Classifying polytomous behavior from environmental features: Model selection, assessment, and interpretation using XGBoost

Additional supporting information for *Classifying animal movement from environmental features: a review of boosted classification trees and XGBoost with an example for bald eagles and wind turbines*

Silas Bergen, Manuela M. Huso, Adam E. Duerr, Missy A. Braham, Sara Schmuecker, Tricia A. Miller, Todd E. Katzner

# Introduction

The following provides detailed examples of fitting, selecting, assessing, and interpreting the XGBoost model when using environmental features to predict a polytomous animal behavioral response. This document supports *A review of supervised learning methods for classifying animal behavioral states from environmental features* by Bergen et al.

# Data

For this tutorial we will use in a subset of 10,000 flight points measured by GPS attached to bald eagles in Iowa, USA. The following code reads in the data file, assuming the data file is in the same directory as this R script.

setwd(dirname(rstudioapi::getActiveDocumentContext()$path))  
eagle\_subset <- read.csv('eagle\_subset.csv')  
#Make sure month is ordered chronologically; comes in useful for plotting later:  
eagle\_subset$month\_factor <- factor(eagle\_subset$month\_factor, levels = month.abb)  
head(eagle\_subset)

## OBJECTID Season\_longshort DEM\_IA TPI\_IA Slope\_IA d2water\_IA  
## 1 2725509 Dispersal/migration 229.74 0.010000000 0.05 0.00  
## 2 1403793 Fledgling period 280.34 -0.220000000 0.55 6.91  
## 3 556960 Dispersal/migration 330.52 4.430000000 5.91 1250.93  
## 4 1120710 Local movements 186.57 0.009989499 0.29 291.36  
## 5 1135477 Local movements 208.59 1.020000000 4.40 54.43  
## 6 1514768 Dispersal/migration 228.23 0.120000000 0.46 25.16  
## d2edge\_IA d2Landfills\_IA d2Feedlots\_IA d2streets d2nestsV2\_IA northness  
## 1 237.40 11203.84 3835.18 226.68 753.29 0.9855848  
## 2 7.30 13331.30 1966.93 394.48 583.21 -0.9586981  
## 3 0.00 15543.21 3456.01 509.18 2139.79 -0.9999987  
## 4 22.13 17156.93 5284.26 504.37 1936.59 -0.2786508  
## 5 4.85 5455.17 3655.67 516.63 10526.36 0.2384761  
## 6 876.81 17963.27 5795.95 1148.37 6436.98 0.1699671  
## month\_factor risk\_class risk\_name validation\_set  
## 1 Jun 0 Low Train  
## 2 Aug 2 High Test  
## 3 Aug 0 Low Train  
## 4 Mar 2 High Train  
## 5 Jan 2 High Train  
## 6 Sep 2 High Train

The columns Season\_longshort, DEM\_IA,…,d2nestsV2\_IA, northness, and month\_factor are the features we want to use to predict risk\_class. Accordingly we have 12 features, 2 of which (month\_factor and Season\_longshort) are factor variables and the others (DEM\_IA through northness) are numeric. The response variable to predicted is risk\_class which assumes values 0 (low risk), 1 (moderate risk), or 2 (high risk):

xtabs(~risk\_class + risk\_name, data = eagle\_subset)

## risk\_name  
## risk\_class High Low Moderate  
## 0 0 3197 0  
## 1 0 0 796  
## 2 6007 0 0

Note that risk\_class is numeric; this is important as the xgboost package requires a numeric response even when fitting classification methods:

class(eagle\_subset$risk\_class)

## [1] "integer"

The validation\_set column indicates which GPS points will be used to train and which will be used to test the SL methods. Roughly 2/3 of the data are used to train, the rest to test:

xtabs(~validation\_set, data = eagle\_subset)

## validation\_set  
## Test Train   
## 3370 6630

# Loading libraries

We will now load the libraries we will need for the following SL applications and assessments.

library(dplyr)  
library(xgboost)  
library(tidyr)  
library(ggplot2)

# The XGBoost method

## Data structures

To set up the data for XGBoost we select the predictor features, split the data into training and test sets, and one-hot encode the factor variables:

df <- eagle\_subset %>%   
 dplyr::select(risk\_class, DEM\_IA, TPI\_IA,  
 Slope\_IA, d2water\_IA,d2edge\_IA, d2Landfills\_IA,  
 d2Feedlots\_IA, d2streets, d2nestsV2\_IA, northness,month\_factor, Season\_longshort,validation\_set)   
train <- df %>% filter(validation\_set=='Train') %>% dplyr::select(-validation\_set)   
test <- df %>% filter(validation\_set=='Test') %>% dplyr::select(-validation\_set)   
trainX <- model.matrix(~.-1, data = train %>% dplyr::select(-risk\_class))  
testX <- model.matrix(~.-1, data = test%>% dplyr::select(-risk\_class))  
head(trainX, 3)

