Predicting Dolphin Activity Using AdaBoost

Timo Pew

December 10, 2019

1 Introduction

Sarasota Florida's bottlenose dolphins are apex predators and are therefore an important indicator species of the health of the entire ecosystem. The Sarasota Dolphin Research Program study the resident bottlenose dolphin population of Sarasota Bay, the longest-running population study of its type. We have 2,030 observations from dolphin follows with information about the depth, habitat, group spread, group size, boats within 100 meters, whether or not humans were illegally close, the distance to the nearest dolphin, and activity of the focal dolphin. Because researchers are primarily interested in how human interactions affect dolphin behavior, the dolphin's activity is one of the most important variables to collect. There are four primary activities of interest: feeding, milling, socializing, and traveling. One of the major challenges with collecting this information is it is often difficult to be certain when the dolphin is feeding. To combat this, researchers include a fifth activity, 'probably feeding,' that they record when they think the dolphin is feeding but never see a fish in the dolphin's mouth. As the primary purpose of this project, we wish to implement the AdaBoost algorithm to predict dolphin behavior—and specifically feeding dolphins—accurately so that we can predict whether or not each of the observations recorded as 'probably feeding' are actually feeding. This information may help the researchers know how accurate their assessment of 'probably feeding' is and potentially allow them to feel comfortable assigning them as 'feeding' in the future.

A secondary purpose of this project is to explore how several changes to the AdaBoost algorithm affect its prediction abilities. Specifically, we will look at two changes to the decision trees used as weak learners: the cost matrix and number of splits. In addition, we will also consider changes to the number of trees grown. Under different combinations of each setting we will compare prediction accuracy using leave-one-out and 5-fold cross-validation.

2 Method

To build a classifier to predict dolphin behavior, we used the SAMME multiclass AdaBoost algorithm described in Hastie et al. (2009) with some modifications. We first explain the intuition behind the algorithm before we lay out its steps. Boosting is known as an ensemble classifier. That is, it combines multiple weak learners to produce more accurate classifiers. Weak classifiers produce classification rules that are at least better than random guessing. An example of a weak learner, and the kind that we will use, is a decision tree with few splits. In an iterative process, a weak learner is applied to the data, observations misclassified by the weak learner are assigned more weight, after which a new weak learner is applied. This process allows the algorithm to place more focus on the observations it misclassifies at each iteration, thus 'learning' a better classification rule. Traditionally, each weak learner is required to have a misclassification rate less than 0.5. The SAMME algorithm takes a more general approach better applied to multiclass problems by only requiring each weak learner to be better than random guessing ($1/K$ where K is the number of classes). See Hastie et al. (2009) for more details about the SAMME algorithm. This process continues until the predetermined number of weak learners are created. The final classification rule is a linear combination of the weak learners with the weights based on weak learners accuracy.

We let $\boldsymbol{x}_i$ denote the predictor variables of the i^{th} observation. This includes all variables except the dolphin's activity. We let a_i denote the activity of the dolphin at the i^{th} observation, and $T(\boldsymbol{x})$ denote our weak decision tree classifier as a function of the variables in $\boldsymbol{x}$. The steps of the algorithm are found in Algorithm 1.

Algorithm 1: SAMME Boosting

```
1. Initialize the weights of each observation to be $\frac{1}{n}$: $w_i^{(1)} = \frac{1}{n}$ for $i = 1, \ldots, n$
2. For b in $1, \ldots, B$
    (i) Fit a weak classifier $T^{(b)}(\boldsymbol{x})$ to the data with the weights $w_i^{(b)}$
    (ii) Compute the error of the classifier, $err^{(b)} = \frac{1}{s}\sum_{i=1}^{n} w_i^{(b)} \mathbb{I}(a_i \neq T^{(b)}(\boldsymbol{x}_i))$, where $s = \sum_{i=1}^{n} w_i^{(b)}$
    (iii) Compute the weight of the classifier, $\alpha^{(b)} = log\left(\frac{1-err^{(b)}}{err^{(b)}}\right) + log(K-1)$
    (iv) Compute new weights, $w_i^{(b+1)} = w_i^{(b)} exp\left(\alpha^{(b)} \mathbb{I}(a_i \neq T^{(b)}(\boldsymbol{x}_i))\right)$ for $i, \ldots, n$
    (v) Renormalize weights
3. The final classifier, $C(\boldsymbol{x}) = \arg\max_k \sum_{b=1}^{B} \alpha^{(b)} \mathbb{I}(k = T^{(b)}(\boldsymbol{x}))$
```

