BANTER: A User's Guide to Acoustic Classification

Shannon Rankin and Frederick Archer

Marine Mammal and Turtle Division, SWFSC, NOAA Fisheries

8901 La Jolla, Shores Dr., La Jolla, CA 92037

12 April 2021

1. Introduction

Passive acoustic monitoring is an effective means of monitoring marine mammals; however, the value of acoustic detections depends on our ability to identify the source of the sounds we detect. Manual classification by trained acousticians can be used to develop a set of training data for supervised classification algorithms, such as *BANTER* (Bio-Acoustic eveNT classifiER).

A *BANTER* acoustic classifier is written in open source software, R, and requires minimal human intervention, providing more consistent results with fewer biases and errors. *BANTER* also produces a classification error rate which is a critical component when there is no independent verification of species identity. *BANTER* has been developed in a general manner such that it can be applied to sounds from any source (anthropogenic, terrestrial animals, marine animals).

BANTER is a flexible, hierarchical supervised machine learning algorithm for classifying acoustic events consisting of two stages, each consisting of a set of Random Forest classifiers (Rankin et al. 2017). The first stage is the Detector Model, where individual classification models are built for any number of call types (detection types). The second stage applies the results of the first stage call classifiers and adds any event-level variables to create an Event Model that classifies the *event*. *BANTER* classifiers (Detector and Event Models) are based on the Random Forest supervised learning algorithm.

Random Forest creates a large number of decision trees, each trained on a different subset of the data, aggregating predictions across all trees (the forest). For each decision tree in the forest, some portion of the samples will be left out during the construction of the decision tree (referred to as the Out-Of-Bag or OOB dataset). The model automatically evaluates its own performance by running each of the samples in the OOB dataset through the forest and comparing its predicted group classification to the a-priori designated group. Thus, there is no need for separate cross-validation or another test to get an unbiased estimate of the error. Random Forest can handle a large number of input variables which can be discrete or categorical, and is not prone to issues related to correlated variables. The random subsetting of samples and variables, and use of OOB data prevents overfitting of the model.

Here we present a user guide for the *BANTER* acoustic classification algorithm, using the built-in dataset provided in the *BANTER* package.

2. Methods

At a minimum, *BANTER* requires data to train a classifier which can then be applied to predict species identity on a novel dataset that has the same predictors. Here we will use some of the data provided within the *BANTER* R package for testing.

Once you have training data, you first need to initialize a *BANTER* model. The *BANTER* model can be developed in stages (first the Detector Model, then the Event Model) or as a single unit. We suggest running these separately so that each model can be modified to improve performance and ensure stability. Once the models are optimized, we present options for summarizing and interpreting your results.

This guide was developed based on *BANTER* v0.9.3.

2.1 Data Requirements and Limitations

BANTER has flexible data requirements which allow it to be applied to a wide array of training data. *BANTER* consists of two stages: (1) Detector Model and (2) Event Model. At its core, *BANTER* is an *event classifier*: it classifies a group of sounds observed at the same time. Multiple call type detectors can be considered; if your species of interest only produces a single call type, we have found that minor changes to the detector settings can lead to differences between species that can be informative (see Rankin et al. 2017).

BANTER accepts data in a generic R data frame format. There can be one or more detector data frame and only one event data frame.

The event data frame must have one row per event. The columns must be:

- **event.id** a unique character or number identifying each event.
- **species** a character or number that assigns each event to a given species.
- All other columns will be used as predictor variables for the event.

A detector data frame must have one row per call. The columns must be:

- **event.id** a unique character or number identifying each event. This is used to connect the call to the appropriate event in the event data frame described above.
- **call.id** unique character or number for this call in this detector.
- All other columns will be used as predictor variables for the call.

If you use PAMGuard open source software (pamguard.org), you can process your data and export your data formatted for *BANTER* using the `export_banter()` function in the *PAMpal* package (https://cran.r-project.org/web/packages/PAMpal/PAMpal.pdf (https://cran.r-project.org/web/packages/PAMpal/PAMpal.pdf)).

