : memory
## names      : layer
## values     : NA, NA  (min, max)
```

```r
rp_2013 <- r
# Fill new raster with NAs.
rp_2013[] <- NA
# Extract raster cell numbers.
my.pts <- as.data.frame(raster::extract(r, avail_resource, cellnumbers = TRUE))
# Only select the cell numbers.
my.pts <- my.pts[, 1]
rp_2013[my.pts] <- 1
# Convert raster to a SpatialPolygonsDataFrame for the upcoming
# assignment of random background points.
P_2013 <- rasterToPolygons(rp_2013, dissolve = TRUE)

# Plot.
plot(avail_resource, col = "red")
plot(P_2013, add = TRUE)
plot(Land_shape, add = TRUE, col = "black")
```



```
# List for placing results into.
SPDF_Combo_2013_list <- list()
```

Run randomization process and merge random and observed point data frames together.

```
for (i in 1:length(COAs_Trans_Diel_2013_list)) {
    # For reproducibility.
    set.seed(1234)

    # Distribute x number of points across defined available
    # resource unit for this particular transmitter, year, diel
    # period combination.
    randLocs <- spsample(P_2013, n = COAs_Trans_Diel_2013_list[[i]]$count,
        type = "random")

    set.seed(1234)
    # Get and randomize coordinates.
    coords <- randLocs@coords
    # Randomize coordinates.
    coords.random <- coords[sample(1:nrow(coords)), ]
    # Make a data frame that matches the number of COAs for that
    # individual.
    df <- do.call("rbind", COAs_Trans_Diel_2013_list[i])
    # Replicate the info to match the observed.
    df2 <- rbind(df, df[rep(1, (df$count - 1)), ])
```

```
    # Delete row names.
    rownames(df2) <- NULL
    df2$x <- coords.random[, 1]
    # Put the coordinates from the random sample into the data
    # frame.
    df2$y <- coords.random[, 2]
    # Label these detections are background points (0) (opposed to
    # observed COAs (1)).
    df2$RealDets <- 0

    # Put 'date' back into the data frame randomly and based on the
    # distribution of the observed data set.
    Select_Trans_df <- COAs_Trans_Diel_Date_2013 %>% filter(Transmitter ==
        unique(df2$Transmitter))
    set.seed(1234)
    df2$DayDate <- sample(Select_Trans_df$DayDate, size = length(df2$Transmitter),
        replace = TRUE, prob = Select_Trans_df$count)

    # Turn into a spdf with correct projection.
    SPDF_Combo <- SpatialPointsDataFrame(coords.random, df2, proj4string = CRS(proj4string(randLocs)))

    # Place completed iteration into a list.
    SPDF_Combo_2013_list[[i]] <- SPDF_Combo
}

# Random points for 2013.  Turn results list into a data frame.
RandomPts_2013 <- do.call("rbind", SPDF_Combo_2013_list)
# Observed points for 2013
ObsPts_2013 <- ObsPts@data %>% filter(Year == 2013)

# Random background points match observed points for 2013.
dim(RandomPts_2013)
```

```
## [1] 343   9
```

```
dim(ObsPts_2013)
```

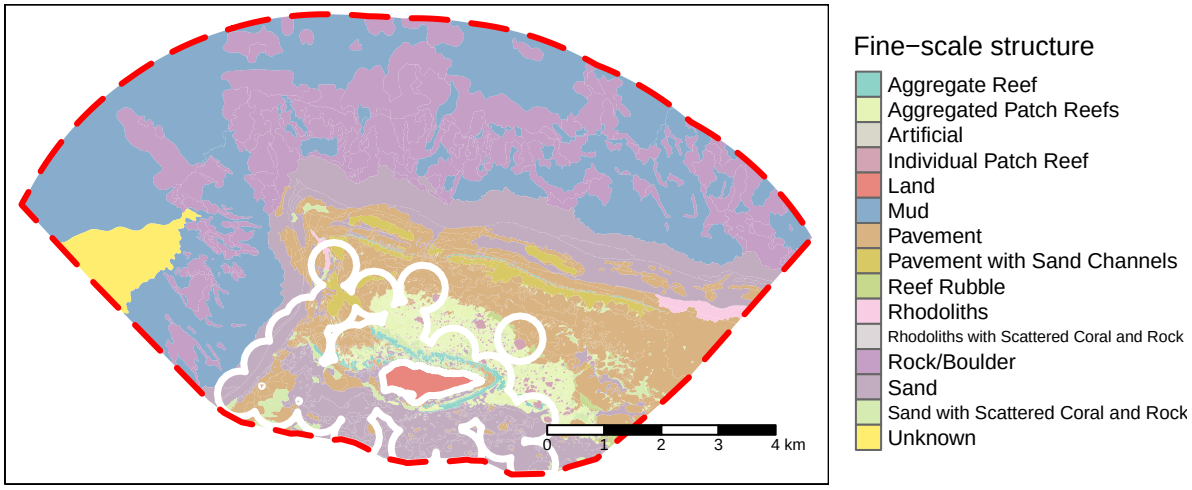
```
## [1] 343  11
```

```
# Ensure columns match.
ObsPts_2013 <- ObsPts_2013 %>% dplyr::select(-c(DateTime, DateTime.Loc,
    common_name, scientific_name)) %>% mutate(RealDets = 1)

RandomPts_2013 <- RandomPts_2013@data %>% dplyr::select(-c(TYD,
    count)) %>% mutate(Species = "Lemon")

# The combined random points and observed points from 2013.
RSF_df <- rbind(ObsPts_2013, RandomPts_2013)

# Turn into spdf again.
RSF_spdf <- SpatialPointsDataFrame(coords = cbind(RSF_df$x, RSF_df$y),
    RSF_df)
proj4string(RSF_spdf) <- CRS("+proj=utm +zone=20 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs")
```

## 2 - Aggregating habitat information

Extract the habitat raster data from the data frame. Here, we show how to extract the covariate, "fine-scale structure", as an example.