## DEM\_IA TPI\_IA Slope\_IA d2water\_IA d2edge\_IA d2Landfills\_IA d2Feedlots\_IA  
## 1 229.74 0.010000000 0.05 0.00 237.40 11203.84 3835.18  
## 2 330.52 4.430000000 5.91 1250.93 0.00 15543.21 3456.01  
## 3 186.57 0.009989499 0.29 291.36 22.13 17156.93 5284.26  
## d2streets d2nestsV2\_IA northness month\_factorJan month\_factorFeb  
## 1 226.68 753.29 0.9855848 0 0  
## 2 509.18 2139.79 -0.9999987 0 0  
## 3 504.37 1936.59 -0.2786508 0 0  
## month\_factorMar month\_factorApr month\_factorMay month\_factorJun  
## 1 0 0 0 1  
## 2 0 0 0 0  
## 3 1 0 0 0  
## month\_factorJul month\_factorAug month\_factorSep month\_factorOct  
## 1 0 0 0 0  
## 2 0 1 0 0  
## 3 0 0 0 0  
## month\_factorNov month\_factorDec Season\_longshortFledgling period  
## 1 0 0 0  
## 2 0 0 0  
## 3 0 0 0  
## Season\_longshortLocal movements  
## 1 0  
## 2 0  
## 3 1

We then create the data structures we will need for the xgboost package functions. Note the creation of the response by way of the label = argument:

dtrain <- xgb.DMatrix(trainX, label = train$risk\_class)  
dtest <- xgb.DMatrix(testX, label = test$risk\_class)

## Model selection

We are now ready to set up the tuning parameter grid.

##########################################################################################  
#Setting up the tuning parameter grid  
#M: the maximum allowed depth of each tree  
#csp: proportion of columns sampled for each tree in the boosting sequence  
#rsp: proportion of training rows sampled for each tree in the boosting sequence  
#eta: shrinkage parameter  
#lambda: penalty parameter on leaf weights  
#gamma: penalty parameter on number of leaves  
#########################################################################################  
  
grid1 <- expand.grid(M = seq(16,20,by=2),  
 csp = seq(.6,1,by=.2),  
 rsp=seq(.6,1,by=.2),  
 eta = c(0.1,0.2,0.3), lambda = c(0,1,2), gamma = c(0,1,2))

Next we set up the list we will use to collect the training results:

grid1\_results <- list()

By default, xgb.train() (the primary XGBoost training function) will return the value of the last eval\_metric provided (which, as we will see below, is the multi-class log-loss). This function parses the training message to also return the value of the misclassification error:

##Function to get the misclassification error from the XGBoost messages  
  
getmerror <- function(msg) {  
 splits <- strsplit(msg,":|\\\ttest", fixed=FALSE)  
 want <- as.numeric(do.call(rbind,splits)[,3])  
 return(want)  
}

Now we’re ready to train! This code will train the first 5 parameter combinations, printing progress notifications along the way, and evaluating training time:

#Train the first 5 parameter combos:  
  
for(i in 1:5) {  
 paramvec <- grid1[i,]  
 p <- with(paramvec, list(max\_depth = M,   
 colsample\_bytree = csp,  
 subsample = rsp,  
 eta = eta, gamma = gamma, lambda = lambda,  
 num\_class = 3))  
 cat("################################\n")  
 print(paste('Training ',i,'of',nrow(grid1)))  
 print(Sys.time())  
 print(unlist(p))  
 set.seed(1122021)  
 time.start <- Sys.time()  
 boostit <- xgb.train(params = p, data=dtrain, nrounds = 20000,  
 objective = "multi:softprob",   
 eval\_metric="merror",   
 eval\_metric="mlogloss",   
 watchlist = list(test = dtest),   
 early\_stopping\_rounds = 10,  
 print\_every\_n = 50)  
 time.end <- Sys.time()  
 train.time <- as.numeric(difftime(time.end, time.start, units='mins'))  
 print(paste('Train time = ', train.time))  
 grid1\_results[[i]] <- data.frame(boostit$params, bestiter = boostit$best\_iteration,  
 bestscore = boostit$best\_score, bestmsg = boostit$best\_msg,  
 runtime = train.time,  
 merror = getmerror(boostit$best\_msg))  
 #Uncomment the line below to save the grid search after each iteration  
 #save(grid1\_results, file = 'grid1\_results.Rdata')  
}