BANTER cannot accommodate missing data (`NA`). Any predictors with missing data will be excluded from the model. As *BANTER* needs to both train and test the model, there must be a minimum of two events for each species in your model. Any species with fewer than 2 events will be excluded from the model. If a species is excluded from one of your detector models, but occurs in other detector models, then it can be used in the event model.

BANTER is a *supervised* machine learning classification model, and the strength of the classifications necessarily relies on the quality of the training data. Likewise, if you are applying a classifier you built to predict novel data, it is imperitive that the novel data be collected in the same manner, and have the same variables, as the training data. Here we provide tools to help you assess your model, but we recommend that you dive into your data to understand its strengths and limitations.

First, install the following R packages

```
install.packages(c("banter","rfPermute", "dplyr", "ggplot2"))
```

Then load the R packages

```
library(banter)
library(rfPermute)
library(dplyr)
library(ggplot2)
```

2.2 Create *BANTER* Model

The first step requires the initialization of a *BANTER* model with a data.frame of events that you provide.

We will use the data provided in the *BANTER* package (`train.data`). We must first load the training data, and take a look at the first few lines of data.

```
# load example data
data(train.data)
# show names of train.data list
names(train.data)
```

```
[1] "events"    "detectors"
```

The `train.data` object is a list that contains both the event data frame (`train.data$events`) and a list of data frames for each of three call detectors (`train.data$detectors`).

2.2.1 Initialize *BANTER* Model

Once we have our data, the next step is to initialize the *BANTER* model.

```
# initialize BANTER model
bant.mdl <- initBanterModel(train.data$events)
```

BANTER is a hierarchical random forest model, with 2 stages. The first stage is the Detector Model, where a random forest model is created for each detector in your dataset. The second stage is the Event Model, which uses information derived from the Detector Model, along with any additional event level predictors. We can develop each of these models independently, or we can approach them as a single function. Here we will approach the Detector Model and the Event Model separately. Please see https://github.com/EricArcher/banter (https://github.com/EricArcher/banter) for more information on combining the models into a single function.

When the *BANTER* model has been initialized, it is good to check the `summary()` to see the distribution of the number of events per species:

```
# summarize BANTER model
summary(bant.mdl)
```

```
Number of events and model classification rate:
          species num.events
1      D.capensis          7
2       D.delphis        116
3       G.griseus          5
4 G.macrorhynchus          1
5   L.obliquidens         10
6       O.orcinus          1
7  S.coeruleoalba         13
8         Overall        153
```

2.2.2 Adding Detectors

The `addBANTERDetector()` function adds Detectors to your model, where the detector information is tagged by Event. If the detector data is a single data frame, then the name of the detector (for example, “bp” is the “bp” detector) needs to be provided. If detector data is a named list of data frames, the name does not need to be provided (can be NULL). The `addBanterDetector()` function can be called repeatedly to add additional detectors or detectors can be added all at once. If your models require different parameters for different detectors, you may want to model them separately. Here we will lump all detectors into a single Detector Model.

```
# Add BANTER Detectors and Run Detector Models
bant.mdl <- addBanterDetector(
  bant.mdl,
  data = train.data$detectors, # Identify all detectors in the train.data dataset
  ntree = 100, # Number of trees to run. See section on 'Tune BANTER Model' for more infor
mation.
  importance=TRUE, # Retain the importance information for downstream analysis
  sampsize = 2 # Number of samples used for each tree. See section on 'Tune _BANTER_ Mode
l' for more information.
)
```

```
Warning: Detector model (dw): sampsize = 2 is >= species frequencies:
  O.orcinus: 2
These species will be used in the model:
  D.capensis: 350
  D.delphis: 5444
  G.griseus: 103
  G.macrorhynchus: 50
  L.obliquidens: 182
  S.coeruleoalba: 486
```