```
# Plot fine-scale structure (labeled as Det_Struct) across the MPA with the tmap package.
tm_basemap("Esri.WorldImagery") +
  tm_shape(shapes) +
  tm_polygons(c("Det_Struct"), title = "Fine-scale structure", style = "cat", lty = "blank") +
  tm_legend(outside = TRUE) +
  tm_shape(avail_resource) +
  tm_borders("white", lwd = 3.5) +
  tm_shape(MPA_shape) +
  tm_borders("red", lty = 5, lwd = 3) +
  tm_scale_bar()
```




```
# Extract.
rp.Det_Struct <- raster::rasterize(shapes, r, "Det_Struct")
# Remove "Land" classification.
values(rp.Det_Struct)[values(rp.Det_Struct) == 5] <- NA
# Remove "Unknown" classification.
values(rp.Det_Struct)[values(rp.Det_Struct) == 15] <- NA
# Assign correct projection.
crs(rp.Det_Struct) <-
  "+proj=utm +zone=20 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
```

```
# Use the extract function to derive values for each observation.
HabPts.Det_Struct <- raster::extract(rp.Det_Struct, # Raster layer.
  RSF_spdf, # Points as spdf
  df = TRUE
) # Return a data frame.
names(HabPts.Det_Struct) <- c("ID", "Det_Struct")

# Merge habitat information with original data frame.
# Make column to merge.
RSF_spdf$ID <- rep(1:length(RSF_spdf))
# Merge to have fine-scale structure data matched.
RSF_spdf <- merge(RSF_spdf, HabPts.Det_Struct, by = "ID")
```

## 3 - Applying RSFs with machine learning

Implement RSFs with random forest using the compiled lemon shark data frame from 2013-2019. For additional and detailed information on the procedures presented below, visit *https://geocompr.robinlovelace.net*[5].

```
# Create two separate data frames. The training dataset (60%)
# will be used to train our model and the testing dataset (40%)
# will be used to evaluate model performance.  Set a ID column
# for indexing.
rf_Lemon_juv$rowID <- seq(1:nrow(rf_Lemon_juv))
set.seed(1234)
# Randomly select 60% of the data frame for the training
# dataset.
rf_Lemon_juv.train <- rf_Lemon_juv[sample(1:nrow(rf_Lemon_juv),
    nrow(rf_Lemon_juv) * 0.6, replace = FALSE), ]
# Remainder (i.e., 40%) becomes the test dataset.
rf_Lemon_juv.test <- rf_Lemon_juv[!(rf_Lemon_juv$rowID %in% rf_Lemon_juv.train$rowID),
    ]

# Coordinates needed for the spatial partitioning.
coords.train_Lemon <- rf_Lemon_juv.train[, c("x", "y")]
coords.test_Lemon <- rf_Lemon_juv.test[, c("x", "y")]

rf_Lemon_juv.train <- dplyr::select(rf_Lemon_juv.train, -x, -y,
    -rowID)
rf_Lemon_juv.test <- dplyr::select(rf_Lemon_juv.test, -x, -y, -rowID)

# Set tasks for training and test datasets.
task.train_Lemon <- makeClassifTask(data = rf_Lemon_juv.train, target = "presf",
    positive = "pres", coordinates = coords.train_Lemon)

task.test_Lemon <- makeClassifTask(data = rf_Lemon_juv.test, target = "presf",
    positive = "pres", coordinates = coords.test_Lemon)

# Make learner.
lrn.Bin <- makeLearner(cl = "classif.ranger", predict.type = "prob")
```

[5]Lovelace, R., J. Nowosad, and J. Muenchow. 2019. Geocomputation with R. CRC Press.

```
# Partition spatially and set random search.
perf_level <- makeResampleDesc("SpCV", iters = 5)
ctrl <- makeTuneControlRandom(maxit = 50L)

# To find optimal hyperparamters, first construct parameter sets
# for iterations to search through.
ps <- makeParamSet(makeIntegerParam("mtry", lower = 1, upper = ncol(rf_Lemon_juv.train) -
    1), makeNumericParam("sample.fraction", lower = 0.2, upper = 0.9),
    makeIntegerParam("min.node.size", lower = 1, upper = 10))
```

Tune and find optimal hyperparamters. Warning, computationally extensive, this code will take multiple hours to run depending on local machine performance.

```
set.seed(1234)
tune_Lemon <- tuneParams(learner = lrn.Bin, task = task.train_Lemon,
    resampling = perf_level, par.set = ps, control = ctrl, measures = mlr::auc)
```

Set optimal hyperparameters and train model.

```
# Set learner again, this time with optimal hyperparameter
# combinations.
lrn_rf.Bin_Lemon <- setHyperPars(makeLearner("classif.ranger", importance = "impurity",
    predict.type = "prob"), par.vals = tune_Lemon$x)

# Train model.
set.seed(1234)
# Run model.
model_rf.Bin_Lemon <- train(lrn_rf.Bin_Lemon, task.train_Lemon)
# Extract results.
results_Lemon <- mlr::getLearnerModel(model_rf.Bin_Lemon)
```

# 4 - Performance, interpretation, and prediction

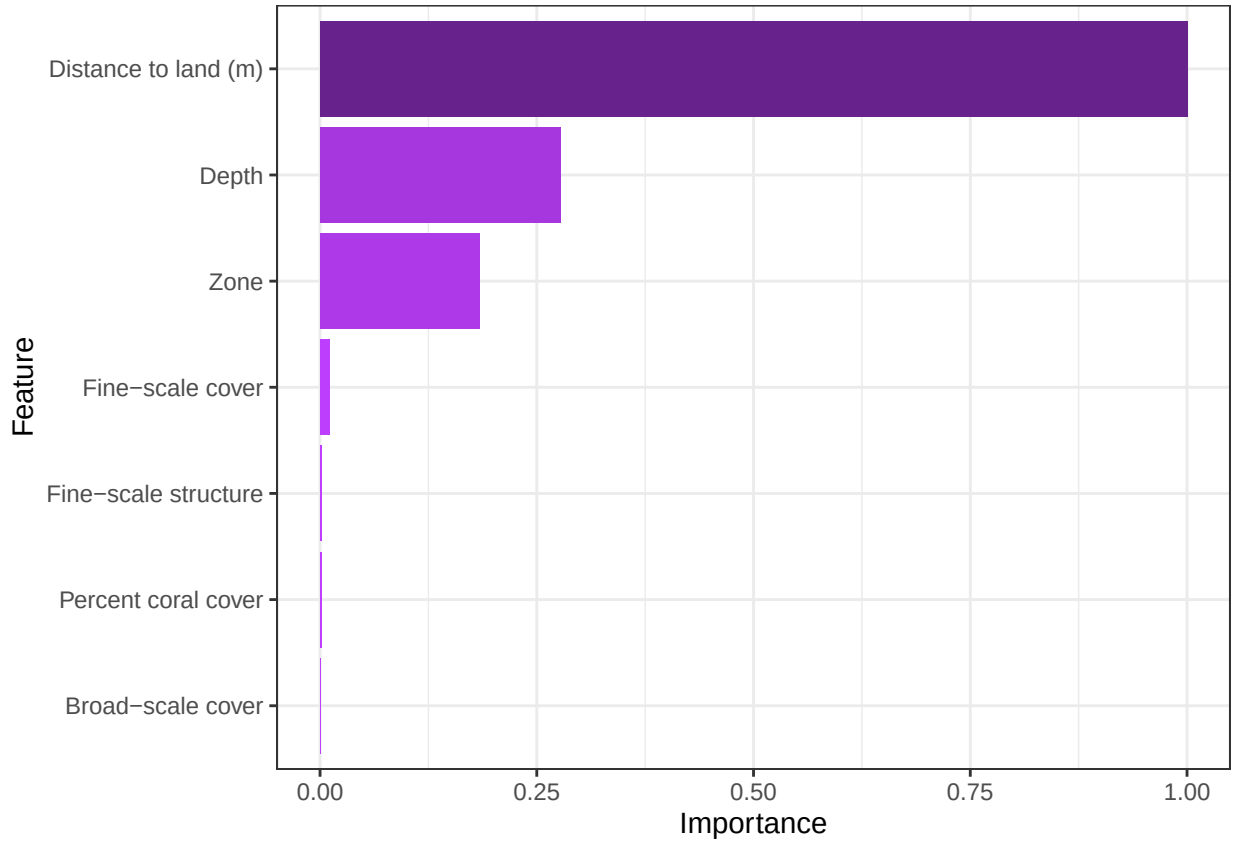
```
# Predictions.
set.seed(1234)
# Using the trained model, test it using the test dataset to
# determine performance
pred_Lemon <- predict(model_rf.Bin_Lemon, task = task.test_Lemon)