In its simplest and most common form, the decision tree $T^{(b)}(\boldsymbol{x})$ weighs the cost of any misclassification equally. However, in multiclass classification it is not uncommon to value the classification or prediction of the different classes unequally as is the case for us predicting dolphin activity. We take advantage of the flexibility that decision trees offer by specifying our own cost matrix used to build each tree. The best split at a given node (we will denote as τ) is determined by finding the predictor and its value that partitions τ into two subsets (τ_1 and τ_2) that maximizes the information gained. The information gained for a given split, $IG(\tau, \tau_1, \tau_2) = I(\tau) - \frac{n_1 I(\tau_1)}{n} - \frac{n_2 I(\tau_2)}{n}$ where n is the number of observations in τ, n_1 is the number of observations in τ_1, n_2 is the number of observations in τ_2 and $I(\cdot)$ is the impurity function. The cost matrix modifies $I(\cdot)$. For example, the Gini index (a common choice for $I(\cdot)$ and the impurity function we used) using an unweighted cost matrix is, $I_g(\tau) = \sum_{k=1}^{K}(p_k|\tau)(1 - p_k|\tau))$ where $p_k|\tau$ is the proportion of observations from class k in τ. We can generalize this to include a cost matrix as follows: $I_g(\tau) = \sum_{k=1}^{K} C(k|c_\tau)(p_k|\tau)(1 - p_k|\tau))$ where $C(k|c_\tau)$ is the associated cost of classifying an observation from class k into the class label associated with τ. It is clear that this simplifies to the same equation as before when $C(k|c_\tau) = 1$ for $k \neq \tau$. Thus the information gained from a potential split is reduced and less likely to be the optimal split when the cost associated with misclassifying observations in the resulting nodes increases.

Therefore we can change three features or parameters of the AdaBoost algorithm and its decision trees: the cost matrix, the number of splits in each tree, and the number of trees grown. In some situations it may be justifiable to fix the cost matrix based on subject-expert knowledge and then use the overall misclassification rate to determine the best classification rule. However, in this application we found that the cost matrix we initially chose provided unsatisfactory predictions. For our approach we chose to treat the cost matrix as something of a tuning parameter. The value (or matrix in the case of the cost matrix) of each parameter can then chosen based on which value produces the 'best' predictions. The question then becomes how the 'best' classifier should be defined.

Traditionally, the best classifier is the one that produces the smallest classification error. We wanted our criteria to determine the best classifier to place a higher priority on accurately predicting feeding dolphins, as that is the primary purpose of our project. We use TP_F, TP_M, TP_S, and TP_T to denote the true positive rates of feeding, milling, socializing, and traveling respectively (i.e. TP_F is the proportion of observations recorded as feeding that were predicted correctly). We denote the positive predictive value for each activity as PV_F, PV_M, PV_S, and PV_T (i.e. PV_F is the proportion of observations predicted as feeding that were actually feeding). In collaboration with a subject-matter expert, we chose the following metric based on weighted true positive and positive predictive rates to determine the best classifier:

$$WR = (4 \cdot TP_F + TP_M + 3 \cdot TP_S + TP_T + 3 \cdot PV_F + 2 \cdot PV_M + PV_S + 2 \cdot PV_T)/17. \tag{1}$$

The coefficients correspond to the importance or weight of each rate. For example, we determined that it was 4 times as valuable to correctly identify a feeding dolphin's activity as it was to correctly identify a traveling or milling dolphin's activity. We decided to place a higher priority on accurately predicting socializing dolphins correctly to prevent it from being underpredicted since it had the lowest prior probability. We note that we placed a high priority on PV_F to protect from preferring models that heavily overpredict dolphins as feeding. We divide by 17 (the sum of the coefficients) so that WR is bounded by 0 and 1, making it a weighted accuracy rate on a familiar scale. $WR = 1$ corresponds to a classifier that predicts perfectly while $WR = 0$ would indicate a classifier that is always wrong. We acknowledge that our choice of weighting in our metric WR is somewhat subjective and arbitrary. However, we like the flexibility to be able to say that a given classifier is the ‘best’ classifier based on what type of prediction accuracy is important to us, especially in a multiclass setting.