## ################################  
## [1] "Training 1 of 729"  
## [1] "2022-09-29 15:50:51 CDT"  
## max\_depth colsample\_bytree subsample eta   
## 16.0 0.6 0.6 0.1   
## gamma lambda num\_class   
## 0.0 0.0 3.0   
## [1] test-merror:0.471217 test-mlogloss:1.060057   
## Multiple eval metrics are present. Will use test\_mlogloss for early stopping.  
## Will train until test\_mlogloss hasn't improved in 10 rounds.  
##   
## Stopping. Best iteration:  
## [31] test-merror:0.358160 test-mlogloss:0.817821  
##   
## [1] "Train time = 0.164206647872925"  
## ################################  
## [1] "Training 2 of 729"  
## [1] "2022-09-29 15:51:01 CDT"  
## max\_depth colsample\_bytree subsample eta   
## 18.0 0.6 0.6 0.1   
## gamma lambda num\_class   
## 0.0 0.0 3.0   
## [1] test-merror:0.471810 test-mlogloss:1.059798   
## Multiple eval metrics are present. Will use test\_mlogloss for early stopping.  
## Will train until test\_mlogloss hasn't improved in 10 rounds.  
##   
## Stopping. Best iteration:  
## [29] test-merror:0.356677 test-mlogloss:0.819616  
##   
## [1] "Train time = 0.198060897986094"  
## ################################  
## [1] "Training 3 of 729"  
## [1] "2022-09-29 15:51:13 CDT"  
## max\_depth colsample\_bytree subsample eta   
## 20.0 0.6 0.6 0.1   
## gamma lambda num\_class   
## 0.0 0.0 3.0   
## [1] test-merror:0.479822 test-mlogloss:1.060032   
## Multiple eval metrics are present. Will use test\_mlogloss for early stopping.  
## Will train until test\_mlogloss hasn't improved in 10 rounds.  
##   
## Stopping. Best iteration:  
## [30] test-merror:0.355490 test-mlogloss:0.822492  
##   
## [1] "Train time = 0.21973751783371"  
## ################################  
## [1] "Training 4 of 729"  
## [1] "2022-09-29 15:51:26 CDT"  
## max\_depth colsample\_bytree subsample eta   
## 16.0 0.8 0.6 0.1   
## gamma lambda num\_class   
## 0.0 0.0 3.0   
## [1] test-merror:0.449852 test-mlogloss:1.058063   
## Multiple eval metrics are present. Will use test\_mlogloss for early stopping.  
## Will train until test\_mlogloss hasn't improved in 10 rounds.  
##   
## Stopping. Best iteration:  
## [26] test-merror:0.354896 test-mlogloss:0.816003  
##   
## [1] "Train time = 0.131557401021322"  
## ################################  
## [1] "Training 5 of 729"  
## [1] "2022-09-29 15:51:34 CDT"  
## max\_depth colsample\_bytree subsample eta   
## 18.0 0.8 0.6 0.1   
## gamma lambda num\_class   
## 0.0 0.0 3.0   
## [1] test-merror:0.462611 test-mlogloss:1.058377   
## Multiple eval metrics are present. Will use test\_mlogloss for early stopping.  
## Will train until test\_mlogloss hasn't improved in 10 rounds.  
##   
## Stopping. Best iteration:  
## [27] test-merror:0.360831 test-mlogloss:0.815907  
##   
## [1] "Train time = 0.152217268943787"

Some of the important arguments to the xgb.train() function:

* params- a vector of modeling tuning parameters;
* data - the training data of class xgb.DMatrix;
* nrounds - maximum number of boosting iterations considered;
* objective - specifies the classification problem, and whether to output class probabilities (multi:softprob) or predicted classes (multi:softmax);
* eval\_metric - the evaluation metric to use for finding the optimal number of boosting iterations. If multiple eval\_metrics are specified, the last one will be used;
* watchlist - a list containing an object of class xgb.DMatrix; the boosting will proceed until the eval\_metric has not improved for early\_stopping\_round iterations;
* early\_stopping\_round - if set to , training will stop if the performance doesn’t improve for rounds.