# Performance.
(performance_Lemon <- calculateROCMeasures(pred_Lemon))
```

```
##             predicted
## true         background pres
##   background 1253       129       tpr: 0.97  fnr: 0.03
##   pres       37         1327      fpr: 0.09  tnr: 0.91
##              ppv: 0.91  for: 0.03 lrp: 10.42 acc: 0.94
##              fdr: 0.09  npv: 0.97 lrm: 0.03  dor: 348.36
##
```

```
##
## Abbreviations:
## tpr - True positive rate (Sensitivity, Recall)
## fpr - False positive rate (Fall-out)
## fnr - False negative rate (Miss rate)
## tnr - True negative rate (Specificity)
## ppv - Positive predictive value (Precision)
## for - False omission rate
## lrp - Positive likelihood ratio (LR+)
## fdr - False discovery rate
## npv - Negative predictive value
## acc - Accuracy
## lrm - Negative likelihood ratio (LR-)
## dor - Diagnostic odds ratio
```

```
# Interpretation.
X_Lemon <- rf_Lemon_juv.train[which(names(rf_Lemon_juv.train) !=
    "presf")]
# Create 'Predictor' object to interpret findings via the iml
# package.
predictor_Lemon <- Predictor$new(model_rf.Bin_Lemon, data = X_Lemon,
    y = rf_Lemon_juv.train$presf)

imp_Lemon <- FeatureImp$new(predictor_Lemon, loss = "ce")  # Calculate importance.
# Construct new data frame and label all predictors with correct
# labels using an ifelse statement.
imp_df_Lemon <- imp_Lemon$results
imp_df_Lemon$feature_label <- ifelse(imp_df_Lemon$feature == "Det_Struct",
    "Fine-scale structure", ifelse(imp_df_Lemon$feature == "P_Hard",
        "Percent hard bottom", ifelse(imp_df_Lemon$feature == "P_Coral_Cv",
            "Percent coral cover", ifelse(imp_df_Lemon$feature ==
                "P_Maj_Cov", "Percent broad-scale cover", ifelse(imp_df_Lemon$feature ==
                "Maj_Struct", "Structure", ifelse(imp_df_Lemon$feature ==
                "Maj_Cover", "Broad-scale cover", ifelse(imp_df_Lemon$feature ==
                "Cover", "Fine-scale cover", ifelse(imp_df_Lemon$feature ==
                "Land_Dist", "Distance to land (m)", imp_df_Lemon$feature)))))))

# Importance
ordered <- imp_df_Lemon %>% arrange(desc(importance)) %>%
# Order from greatest to lowest importance values.
distinct(feature_label)
ordered_fin <- ordered$feature_label
# Re-order factor levels.
imp_df_Lemon$feature_label <- factor(imp_df_Lemon$feature_label,
    levels = rev(ordered_fin))

# Normalize importance with function.
normalize <- function(x) {
    return((x - min(x))/(max(x) - min(x)))
}

imp_df_Lemon$importance_norm <- normalize(imp_df_Lemon$importance)