Typically the decision trees are constructed using prior probabilities equal to the observed proportions. We adjusted and held fixed the prior probabilities to reflect a subject-matter expert’s beliefs about the time dolphins spend feeding, milling, socializing, and traveling (feeding = 0.14, milling = 0.24, socializing = 0.10, and traveling = 0.52). Because we use WR to determine how well a classifier predicts, and we treat the cost matrix as a tuning parameter, our choice for the cost matrix was the result of trial and error and based on collaboration with a subject-matter expert. It should be unsurprising that the cost matrix is closely related with our choice for WR. The cost matrix we used is shown in Table 1. The interpretation of the costs are straightforward, but to give an example, the second element in the first row indicates that predicting a dolphin that was feeding as milling is 1.25 times more costly than predicting a feeding dolphin as socializing.

Table 1: Adjusted cost matrix for decision trees

	Predicted			
Truth	Feeding	Milling	Socializing	Traveling
Feeding	0.00	1.25	1.00	1.75
Milling	1.50	0.00	0.75	1.25
Socializing	1.00	1.25	0.00	1.50
Traveling	1.00	0.75	0.50	1.00

In addition to using the cost matrix in Table 1, we used a standard cost matrix with zeroes down the diagonal and ones on the off-diagonal. We would have considered other choices as well if not for time constraints. We used values of 10, 50, and 100 for B, the number of decision trees; and 1, 2, and 8 for the number of splits allowed in $T^{(b)}(\boldsymbol{x})$. As the value of B increases, the in-sample prediction accuracy increases, but potentially at the cost of out-of-sample prediction accuracy. Likewise, as the number of splits allowed increases, the in-sample prediction accuracy generally increases but eventually at the cost of out-of-sample accuracy. It is common to use a single split for each weak learner, but here we explore allowing 2 or 8 splits as well. By using a range of values, we hope to find the point where the algorithm has learned an adequate classification rule without overfitting.

We explored WR using leave-one-out and 5-fold cross validation with adjustments to the cost matrix, the number of splits allowed, and the number of weak classifiers made. This exploration accomplishes two purposes. First, we can use the values that produce the best predictions to build our final classifier that we will use to predict the ‘probably feeding’ observations. Second, we will learn how the different tuning parameters affect the prediction accuracy, even if it’s only in our specific application. In leave-one-out cross-validation, each point is held out, the algorithm is run with the specified parameters, and the left-out observation is predicted. In 5-fold cross validation, the data is randomly partitioned into five groups afterwhich one group is held out, the algorithm is run, and the held out group is predicted. Because 5-fold cross-validation involves randomly partitioning the data, we repeated the 5-fold cross-validation 100 times for each combination of parameters and took the average value of WR.

To predict the activity of the observations recorded as probably feeding, we used the best combination of cost matrix, number of trees, and number of splits as defined by WR to build the final classifier $C(\boldsymbol{x})$. We then applied this classifier to each of the ‘probably feeding’ observations to predict the activity based on its observed depth, habitat, etc. $(\boldsymbol{x})$. The estimated probability that a_i is in the k^{th} class is then equal to:

$$P(a_i = k) = \frac{\sum_{b=1}^{B} \alpha^{(b)} \mathbb{I}(k = T^{(b)}(\boldsymbol{x}))}{\sum_{k=1}^{K} \sum_{b=1}^{B} \alpha^{(b)} \mathbb{I}(k = T^{(b)}(\boldsymbol{x}))}. \tag{2}$$

3 Results

The results from our cross-validation study are shown in Table 2. The overall accuracy is also indicated in parenthesis. In general, our chosen cost matrix from Table 1 produced better predictions than the traditional symmetric cost matrix using leave-one-out and 5-fold cross-validation. The prediction abilities suffer when only one split is used in the weak classifier. This is a little surprising considering it is common to use a single split for decision trees in boosting algorithms. Using 2 or 8 splits produce similar results with 8 splits performing better in general. It appears that using 10 weak learners is not sufficient to build a strong boosting classifier, and the results for 50 or 100 weak learners were similar. Somewhat surprisingly, we did not appear to reach the point where increasing the number of trees or splits overfit the training data and predicted worse. Jumping from 50 to 100 trees did decrease WR using the adjusted cost matrix and 8 splits, otherwise we do not see much evidence of overfitting. We would have tried using more trees or splits to have a better understanding of where overfitting starts to happen if not for the computation burden of performing the cross-validation and time restraint. The results were comparable for both leave-one-out and 5-fold cross validation.