Next, we create a data frame of the training results and print some selected columns:

grid1\_res <- do.call(rbind, grid1\_results)  
grid1\_res %>%   
 arrange(merror) %>%   
 dplyr::select(max\_depth:num\_class, bestiter, bestscore, runtime, merror)

## max\_depth colsample\_bytree subsample eta gamma lambda num\_class bestiter  
## 1 16 0.8 0.6 0.1 0 0 3 26  
## 2 20 0.6 0.6 0.1 0 0 3 30  
## 3 18 0.6 0.6 0.1 0 0 3 29  
## 4 16 0.6 0.6 0.1 0 0 3 31  
## 5 18 0.8 0.6 0.1 0 0 3 27  
## bestscore runtime merror  
## 1 0.8160028 0.1315574 0.354896  
## 2 0.8224921 0.2197375 0.355490  
## 3 0.8196162 0.1980609 0.356677  
## 4 0.8178214 0.1642066 0.358160  
## 5 0.8159067 0.1522173 0.360831

## Model assessment

Having found our “best” tuning parameter combination, we will carry out 5-fold cross-validation by splitting the entire data set into 5 folds and assess the model’s predictive accuracy. Note that we fix nrounds (the number of boosting iterations) equal to the bestiter value from our parameter grid search.

######################################  
###5-fold CV and model assessment  
######################################  
  
  
fullmodel\_df <- eagle\_subset %>%   
 dplyr::select(DEM\_IA, TPI\_IA,  
 Slope\_IA, d2water\_IA,d2edge\_IA, d2Landfills\_IA,  
 d2Feedlots\_IA, d2streets, d2nestsV2\_IA, northness,month\_factor,Season\_longshort)   
X <- model.matrix(~.-1, data = fullmodel\_df)  
dX <- xgb.DMatrix(X, label = eagle\_subset$risk\_class)  
  
  
  
## Creating the 5 "folds" and putting the indices of each fold in a list:  
set.seed(282828)  
cv5\_id <- sample(1:5, nrow(eagle\_subset),replace=TRUE)  
fold\_list <- list()   
for(i in 1:5) fold\_list[[i]] <- which(cv5\_id==i)  
  
  
## XGBoost has its own K-fold cross-validation function:  
cv5.boost <- xgb.cv(data=dX, params = list(max\_depth = 16,   
 eta = 0.1,   
 colsample\_bytree = 0.8,  
 subsample = 0.6,  
 num\_class = 3,  
 gamma = 0,  
 lambda = 0),  
 nrounds = 26,  
 objective = "multi:softprob",   
 eval\_metric = 'mlogloss',  
 folds = fold\_list,   
 print\_every\_n = 20,  
 prediction = TRUE,showsd = FALSE)

## [1] train-mlogloss:1.020453 test-mlogloss:1.057741   
## [21] train-mlogloss:0.371598 test-mlogloss:0.809662   
## [26] train-mlogloss:0.312133 test-mlogloss:0.801490

We then put the predicted class probabilities in a matrix and compute the predicted classes as the maximum of the three probabilities:

xgboost.phat <- cv5.boost$pred  
xgboost.predclass <- apply(xgboost.phat, 1, function(x) which.max(x)-1)  
head(xgboost.phat)

## [,1] [,2] [,3]  
## [1,] 0.61336833 0.06400876 0.3226230  
## [2,] 0.26843476 0.09063949 0.6409258  
## [3,] 0.41262874 0.22390634 0.3634649  
## [4,] 0.18495451 0.08676135 0.7282842  
## [5,] 0.07613814 0.07630417 0.8475577  
## [6,] 0.10287201 0.08907115 0.8080569

head(xgboost.predclass)

## [1] 0 2 0 2 2 2

To assess the model, we compute pairwise area under the ROC curve (auROC) and average these to find the overall auROC.

##Function to compute pairwise and averaged pairwise area under ROC curve  
multiroc <- function(y, pmat) {  
 library(pROC)  
 df <- data.frame(y,pmat)   
 names(df) <- c('y','c0','c1','c2')  
 d01 <- df %>% mutate(p = c0/(c0+c1)) %>% filter(y!=2)  
 d02 <- df %>% mutate(p = c0/(c0+c2)) %>% filter(y!=1)  
 d12 <- df %>% mutate(p = c1/(c1+c2)) %>% filter(y!=0)  
 roc01 <- roc(factor(y)~p, data = d01)$auc %>% as.numeric  
 roc02 <- roc(factor(y)~p, data = d02)$auc %>% as.numeric  
 roc12 <- roc(factor(y)~p, data = d12)$auc %>% as.numeric  
 overall\_roc <- 2\*(roc01 + roc02 + roc12)/6  
 pairwise\_auc <- c('0v1'=roc01, '0v2'=roc02, '1v2'=roc12)  
 return(list('pairwise'=pairwise\_auc, 'overall'=overall\_roc))  
}  
  