ggplot(data = imp_df_Lemon, aes(y = feature_label, x = importance_norm,
```

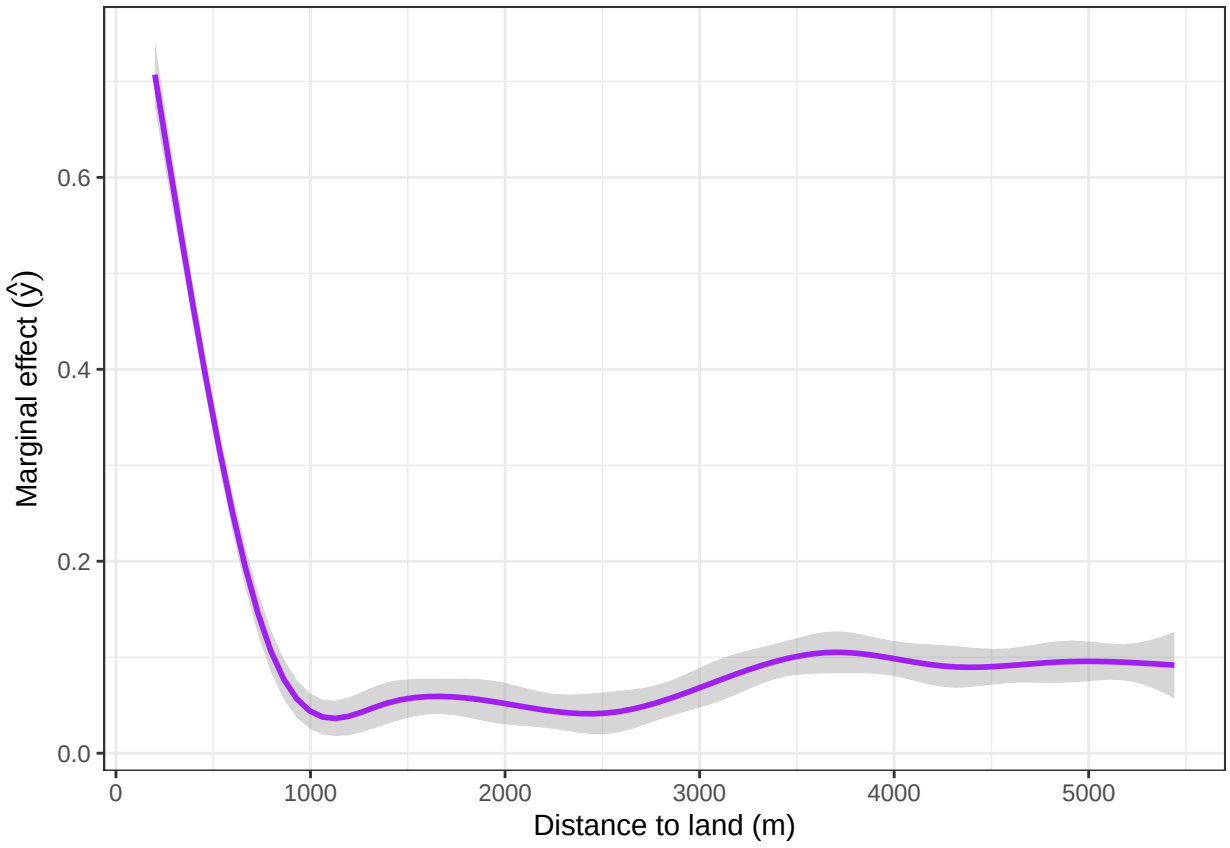
```
    fill = importance_norm)) + geom_bar(stat = "identity", position = "dodge") +
    scale_fill_gradient(low = "darkorchid1", high = "darkorchid4") +
    ylab("Feature") + xlab("Importance") + guides(fill = F) + theme_bw()
```



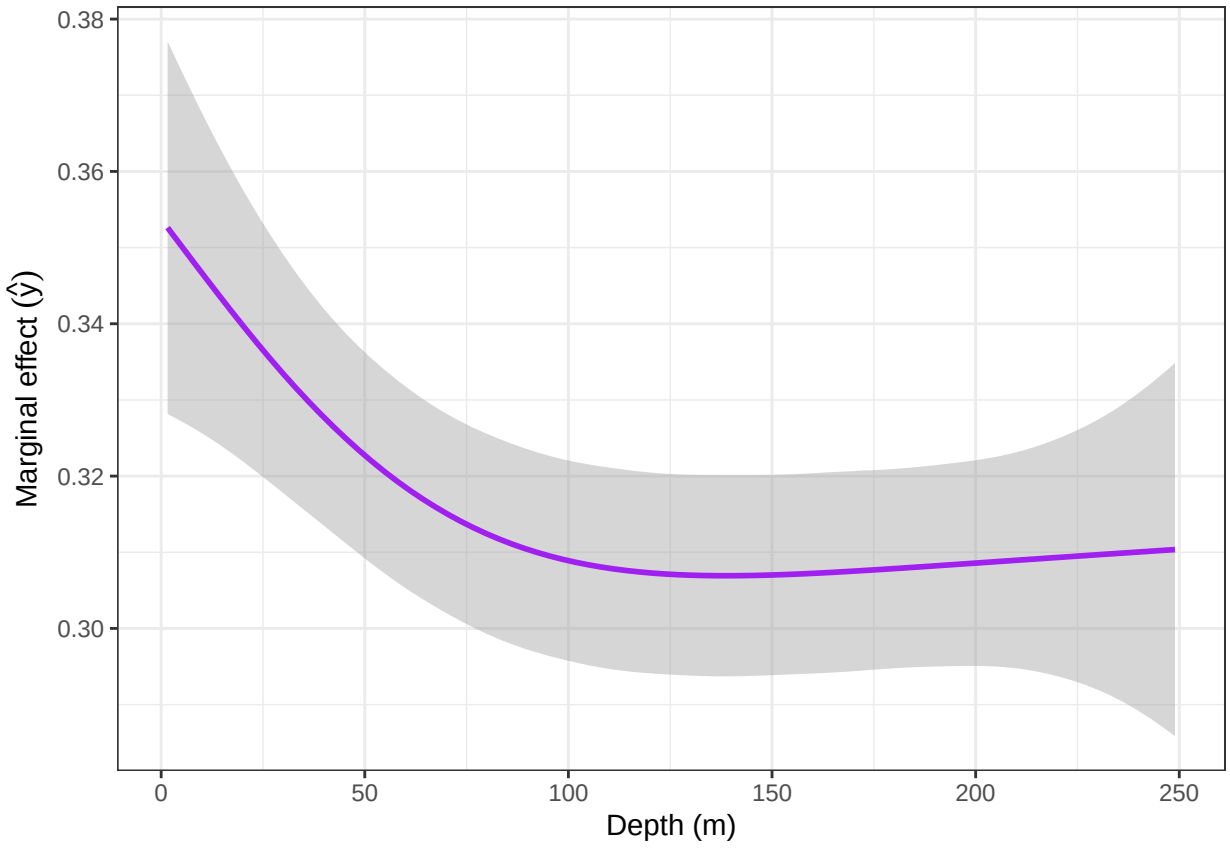

With most important predictors identified, calculate and plot the marginal effects of the top three via partial dependency plots (pdp): distance to land (Land_Dist), depth (Depth), zone (Zone).

```
# pdps
top_1_pdp_df_Lemon <- pdp::partial(results_Lemon, pred.var = imp_df_Lemon[1,
    1], prob = TRUE, train = rf_Lemon_juv.train, progress = "text",
    which.class = 2)

top_2_pdp_df_Lemon <- pdp::partial(results_Lemon, pred.var = imp_df_Lemon[2,
    1], prob = TRUE, train = rf_Lemon_juv.train, progress = "text",
    which.class = 2)