Table 2: Results from cross-validation study. Values for 5-fold cross-validation are averages over the 100 iterations. Unweighted accuracy are recorded in parenthesis

		Leave-One-Out		5-Fold	
Max Splits	B	CM 1	CM 2	CM 1	CM 2
	10	0.203 (0.51)	0.283 (0.63)	0.261 (0.53)	0.277 (0.62)
1	50	0.203 (0.51)	0.255 (0.64)	0.231 (0.61)	0.272 (0.64)
	100	0.203 (0.51)	0.233 (0.64)	0.226 (0.62)	0.270 (0.66)
	10	0.433 (0.59)	0.365 (0.65)	0.412 (0.57)	0.352 (0.63)
2	50	0.443 (0.60)	0.364 (0.65)	0.423 (0.57)	0.347 (0.65)
	100	0.458 (0.61)	0.341 (0.65)	0.427 (0.57)	0.348 (0.66)
	10	0.452 (0.63)	0.435 (0.65)	0.439 (0.59)	0.407 (0.63)
8	50	0.465 (0.64)	0.435 (0.65)	0.466 (0.62)	0.424 (0.65)
	100	0.452 (0.64)	0.450 (0.65)	0.469 (0.62)	0.422 (0.66)

Overall, we felt that using our adjusted cost matrix, allowing 8 splits, and using 50 weak learners was the best combination of parameters. We used these parameters on all of the data to build our final classifier to predict the dolphin's activity when they were recorded as probably feeding, as will be shown later.

We also found it encouraging that the unweighted accuracy was generally not substantially lower using our adjusted cost matrix. One concern we had with using our adjusted cost matrix and WR was that it would correctly prioritize predicting dolphin's feeding accurately but at the cost of much lower prediction accuracy overall. Our results suggest that our increased emphasis on predicting feeding observations only came at a small cost to the overall prediction accuracy. The confusion matrix from each cost matrix with 50 weak learners and 2 splits shown in Table 3 demonstrates this trade-off. The adjusted cost matrix correctly predicts 3 more feeding dolphins. However, it also inaccurately predicts more milling and traveling dolphins as feeding as a result. It also does a much better (although still not great) job predicting socializing dolphins. The overall accuracy goes down from 0.65 to 0.60, but it does better predicting feeding and socializing dolphins, two things that we considered worth the overall decrease in accuracy. Another important finding from the cross-validation study is that the cost matrix, number of splits, and number of weak learners are all viable options to 'tune' the algorithm to try and improve its predictive abilities.

The final thing we looked at in this cross-validation study was the true positive and positive predictive rate for feeding. The highest true positive rate for feeding from the adjusted cost matrix was 15/28, just better than 1/2. The positive predictive value for the same combination of parameters was 15/173, or just 0.087. The true positive and positive predictive rate for the combination of parameters we consider to create the best classifier were 10/25 and 10/81 respectively. These do not inspire much confidence in the ability to correctly predict future feeding observations. Although we were able to adjust the cost matrix to build a multiclass classifier that does a better job predicting feeding observations, we have to use exercise caution when predicting future feeding observations. The results of our cross-validation study suggest that

Table 3: Confusion matrix of predictions for adjusted cost matrix on the left and standard cost matrix on the right with 50 weak learners and 2 splits

		Predicted			
		FE	MI	SO	TR
Truth	FE	15	3	0	10
	MI	89	86	84	415
	SO	0	3	37	78
	TR	69	16	48	1077

		Predicted			
		FE	MI	SO	TR
Truth	FE	12	3	0	13
	MI	52	140	1	481
	SO	0	21	0	97
	TR	29	12	0	1169

the variables currently collected do contain some information to predict dolphin behavior, but that there may be other variables that would better predict behavior, particularly feeding.

After using all of the data to build our final classifier, of the 194 observations recorded as probably feeding, just 40 of them were predicted as feeding. We also used Equation 2 to calculate the predicted probabilities of being in each class. There were 58 observations with a predicted probability of feeding above 0.40. Although feeding was not the class with the highest predicted probability for all of these, they are observations where we predict that the probability the dolphin is feeding to be moderately likely. Overall, this is a low proportion of the 'probably feeding' observations we predicted to be feeding based on our final classifier. If we had more confidence in the predictive power of our classifier, we would be able to say that the observations currently being marked as probably feeding are often not actually feeding. Unfortunately, based on our cross validation study we do not feel confident making such conclusions despite finding what we believe to be within close range of the best possible classifier given the nature of the data. This again highlights that the variables currently available do not classify dolphin activity as strongly as we would like to be able to accurately predict observations where the activity is unknown.