##Evaluating pairwise auROCs:  
multiroc(eagle\_subset$risk\_class, xgboost.phat)

## $pairwise  
## 0v1 0v2 1v2   
## 0.6826190 0.7438505 0.5812624   
##   
## $overall  
## [1] 0.669244

Thus we have an auROC of 68.9% for evaluating low versus moderate risk; 74.7% for evaluating low versus high risk; and 61.1% for evaluating moderate versus high risk. Averaging these three gives us an overall auROC of 67.9%. These results are not great, and could be improved with a less hasty parameter search and larger portion of the original data.

We can also assess model accuracy by computing the confusion matrix. The confusionMatrix() function from the caret package computes additional prediction accuracy metrics as well such as class-specific sensitivity, specificity, and positive and negative predictive values.

caret::confusionMatrix(reference = factor(eagle\_subset$risk\_class), data = factor(xgboost.predclass))

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2  
## 0 1395 163 839  
## 1 5 4 11  
## 2 1797 629 5157  
##   
## Overall Statistics  
##   
## Accuracy : 0.6556   
## 95% CI : (0.6462, 0.6649)  
## No Information Rate : 0.6007   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.2636   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2  
## Sensitivity 0.4363 0.005025 0.8585  
## Specificity 0.8527 0.998262 0.3924  
## Pos Pred Value 0.5820 0.200000 0.6801  
## Neg Pred Value 0.7630 0.920641 0.6483  
## Prevalence 0.3197 0.079600 0.6007  
## Detection Rate 0.1395 0.000400 0.5157  
## Detection Prevalence 0.2397 0.002000 0.7583  
## Balanced Accuracy 0.6445 0.501643 0.6255

Overall we have 65.1% correct classification. The “by-class” statistics provide more detail as to how the model performs classifying one class versus the other () classes.

Finally, we fit the “best” XGBoost model to the entire data. We will use this best model to investigate variable importance and relationships with predicted class probabilities:

#############################################  
##FIT XGBOOST MODEL TO ENTIRE DATA:  
#############################################  
  
  
X <- model.matrix(~.-1, data = fullmodel\_df)  
dX <- xgb.DMatrix(X, label = eagle\_subset$risk\_class)  
  
best\_xgmod <- xgb.train(data=dX, list(max\_depth = 16,   
 eta = 0.1,   
 colsample\_bytree = 0.8,  
 subsample = 0.6,  
 num\_class = 3,  
 gamma = 0,  
 lambda = 0),  
 nrounds = 26,  
 objective = "multi:softprob",   
 eval\_metric = 'mlogloss')

## Interpreting the XGBoost model

### SHAP values

Having fit the “best” XGBoost model to the entire data set we can return the SHAP values, which measure the contribution of each observation’s feature values to its predicted class probabilities. Here we create SHAP values for the entire data set (consisting of only 10,000 rows); in practice if the entire data set consists of many more rows it may be necessary to create SHAP values for a subset to make computational time feasible. Setting approxcontrib=TRUE also aids computational time.

shapvals <- predict(best\_xgmod, newdata = X, predcontrib = TRUE, approxcontrib = TRUE)

The output shapvals is a list of length (number of classes), each list a matrix with one row per observation and one column per feature:

length(shapvals)

## [1] 3

dim(shapvals[[1]])

## [1] 10000 25

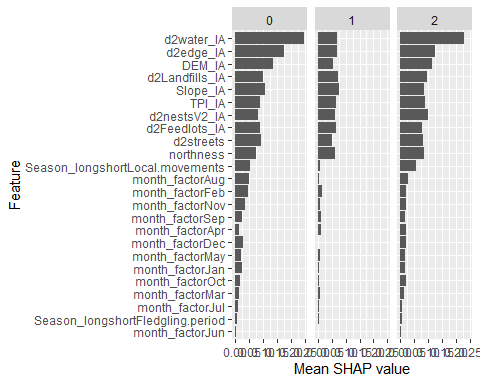
Combining these lists and finding the mean absolute value of the SHAP values for each feature and each predicted class allows us to identify predictor features that contribute the most to predicted class probabilities:

nshapvals <- nrow(shapvals[[1]])  
  
meanabs <- function(x) mean(abs(x))  
  
shap\_summary\_df <- do.call(rbind, shapvals) %>%   
 data.frame() %>%   
 dplyr::select(-BIAS) %>%   
 mutate(class = rep(0:2, each = nshapvals)) %>%   
 mutate(rowid = rep(1:nshapvals, 3)) %>%   
 gather(key = variable, value = value, DEM\_IA:Season\_longshortLocal.movements) %>%   
 group\_by(variable, class) %>%   
 summarize(meanshap = meanabs(value)) %>%   
 ungroup(variable) %>%   
 mutate(variable = reorder(variable, meanshap, FUN = mean))  
   
head(shap\_summary\_df)