top_3_pdp_df_Lemon <- pdp::partial(results_Lemon, pred.var = imp_df_Lemon[3,
    1], prob = TRUE, train = rf_Lemon_juv.train, progress = "text",
    which.class = 2)
```

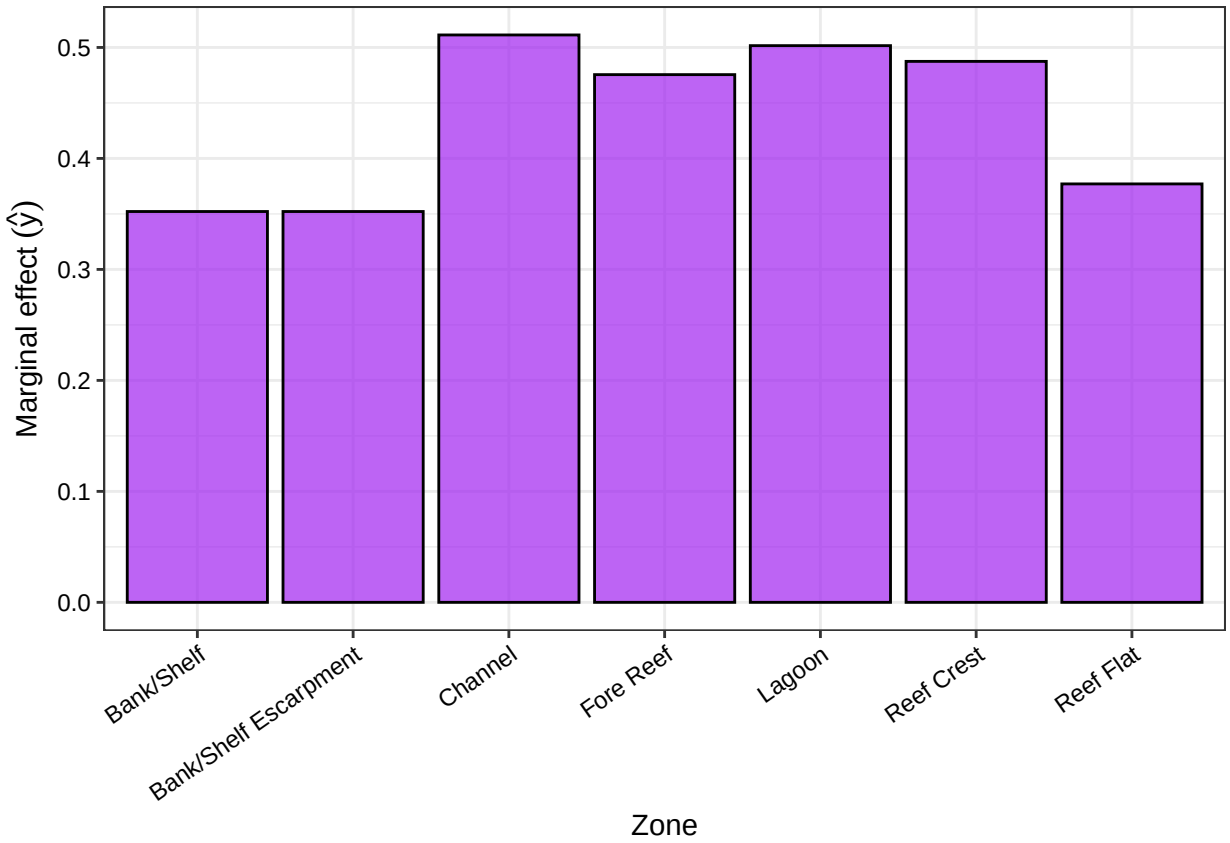
```
ggplot(top_1_pdp_df_Lemon, aes(x = Land_Dist, y = yhat)) +
  geom_smooth(col = "purple", method = "gam") +
  theme_bw() +
  xlab("Distance to land (m)") +
  ylab(bquote("Marginal effect" ~ (hat(y))))
```




```
ggplot(top_2_pdp_df_Lemon, aes(x = Depth, y = yhat)) +
  geom_smooth(col = "purple", method = "gam") +
  theme_bw() +
  xlab("Depth (m)") +
  ylab(bquote("Marginal effect" ~ (hat(y))))
```




```r
# Since the last most important predictor is discrete, we must first organize the appropiate labels
attributes <- levels(shapes@data$Zone)
ZoneTicks <- as.data.frame(attributes)
ZoneTicks$positions <- as.factor(seq(1:length(levels(shapes@data$Zone))))

positions <- as.factor(levels(rf_Lemon_juv.train$Zone))
Observed_attributes <- as.data.frame(positions)
Observed_attributes$Keep <- "Yes"

ZoneTicks_fin <- left_join(ZoneTicks, Observed_attributes) %>%
  filter(Keep == "Yes")

ggplot(top_3_pdp_df_Lemon, aes(x = Zone, y = yhat)) +
  stat_summary(fun = mean, geom = "bar", fill = "purple", col = "black", alpha = 0.7) +
  stat_summary(fun.data = mean_cl_normal, geom = "errorbar", width = 0.1) +
  theme_bw() +
  xlab("Zone") +
  ylab(bquote("Marginal effect" ~ (hat(y)))) +
  theme(
    axis.text.x = element_text(color = "black", angle = 35, vjust = 1, hjust = 1),
    axis.text.y = element_text(color = "black")) +
  scale_x_discrete(labels = ZoneTicks_fin$attributes)
```




Calculated interaction strengths across all predictors and plot the marginal effects of the top two-way interactions via partial dependency plots.

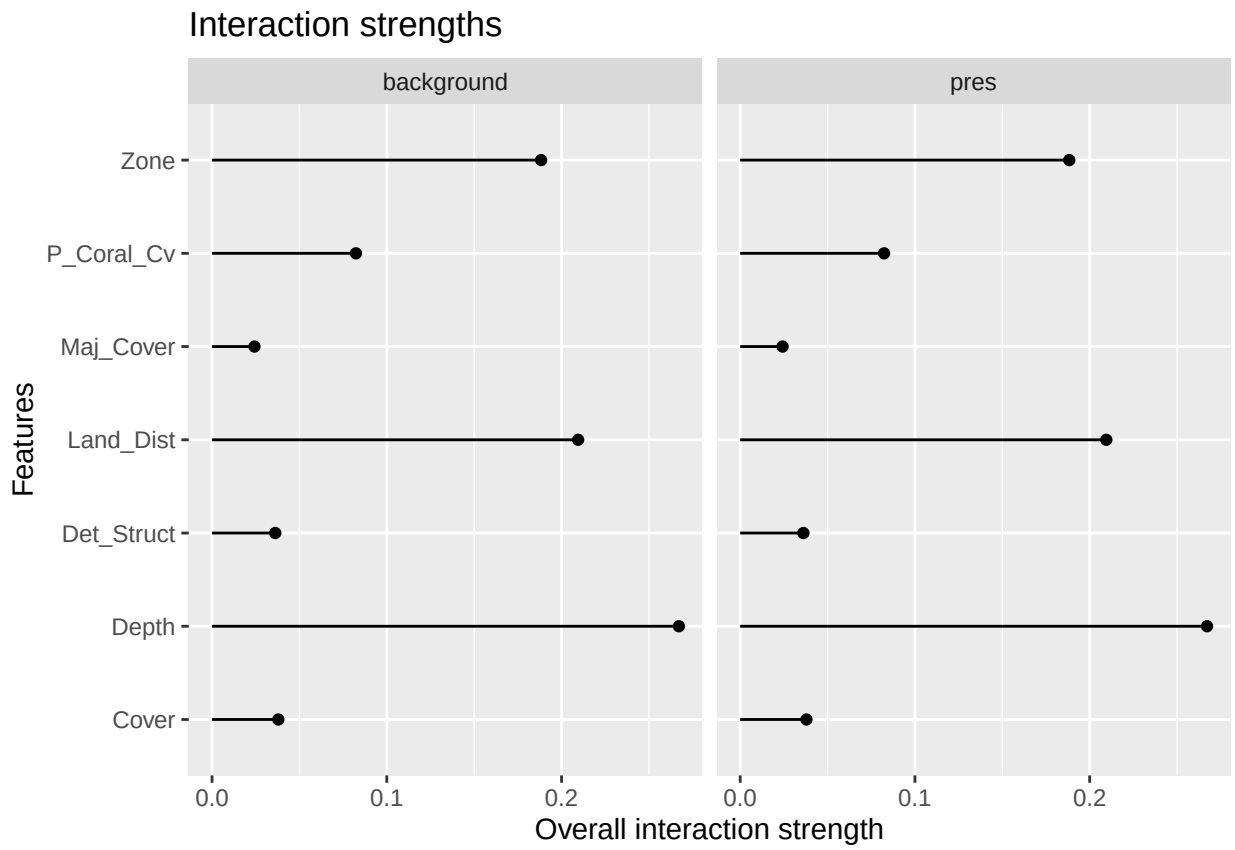
```
# Sub-sample and re-run model to reduce computational times when
# determining the top two-way interaction variables.
set.seed(1234)
rf_Lemon_juv.train_sub <- rf_Lemon_juv.train %>% mutate(rowID = seq(1:nrow(rf_Lemon_juv.train)))
rf_Lemon_juv.train_sub_f <- rf_Lemon_juv.train_sub[sample(1:nrow(rf_Lemon_juv.train_sub),
    nrow(rf_Lemon_juv.train_sub) * 0.25, replace = FALSE), ]