4 Conclusion

We were successful in our attempt to modify the AdaBoost algorithm to more accurately predict dolphins that were feeding without substantially lowering the overall accuracy. However, the positive predictive and true positive rates for feeding were not high enough to make strong conclusions about our predictions when the observation is recorded as probably feeding. We believe that this is due to the randomness in dolphin activity relative to the variables we had available. Perhaps there are other variables that, if observed, would improve the ability to predict dolphin activity and feeding specifically.

One aspect of this project that we would like to improve is how we chose WR. Though we carefully considered which accuracy metrics to use and how to weigh them, it was ultimately a subjective choice. The advantage is that we could evaluate how well each classifier performed based on what was important to us. The disadvantage is a different researcher working on the same problem may have a different opinion on how WR should be constructed. We would also like to explore the possibility of incorporating the cost matrix into the computation of the error of the classifier $err^{(b)}$ rather than in the impurity function of the decision trees. Another item we would like to explore further, is how to better incentivize predicting rare observations in classification settings. There may be better ways to do so beyond adjusting the cost matrix. As is commonly the case in machine learning methods, we were able to say little about our uncertainty in our predictions and it is difficult to describe the relationships between our predictor variables and the dolphin's activity.

References

Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. *Statistics and its Interface*, 2 (3):349–360, 2009.