## # A tibble: 6 × 3  
## variable class meanshap  
## <fct> <int> <dbl>  
## 1 d2edge\_IA 0 0.173   
## 2 d2edge\_IA 1 0.0694  
## 3 d2edge\_IA 2 0.123   
## 4 d2Feedlots\_IA 0 0.0880  
## 5 d2Feedlots\_IA 1 0.0660  
## 6 d2Feedlots\_IA 2 0.0765

Plotting these mean absolute SHAP values illustrates the importance of each predictor feature:

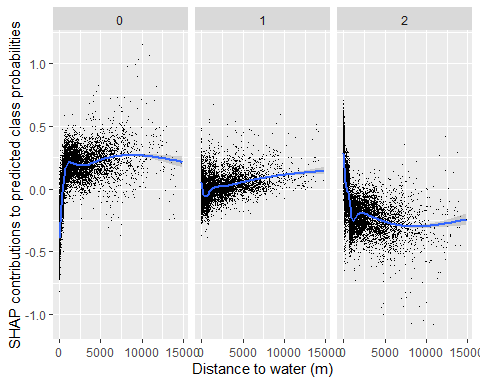
ggplot(data = shap\_summary\_df) +   
 geom\_bar(aes(x = meanshap, y= variable), stat = 'identity') +   
 facet\_wrap(~class) +   
 xlab('Mean SHAP value') +   
 ylab('Feature')



Here we can see that the d2water, d2edge, and DEM variables appear contribute the most to predicting class probabilities.

Next we can examine relationships between the values of certain features and their associated SHAP values. Here we plot the SHAP contributions of the d2water variable for each class:

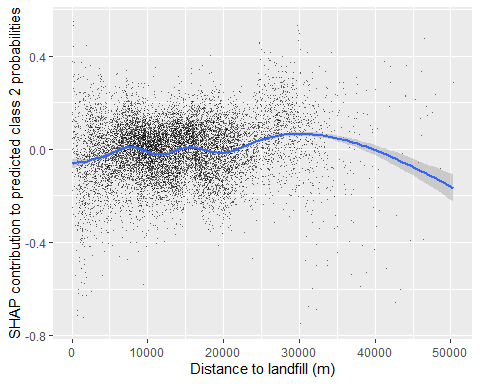
d2water\_shap <- data.frame(class = rep(0:2, each = nrow(df)),  
 watershap = c(shapvals[[1]][,'d2water\_IA'],  
 shapvals[[2]][,'d2water\_IA'],  
 shapvals[[3]][,'d2water\_IA']),  
 d2water = rep(df$d2water\_IA, 3)  
 )  
  
ggplot(data = d2water\_shap, aes(x = d2water, y = watershap)) +   
 geom\_point(shape = '.') +  
 geom\_smooth() +   
 facet\_wrap(~class) +   
 xlab('Distance to water (m)') +   
 ylab('SHAP contributions to predicted class probabilities')



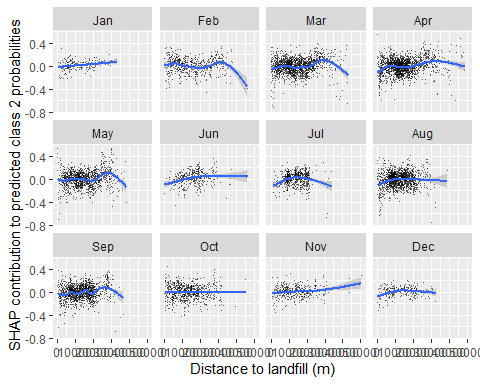
From this plot we can discern that large distances to water are associated with positive contributions to class 0 probabilities and negative contributions to class 2 probabilities. Vertical variability in the points illustrates that the contributions are interacting with other predictor features that are not visualized.