X_Lemon_sub <- rf_Lemon_juv.train_sub_f %>% dplyr::select(-c(presf,
    rowID))
# Create 'Predictor' object to interpret findings via the iml
# package.
predictor_Lemon_sub <- Predictor$new(model_rf.Bin_Lemon, data = X_Lemon_sub,
    y = rf_Lemon_juv.train_sub_f$presf)
```

Calculate interaction strengths for each predictor.

```
interact_Lemon_sub <- Interaction$new(predictor_Lemon_sub) %>% plot() +
    ggtitle("Interaction strengths")
```

Examine plot and determine which predictors produced the highest interaction strength.

```
interact_Lemon_sub
```




```
(interact_results_Lemon <- interact_Lemon_sub$data %>% group_by(.feature) %>%
    summarise(H = mean(.interaction)) %>% rename(feature = .feature) %>%
    as.data.frame() %>% arrange(desc(H)))
```

```
##      feature          H
## 1      Depth 0.26715009
## 2  Land_Dist 0.20950643
## 3       Zone 0.18832189
## 4 P_Coral_Cv 0.08235612
## 5      Cover 0.03791803
## 6 Det_Struct 0.03617434
## 7  Maj_Cover 0.02426464
```

Run top predictor again to determine which other predictor it should be interacting with.

```
(interact_top_1_Lemon <- iml::Interaction$new(predictor_Lemon_sub,
    feature = interact_results_Lemon$feature[1]))  # Running top predictor again.
```

```
# Determine which two-way interactions to examine.
(interact_top_1_Lemon_df <- interact_top_1_Lemon$results %>% group_by(.feature) %>%
    summarise(H = mean(.interaction)) %>% rename(feature = .feature) %>%
    as.data.frame() %>% arrange(desc(H)) %>% separate(col = feature,
    into = c("top1", "top2"), sep = "\\:"))
```

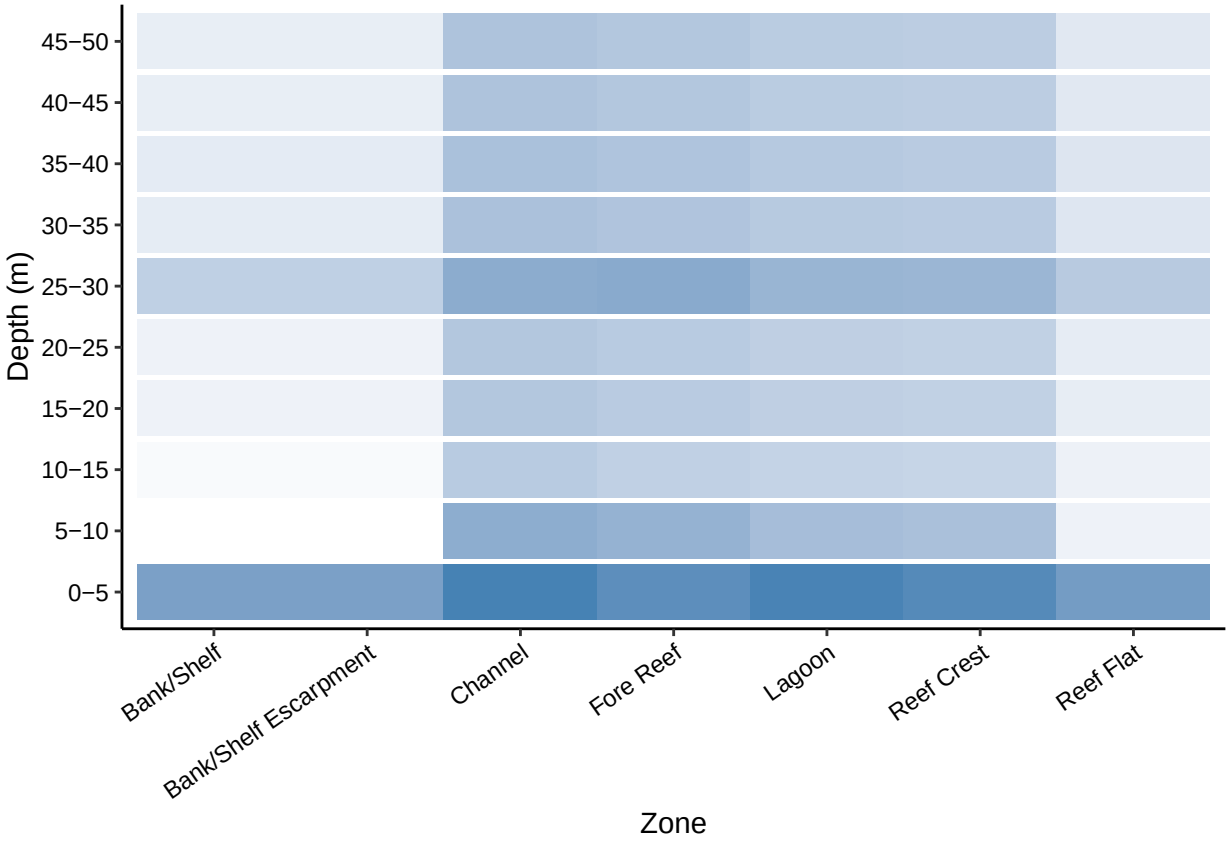
```
##         top1  top2         H
## 1       Zone Depth 0.2833353
## 2 P_Coral_Cv Depth 0.2342249
## 3  Land_Dist Depth 0.2055773
## 4      Cover Depth 0.1571744
## 5 Det_Struct Depth 0.0934953
## 6  Maj_Cover Depth 0.0382035
```

Generate a partial dependency plot for the two-way interaction of zone (Zone) and depth (Depth).