5 Appendix

```
library(adabag)
library(tcltk)
# My modified version of the AdaBoost function
multi.adaboost <- function(formula,data,boos=TRUE,mfinal=100,
                           coeflearn="Zhu",priors,cost.matrix,splits=1){
  if(!(as.character(coeflearn)%in%c("Freund","Breiman","Zhu"))){
    stop("coeflearn must be 'Freund', 'Breiman' or 'Zhu' ")
  }
  formula <- as.formula(formula)
  vardep <- data[,as.character(formula[[2]])]
  n <- length(data[,1])
  nclases <- nlevels(vardep)
  pesos <- rep(1/n,n)
  guardarpesos <- array(0,c(n,mfinal))
  w <- rep(1/n,n)
  data <- cbind(pesos,data)
  arboles <- list()
  pond <- rep(0, mfinal)
  pred <- data.frame(rep(0,n))
  arboles[[1]] <- rpart(formula,data=data[,-1],
                        control=rpart.control(minsplit=1,cp=-1,splitsepth=30))
  nvar <- dim(varImp(arboles[[1]],surrogates=FALSE,competes=FALSE))[1]
  imp <- array(0,c(mfinal,nvar))
  for (m in 1:mfinal){
    if (boos==TRUE){
      k <- 1
      while (k==1){
        boostrap <- sample(1:n,replace=TRUE,prob=pesos)
        fit <- rpart(formula,data=data[boostrap,-1],parms=list(prior=priors,loss=cost.matrix),
                     control=rpart.control(splitsepth=splits))
        k <- length(fit$frame$var)
      }
      flearn <- predict(fit,newdata=data[,-1],type="class")
      ind <- as.numeric(vardep!=flearn)
      err <- sum(pesos*ind)
    }
    if (boos==FALSE){
      w <<- pesos
      fit <- rpart(formula = formula, data = data[, -1],
                   weights = w,parms=list(prior=priors,loss=cost.matrix),
                   control=rpart.control(splitsepth=splits))
      flearn <- predict(fit,data=data[,-1],type="class")
      ind <- as.numeric(vardep!=flearn)
      err <- sum(pesos*ind)
    }
    c <- log((1 - err)/err)
    if (coeflearn=="Breiman"){
      c <- (1/2)*c
    }
    if (coeflearn=="Zhu"){
      c <- c+log(nclases-1)
    }
    guardarpesos[,m] <- pesos
    pesos <- pesos*exp(c*ind)
    pesos <- pesos/sum(pesos)
    maxerror <- 0.5
    eac <- 0.001
    if (coeflearn=="Zhu"){
      maxerror <- 1-1/nclases
    }
    if (err>=maxerror){
      pesos <- rep(1/n,n)
      maxerror <- maxerror-eac
      c <- log((1-maxerror)/maxerror)
      if (coeflearn=="Breiman"){
        c <- (1/2)*c
      }
      if (coeflearn=="Zhu"){
        c <- c+log(nclases-1)
      }
    }
    if (err==0){
      pesos <- rep(1/n,n)
      c <- log((1 -eac)/eac)
      if (coeflearn=="Breiman"){
        c <- (1/2)*c
      }
      if (coeflearn=="Zhu"){
        c <- c+log(nclases-1)
      }
    }
    arboles[[m]] <- fit
    pond[m] <- c
    if (m==1){
      pred <- flearn
    }
    else{
      pred <- data.frame(pred,flearn)
    }
    if (length(fit$frame$var)>1){
      k <- varImp(fit,surrogates=FALSE,competes=FALSE)
      imp[m, ] <- k[sort(row.names(k)),]
    }
    else{
      imp[m,] <- rep(0,nvar)
    }
  }
  classfinal <- array(0,c(n,nlevels(vardep)))
  for (i in 1:nlevels(vardep)){
    classfinal[, i] <- matrix(as.numeric(pred==levels(vardep)[i]),nrow=n)%*%as.vector(pond)
  }
  predclass <- rep("O",n)
  predclass[] <- apply(classfinal,1,FUN=which.max)
  imppond <- as.vector(as.vector(pond)%*%imp)
  imppond <- imppond/sum(imppond)*100
  names(imppond) <- sort(row.names(k))
```

```
  votosporc <- classfinal/apply(classfinal,1,sum)
  ans <- list(formula=formula,trees=arboles,weights=pond,
              votes=classfinal,prob=votosporc,class=predclass,
              importance = imppond)
  attr(ans, "vardep.summary") <- summary(vardep,maxsum=700)
  mf <- model.frame(formula=formula,data=data[,-1])
  terms <- attr(mf,"terms")
  ans$terms <- terms
  ans$call <- match.call()
  class(ans) <- "boosting"
  ans
}
# Reading in both sets of data
all.data <- read.csv("dolphin_activity.csv")
pf.data <- read.csv("pf_activity.csv")
# Function that calculates the "goodness" criteria WR
criteria <- function(PPV,TP,prop.TP,prop.PPV){
  sum(diag(prop.TP)*TP+diag(prop.PPV)*PPV)/sum(TP+PPV)
}
##############  Setting up parameters to test  ##############
# Cost matrices to examine
cost.matrices <- list()
cost.matrices[[1]] <- rbind(c(0,1.25,1,1.75),c(1.75,0,0.75,1.25),
                            c(1,1.25,0,1.5),c(1,0.75,0.5,0))
cost.matrices[[2]] <- matrix(1,nrow=4,ncol=4)