We can investigate interactions between features by faceting. In the next plot we demonstrate how the relationship of distance to landfill with contributions to class 2 probabilities varies by month:

### Add the class 2 D2Landfill SHAP values to original data frame   
eagle\_subset$landfill\_shap <- shapvals[[3]][,'d2Landfills\_IA']  
  
  
### Create class 2 SHAP plot for D2Landfill  
ggplot(data = eagle\_subset,aes(x = d2Landfills\_IA, y = landfill\_shap)) +   
 geom\_point( shape='.',alpha= 0.6) +   
 xlab('Distance to landfill (m)') +   
 ylab('SHAP contribution to predicted class 2 probabilities') +   
 geom\_smooth()



### Faceted by month  
ggplot(data = eagle\_subset,aes(x = d2Landfills\_IA, y = landfill\_shap)) +   
 geom\_point( shape='.',alpha= 0.6) +   
 geom\_smooth() +   
 xlab('Distance to landfill (m)') +   
 ylab('SHAP contribution to predicted class 2 probabilities') +   
 facet\_wrap(~month\_factor)



We can discern from here that large values of distance to water are perhaps more strongly associated with increased contributions to class 2 probabilities in the summer months, though these relationships are slight.

### ICE and partial dependence plots

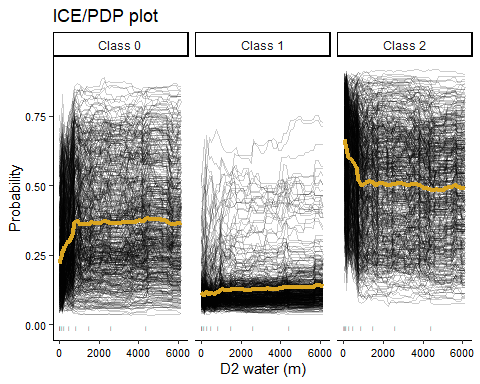
A single line on an individual conditional expectation (ICE) plot shows how predictions for each data point vary for changes in a predictor of interest, keeping all other predictors for that observation constant at their observed values. The partial dependence is simply the average of all ICE lines. Since plots of ICE involve one line per data point, to avoid overplotting it may be useful to subsample the data before creating ICEs and their subsequence partial dependencies. In what follows we compute ICEs for a sample of 500 points and visualize them along with their partial dependencies.

#To create the ICE's, form a sequence along each predictor and predict for the sequence.  
#All other variables are held constant.  
  
landvars <- eagle\_subset %>%   
 dplyr::select(DEM\_IA, TPI\_IA,  
 Slope\_IA, d2water\_IA,d2edge\_IA, d2Landfills\_IA,  
 d2Feedlots\_IA, d2streets, d2nestsV2\_IA, northness,month\_factor,Season\_longshort) %>%   
 data.frame()  
  
allvars <- names(landvars)  
  
  
#Using SHAP plot to order:  
varnames <- c('d2water\_IA','d2Landfills\_IA','d2nestsV2\_IA',  
 'DEM\_IA','d2Feedlots\_IA','d2edge\_IA','Slope\_IA',  
 'd2streets','TPI\_IA','northness','Season\_longshort','month\_factor')  
pdp\_df\_list\_multiclass <- list()  
  
  
  
##Creating the ICE for each variable for a Sample of 1000 points:  
  
for(i in 1:length(varnames)) {  
 feature <- varnames[i]  
 x <- landvars[,feature]  
 deciles <- NULL  
 if(!feature %in% c('month\_factor','Season\_longshort')) {  
 gridbounds <- quantile(x, c(.05, .95))  
 grid <- seq(gridbounds[1],gridbounds[2],l=50)  
 deciles <- quantile(x, seq(.1, .9, by = .1))  
 }  
 if(feature =='month\_factor') grid <- factor(month.abb)  
 if(feature =='Season\_longshort') grid <- factor(c('Dispersal/migration','Fledgling period','Local movements'))  
 set.seed(1234)  
 sampind <- sample(1:nrow(landvars),500,replace=FALSE)  
 Xpred <- landvars %>%   
 dplyr::slice(sampind)%>%  
 dplyr::slice(rep(1:n(),each=length(grid)))  
 Xpred[,feature] <- rep(grid,length(sampind))  
 if(feature=='month\_factor') Xpred[,feature] <- factor(Xpred[,feature],levels=month.abb)  
 Xpredmat <- model.matrix(~.-1, data = Xpred)  
 yhat <- predict(best\_xgmod, newdata = Xpredmat)  
 yhat\_mat <- matrix(yhat, nrow(Xpred),3,byrow = TRUE)  
 ice <- Xpred %>%   
 mutate(p0 = yhat\_mat[,1],  
 p1=yhat\_mat[,2],  
 p2 = yhat\_mat[,3]) %>%   
 dplyr::select(feature, p0:p2) %>%   
 mutate(rowid = rep(1:500,each=length(grid)))  
 names(ice)[1] <- 'ventile'  
 pdp <- ice %>%  
 group\_by(ventile) %>%   
 summarize(across(.cols=p0:p2,.fns=list(mean = mean,sd = sd)))  
 names(pdp)[1] <- names(ice)[1] <- feature  
 print(paste('done with ',feature))  
 pdp\_df\_list\_multiclass[[i]] <- list('feature' = feature, 'pdp' = pdp, 'deciles'=deciles,'ice'=ice)  
 #Uncomment below line to save data object as variables complete  
 #save(pdp\_df\_list\_multiclass, file = 'pdp\_df\_list\_multiclass.Rdata')  
}