```
# pdp of zone (Zone) and depth (Depth)
top_int_1_pdp_df_Lemon <- results_Lemon %>% pdp::partial(pred.var = c(interact_top_1_Lemon_df$top1[1],
    interact_top_1_Lemon_df$top2[1]), prob = TRUE, which.class = 2,
    progress = "text", train = rf_Lemon_juv.train)
```

```
# Bin depth.
top_int_1_pdp_df_Lemon$Variable <- paste(top_int_1_pdp_df_Lemon$Zone,
                                         top_int_1_pdp_df_Lemon$Depth)
top_int_1_pdp_df_Lemon$Inter_Variable <- "Zone_Depth"
top_int_1_pdp_df_Lemon_bins_Depth <-
  cut(top_int_1_pdp_df_Lemon$Depth,
    breaks = c(1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 75, 100, 150, 200, 250),
    labels = c(
      "0-5", "5-10", "10-15", "15-20", "20-25", "25-30",
      "30-35", "35-40", "40-45", "45-50", "50-75", "75-100", "100-150", "150-200", "200-250"
    )
  )
top_int_1_pdp_df_Lemon$bins_Depth <- top_int_1_pdp_df_Lemon_bins_Depth

# Plot.
{
  top_int_1_pdp_df_Lemon_mean <-
    top_int_1_pdp_df_Lemon %>%
    data.frame() %>%
    filter(Depth <= 50) %>%
    group_by(bins_Depth, Zone) %>%
    summarise(ci = list(mean_cl_normal(yhat) %>%
      rename(mean = y, lwr = ymin, upr = ymax))) %>%
    unnest() %>%
    mutate(variance = (upr - lwr))

  ggplot(top_int_1_pdp_df_Lemon_mean, aes(Zone, bins_Depth)) +
    geom_tile(aes(fill = mean), width = 1, height = .9) +
    scale_fill_gradient(low = "white", high = "steelblue", breaks = seq(0, 1, by = 1)) +
    theme_classic() +
    xlab("Zone") +
    ylab("Depth (m)") +
    theme(
      axis.text.y = element_text(colour = "black"),
      axis.text.x = element_text(angle = 35, hjust = 1, colour = "black"),
      strip.text = element_text(face = "bold", size = 8, lineheight = 5.0),
      legend.position = "right"
    ) +
```

```
  scale_x_discrete(labels = ZoneTicks_fin$attributes)+
    labs(fill = "Mean Marginal effect" ~ (hat(y))) +
    guides(fill = FALSE)
}
```




Use the trained model to predict and extrpolate across the study area.

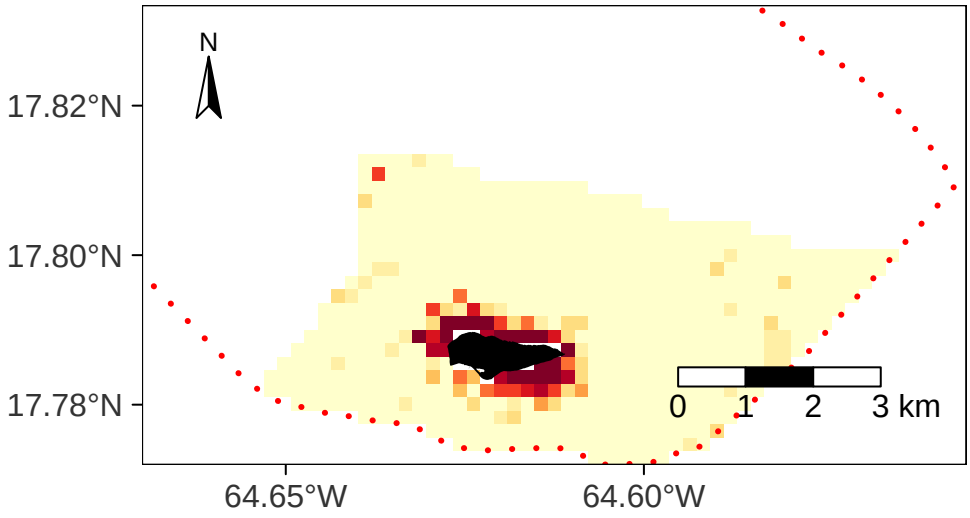
```
# Convert habitat raster brick into data frame.
new_data = as.data.frame(as.matrix(hab_r))
# Assign column names.
names(new_data) <-  names(hab_r)
# Convert to correct classes.
new_data$Cover = as.character(new_data$Cover)
new_data$Cover = as.factor(new_data$Cover)
new_data$Det_Struct = as.character(new_data$Det_Struct)
new_data$Det_Struct = as.factor(new_data$Det_Struct)
new_data$Maj_Cover = as.character(new_data$Maj_Cover)
new_data$Maj_Cover = as.factor(new_data$Maj_Cover)
new_data$P_Coral_Cv = as.character(new_data$P_Coral_Cv)
new_data$P_Coral_Cv = as.factor(new_data$P_Coral_Cv)
new_data$Zone = as.character(new_data$Zone)
new_data$Zone = as.factor(new_data$Zone)

# Predict function will not work with NAs,
# have to subset out, re-run, and merge by cell IDs.
new_Data_NAs= new_data # New data frame.
```

```
new_Data_NAs$ID = seq(1,length(new_Data_NAs$Cover)) # Generate ID column.
new_data2 = na.omit(new_data) # Omit NAs.
df_Lemon_PresOnly = rf_Lemon_juv %>% filter(presf == "pres") # Select presences only.
# Constrain data to the maximum depth observed.
max(new_data2$Depth)
```

```
## [1] 1335.767
```

```
new_data2 = new_data2 %>% filter(Depth <= max(df_Lemon_PresOnly$Depth))

# Use the trained model to predict across new data frame.
pred_total_Lemon = predict(model_rf.Bin_Lemon, newdata = new_data2,na.rm=TRUE,match.ID = TRUE)

# Merge predictions into one data frame.
predicteds_Lemon = bind_cols(pred_total_Lemon$data, new_data2)
predicteds_Lemon_full = left_join(new_Data_NAs,predicteds_Lemon)
predicteds_Lemon_full = arrange(predicteds_Lemon_full, ID)
# Remove duplicates.
predicteds_Lemon_full = predicteds_Lemon_full %>%
  distinct(ID, .keep_all = T)

# Place values inside an empty raster.
pred.raster_Lemon = r
pred.raster_Lemon[] = predicteds_Lemon_full$prob.pres
# Set projection.
crs(pred.raster_Lemon)= "+proj=utm +zone=20 +datum=NAD83 +units=m +no_defs +ellps=GRS80 +towgs84=0,0,0