diag(cost.matrices[[2]]) <- 0
# Number of trees
num.trees <- c(10,50,100)
# Max depths
max.depths <- c(1,2,8)

loo_cv <- function(costMatrix,num.tree,max.depth,data){
  prediction <- character(nrow(data))
  pb <- txtProgressBar(0,nrow(data),style=3)
  for(i in nrow(data)){
    setTxtProgressBar(pb,i)
    model <- multi.adaboost(activity~depth+HabitatCA+HabitatCH+HabitatGU+HabitatPA+
               HabitatSA+HabitatSG+group.spread+Group.Size+Boats100+illegal1+NN.dist,
               boos=TRUE,data=data[-i,],mfinal=num.tree,coeflearn="Zhu",
               priors=c(0.14,0.24,0.10,0.52),cost.matrix=costMatrix,splits=max.depth)
    prediction[i] <- predict(model,newdata=data[i,])$class
  }
  confusion <- table(prediction,data$activity)
  return(list(confusion,prop.TP=prop.table(confusion,2),prop.PPV=prop.table(confusion,1)))
}
# Leave-one out cross-validation for each combation of parameters
loo111 <- loo_cv(cost.matrices[[1]],num.trees[1],max.depths[1],all.data)
loo112 <- loo_cv(cost.matrices[[1]],num.trees[1],max.depths[2],all.data)
loo113 <- loo_cv(cost.matrices[[1]],num.trees[1],max.depths[3],all.data)
loo121 <- loo_cv(cost.matrices[[1]],num.trees[2],max.depths[1],all.data)
loo122 <- loo_cv(cost.matrices[[1]],num.trees[2],max.depths[2],all.data)
loo123 <- loo_cv(cost.matrices[[1]],num.trees[2],max.depths[3],all.data)
loo131 <- loo_cv(cost.matrices[[1]],num.trees[3],max.depths[1],all.data)
loo132 <- loo_cv(cost.matrices[[1]],num.trees[3],max.depths[2],all.data)
```

```
loo133 <- loo_cv(cost.matrices[[1]],num.trees[3],max.depths[3],all.data)

loo211 <- loo_cv(cost.matrices[[2]],num.trees[1],max.depths[1],all.data)
loo212 <- loo_cv(cost.matrices[[2]],num.trees[1],max.depths[2],all.data)
loo213 <- loo_cv(cost.matrices[[2]],num.trees[1],max.depths[3],all.data)
loo221 <- loo_cv(cost.matrices[[2]],num.trees[2],max.depths[1],all.data)
loo222 <- loo_cv(cost.matrices[[2]],num.trees[2],max.depths[2],all.data)
loo223 <- loo_cv(cost.matrices[[2]],num.trees[2],max.depths[3],all.data)
loo231 <- loo_cv(cost.matrices[[2]],num.trees[3],max.depths[1],all.data)
loo232 <- loo_cv(cost.matrices[[2]],num.trees[3],max.depths[2],all.data)
loo233 <- loo_cv(cost.matrices[[2]],num.trees[3],max.depths[3],all.data)

Function to calculate WR or overall accuracy
calculate_criteria <- function(object,criteria=TRUE){
  if(nrow(object$prop.TP)!=4){
    ind <- !c("F","MI","SO","TR")%in%rownames(object$prop.TP)
    mat1 <- mat2 <- mat3 <- matrix(NA,nrow=4,ncol=4)
    mat1[ind,] <- mat2[ind,] <- mat3[ind,] <- 0
    mat1[!ind,] <- object$prop.TP
    mat2[!ind,] <- object$prop.PPV
    mat3[!ind,] <- object[[1]]
    object$prop.TP <- mat1
    object$prop.PPV <- mat2
    object[[1]] <- mat3
  }
  if(criteria==TRUE){
    return(criteria(c(3,2,1,2),c(4,1,3,1),object$prop.TP,object$prop.PPV))
  } else{
    return(sum(diag(object[[1]]))/nrow(all.data))
  }
}
# Calculating WR for each combination
calculate_criteria(loo111)
calculate_criteria(loo121)
calculate_criteria(loo131)
calculate_criteria(loo112)
calculate_criteria(loo122)
calculate_criteria(loo132)
calculate_criteria(loo113)
calculate_criteria(loo123)
calculate_criteria(loo133)

calculate_criteria(loo211)
calculate_criteria(loo221)
calculate_criteria(loo231)
calculate_criteria(loo212)
calculate_criteria(loo222)
calculate_criteria(loo232)
calculate_criteria(loo213)
calculate_criteria(loo223)
calculate_criteria(loo233)

# 5-fold cross validation
k.fold_cv <- function(costMatrix,num.tree,max.depth,data,k){
  inds <- createFolds(1:nrow(data),k=k)
  prediction <- character(nrow(data))
  for(i in 1:k){
    model <- multi.adaboost(activity~depth+HabitatCA+HabitatCH+HabitatGU+HabitatPA+
                HabitatSA+HabitatSG+group.spread+Group.Size+Boats100+illegal1+NN.dist,
                boos=TRUE,data=data[-inds[[i]],],mfinal=num.tree,coeflearn="Zhu",
                priors=c(0.14,0.24,0.10,0.52),cost.matrix=costMatrix,splits=max.depth)
    prediction[inds[[i]]] <- predict(model,newdata=data[inds[[i]],])$class
  }
  return(prediction)
}

# 5-fold cross validation for every combination
preds111 <- sapply(1:100,function(x) k.fold_cv(cost.matrices[[1]],num.trees[1],
    max.depths[1],all.data,5))
preds112 <- sapply(1:100,function(x) k.fold_cv(cost.matrices[[1]],num.trees[1],