## [1] "done with d2water\_IA"  
## [1] "done with d2Landfills\_IA"  
## [1] "done with d2nestsV2\_IA"  
## [1] "done with DEM\_IA"  
## [1] "done with d2Feedlots\_IA"  
## [1] "done with d2edge\_IA"  
## [1] "done with Slope\_IA"  
## [1] "done with d2streets"  
## [1] "done with TPI\_IA"  
## [1] "done with northness"  
## [1] "done with Season\_longshort"  
## [1] "done with month\_factor"

The resulting list contains the name of the feature for which ICEs are computed; the partial dependence of the class probabilities on that feature; deciles of the distribution of the feature; and the ICEs themselves. Using this we can create ICE and partial dependence plots:

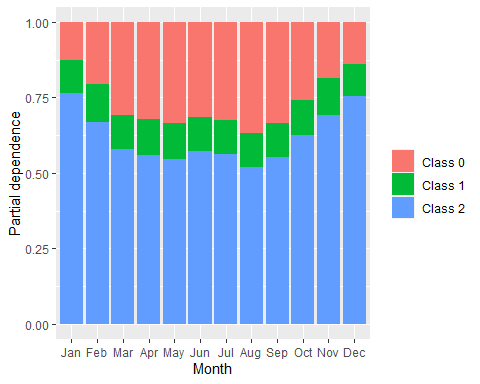
###############################  
##ICE/PDP plot for d2water  
###############################  
  
waterdf <- pdp\_df\_list\_multiclass[[1]]$ice %>%   
 gather(key = p, value = ice, p0:p2)  
  
waterdeciles <- data.frame(deciles = pdp\_df\_list\_multiclass[[1]]$deciles)  
  
(watericeplot <- ggplot(data = waterdf) +   
 geom\_line(aes(x = d2water\_IA, y = ice, group=rowid),alpha = .2, size = .1) +  
 stat\_summary(aes(x = d2water\_IA, y = ice),fun='mean',col='goldenrod',geom='line',size = 1.5) +   
 facet\_wrap(~p, labeller = labeller(p = c('p0' = 'Class 0', 'p1' = 'Class 1','p2' = 'Class 2'))) +   
 ylab('Probability') + xlab('D2 water (m)') +   
 geom\_point(aes(x = deciles, y = -0.01), shape='|', data = waterdeciles,col='black',size=1) +   
 theme\_classic() +   
 theme(axis.text = element\_text(color='black', size = 8)) +   
 xlim(c(0,6203))+  
 ggtitle('ICE/PDP plot'))



Similarly to the SHAP plot, we can see that increasing distances to water appear to be associated with increasing class 0 probabilities and decreasing class 2 probabilities. The deciles show us that the distribution of distance to water is quite right-skewed.

Stacked bar graphs work well to visualize the partial dependence of class probabilities on categorical features:

month\_pdp <- pdp\_df\_list\_multiclass[[12]]$pdp %>%   
 dplyr::select(month\_factor, contains('mean')) %>%   
 gather(key = class, value = prob, -month\_factor)  
  
ggplot(data = month\_pdp) +   
 geom\_bar(aes(x = month\_factor, fill = class, y = prob), stat='identity') +   
 scale\_fill\_discrete(labels= c('Class 0','Class 1','Class 2'), name = '') +   
 xlab('Month') + ylab('Partial dependence')



Here we can see that class 0 probabilities tend to be highest in summer, inversely related to class 2 probabilities. Class 1 probabilities appear relatively constant across seasons.