# Plot predictions across the MPA with the tmap package.
tm_basemap("Esri.WorldImagery") +
  tm_shape(pred.raster_Lemon) +
  tm_raster("layer", palette="YlOrRd",legend.show = FALSE,
              title="Relative selection", style = "fixed",
              breaks = c(0,.10,.20,.30,.40,.50,.60,.70,.80,.90,1),
              labels = c("0.0 - 0.1","0.1 - 0.2", "0.2 - 0.3","0.3 - 0.4","0.4 - 0.5",
                         "0.5 - 0.6", "0.6 - 0.7","0.7 - 0.8", "0.8 - 0.9", "0.9 - 1.0"))+
  tm_shape(Land_shape) +
  tm_fill("black")+
  tm_shape(MPA_shape)  +
  tm_borders("red", lty = 3, lwd=3)+
  tm_layout(outer.margins = c(.1,.1,.1,.1),legend.bg.color = "white", legend.position = c("right", "top
              legend.width = 0.35, legend.height = 1) +
  tm_graticules(ticks = TRUE, lines = FALSE, labels.size = 1, n.x = 3, n.y = 3,col = "black")+
  tm_scale_bar(breaks = c(0,1,2,3), size = 1, position = c("right", "bottom"))+
  tm_compass(type = "arrow", position = c("left", "top"))
```

```
## Warning: The argument size of tm_scale_bar is deprecated. It has been renamed to
## text.size
```




We will use the dsmextra package to assess the model's extrapolation reliability. See Mesgaran et al. (2014)[6] and Bouchet et al. (2020)[7]for much greater detail on calculations, metrics, and use. This code below is heavily borrowed from the dsmextra vignette[8]. Please see vignette and additional suggested readings within to understand the three types of extrapolation metrics produced below.

Briefly, these metrics are based on values of the ExDet (EXtrapolation DETection) and include univariate extrapolation (<0, predictions outside the range of covariates), combinational extrapolation (>1, predictions made within range of covariates but for novel combinations), and geographical/temporal interpolation/extrapolation (0-1, predictions made within range of covariates and in analogous conditional space).

```
# remotes::install_github("densitymodelling/dsmextra", dependencies = TRUE) # If install is needed.
library(dsmextra)

# Derive predictive grid with coordinates specified.
xy_hab_r = as.data.frame(coordinates(hab_r))
predgrid = new_data
predgrid$x = xy_hab_r$x
predgrid$y = xy_hab_r$y

shark.extrapolation <- compute_extrapolation(samples = rf_Lemon_juv,
                                     covariate.names = c("Depth", "Land_Dist"), # Only numeric classes.
```

[6]Mesgaran, M. B., Cousens, R. D., & Webber, B. L. (2014). Here be dragons: a tool for quantifying novelty due to covariate range and correlation change when projecting species distribution models. Diversity and Distributions, 20(10), 1147-1159.

[7]Bouchet, P. J., Miller, D. L., Roberts, J. J., Mannocci, L., Harris, C. M., & Thomas, L. (2020). dsmextra: Extrapolation assessment tools for density surface models. Methods in Ecology and Evolution, 11(11), 1464-1469.

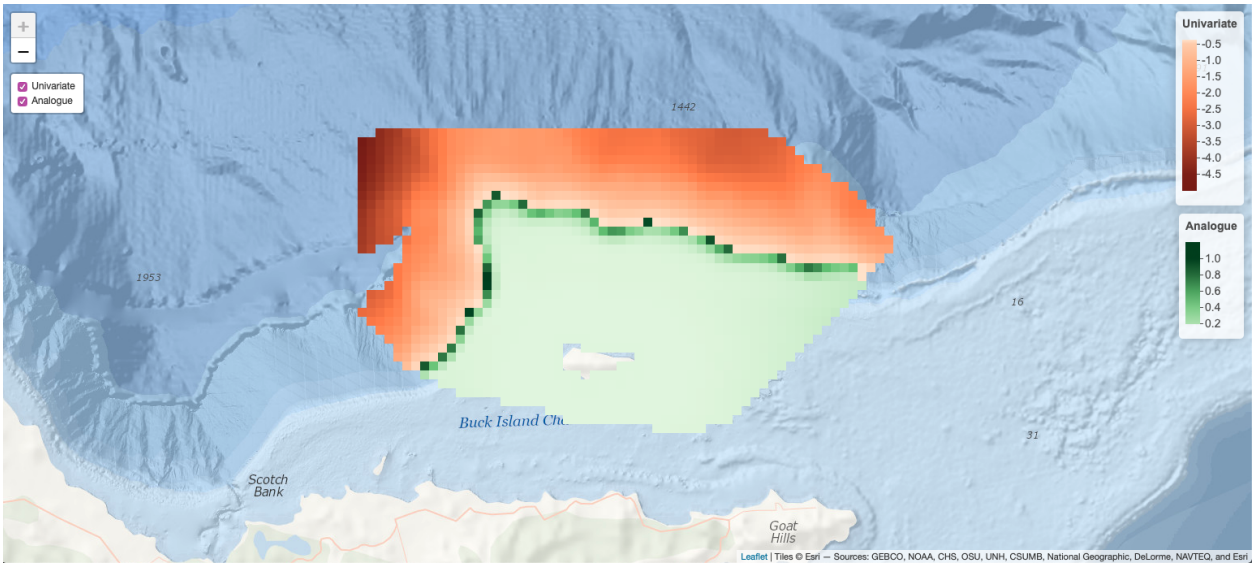
[8]https://densitymodelling.github.io/dsmextra/index.html, developed by Phil Bouchet, David Miller, Laura Mannocci, Jason Roberts, Catriona Harris, Len Thomas.

```
                                  prediction.grid = predgrid,
                                  coordinate.system = CRS("+proj=utm +zone=20
                                                          +datum=NAD83 +units=m
                                                          +no_defs +ellps=GRS80
                                                          +towgs84=0,0,0 "))
# Summary of extrapolation.
summary(shark.extrapolation)
```

```
##
##
## Table: Extrapolation
##
## Type          Count         Percentage
## ------------  ------------  ------------
## Univariate    770           48.7
## Analogue      811           51.3
## -----------   -----------   -----------
## Total         1581          100
##
##
## Table: Most influential covariates (MIC)
##
## Type          Covariate     Count         Percentage
## ------------  ------------  ------------  ------------
## Univariate    Depth         768           49
## Univariate    Land_Dist     2             0.13
## -----------   -----------   -----------   -----------
## Total                       770           49
```

## Map extrapolation detection/sensitivity across study area.

```
map_extrapolation(map.type = "extrapolation",
                  extrapolation.object = shark.extrapolation)
```


Univariate
Analogue
Univariate
-0.5
-1.0
-1.5
-2.0
-2.5
-3.0
-3.5
-4.0
-4.5
Analogue
1.0
0.8
0.6
0.4
0.2
1442
1953
16
31
Buck Island Ch
Scotch
Bank
Goat
Hills
Leaflet | Tiles © Esri — Sources: GEBCO, NOAA, CHS, OSU, UNH, CSUMB, National Geographic, DeLorme, NAVTEQ, and Esri