    max.depths[2],all.data,5))
preds113 <- sapply(1:100,function(x) k.fold_cv(cost.matrices[[1]],num.trees[1],
    max.depths[3],all.data,5))
preds121 <- sapply(1:100,function(x) k.fold_cv(cost.matrices[[1]],num.trees[2],
    max.depths[1],all.data,5))
preds122 <- sapply(1:100,function(x) k.fold_cv(cost.matrices[[1]],num.trees[2],
    max.depths[2],all.data,5))
preds123 <- sapply(1:100,function(x) k.fold_cv(cost.matrices[[1]],num.trees[2],
    max.depths[3],all.data,5))
preds131 <- sapply(1:100,function(x) k.fold_cv(cost.matrices[[1]],num.trees[3],
    max.depths[1],all.data,5))
preds132 <- sapply(1:100,function(x) k.fold_cv(cost.matrices[[1]],num.trees[3],
    max.depths[2],all.data,5))
preds133 <- sapply(1:100,function(x) k.fold_cv(cost.matrices[[1]],num.trees[3],
    max.depths[3],all.data,5))

preds211 <- sapply(1:100,function(x) k.fold_cv(cost.matrices[[2]],num.trees[1],
    max.depths[1],all.data,5))
preds212 <- sapply(1:100,function(x) k.fold_cv(cost.matrices[[2]],num.trees[1],
    max.depths[2],all.data,5))
preds213 <- sapply(1:100,function(x) k.fold_cv(cost.matrices[[2]],num.trees[1],
    max.depths[3],all.data,5))
preds221 <- sapply(1:100,function(x) k.fold_cv(cost.matrices[[2]],num.trees[2],
    max.depths[1],all.data,5))
preds222 <- sapply(1:100,function(x) k.fold_cv(cost.matrices[[2]],num.trees[2],
    max.depths[2],all.data,5))
preds223 <- sapply(1:100,function(x) k.fold_cv(cost.matrices[[2]],num.trees[2],
    max.depths[3],all.data,5))
preds231 <- sapply(1:100,function(x) k.fold_cv(cost.matrices[[2]],num.trees[3],
    max.depths[1],all.data,5))
preds232 <- sapply(1:100,function(x) k.fold_cv(cost.matrices[[2]],num.trees[3],
    max.depths[2],all.data,5))
preds233 <- sapply(1:100,function(x) k.fold_cv(cost.matrices[[2]],num.trees[3],
    max.depths[3],all.data,5))

calculate_criteria.5fold <- function(object,criteria=TRUE){
  confusions <- lapply(1:100,function(x) table(object[,x],all.data$activity))
  prop.preds <- lapply(1:100,function(x) prop.table(table(object[,x],all.data$activity),2))
  prop.PPV <- lapply(1:100,function(x) prop.table(table(object[,x],all.data$activity),1))
  check <- sapply(1:100,function(x) nrow(prop.preds[[x]]))
  for(i in 1:100){
    if(check[i]!=4){
      ind <- !c("F","MI","SO","TR")%in%rownames(prop.preds[[i]])
      mat1 <- mat2 <- mat3 <- matrix(NA,nrow=4,ncol=4)
      mat1[ind,] <- mat2[ind,] <- mat3[ind,] <- 0
      mat1[!ind,] <- prop.preds[[i]]
      mat2[!ind,] <- prop.PPV[[i]]
      mat3[!ind,] <- confusions[[i]]
      prop.preds[[i]] <- mat1
      prop.PPV[[i]] <- mat2
      confusions[[i]] <- mat3
    }
  }
  if(criteria==TRUE){
    return(sapply(1:100,function(x)
        criteria(c(3,2,1,2),c(4,1,3,1),prop.preds[[x]],prop.PPV[[x]])))
  } else {
    return(sapply(1:100,function(x) sum(diag(confusions[[x]]))/nrow(all.data)))
  }
}
# Matrix, trees, splits
mean(calculate_criteria.5fold(preds111))
mean(calculate_criteria.5fold(preds121))
mean(calculate_criteria.5fold(preds131))
mean(calculate_criteria.5fold(preds112))
mean(calculate_criteria.5fold(preds122))
mean(calculate_criteria.5fold(preds132))
mean(calculate_criteria.5fold(preds113))
mean(calculate_criteria.5fold(preds123))
mean(calculate_criteria.5fold(preds133))

mean(calculate_criteria.5fold(preds211))
mean(calculate_criteria.5fold(preds212))
mean(calculate_criteria.5fold(preds213))
mean(calculate_criteria.5fold(preds221))
mean(calculate_criteria.5fold(preds222))
mean(calculate_criteria.5fold(preds223))
mean(calculate_criteria.5fold(preds231))
mean(calculate_criteria.5fold(preds232))
mean(calculate_criteria.5fold(preds233))

# Building final classifier
full.model <- multi.adaboost(activity~depth+HabitatCA+HabitatCH+HabitatGU+HabitatPA+
                HabitatSA+HabitatSG+group.spread+Group.Size+Boats100+illegal1+NN.dist,
                data=all.data,mfinal=50, boos=FALSE,coeflearn="Zhu",
                priors=c(0.14,0.24,0.10,0.52),cost.matrix=cost.matrices[[1]],splits=8)
# Predicting probably feeding observations
pred.pf <- predict(full.model,newdata=pf.data)
# Confusion matrix for predictions
pred.pf$confusion
# Bins of predicted probabilities
table(cut(pred.pf$prob[,1],breaks=c(-0.001,0.2,0.4,0.6,0.8,1)))
```