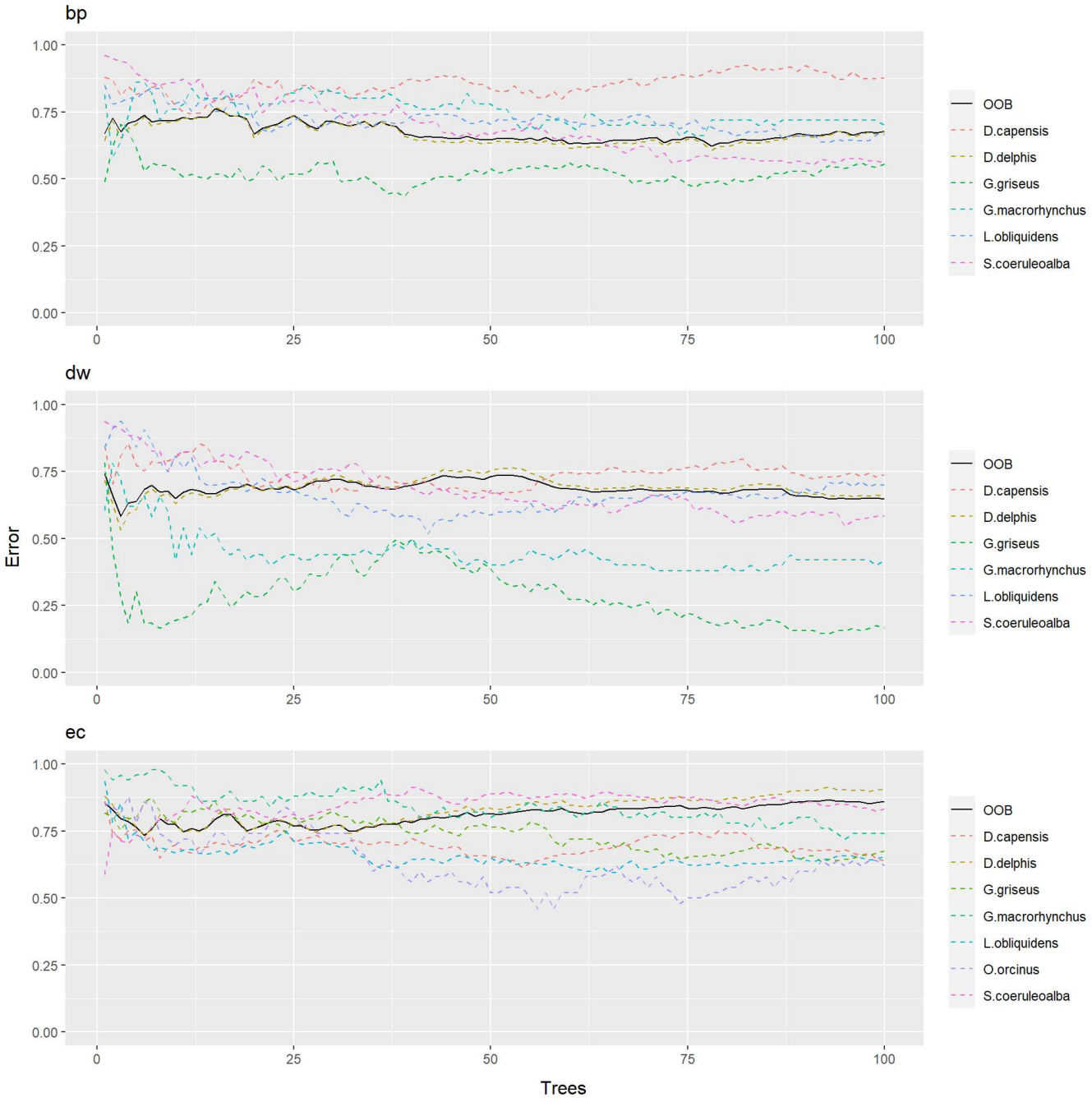
This will create the Random Forest detector models for every detector added. The function will generate reports of species excluded from models due to an insufficient number of samples. When complete, a summary of the model shows mean classification rates of each species in each detector:

```
summary(bant.mdl)
```

```
Number of events and model classification rate:
          species num.events    bp    dw     ec
1      D.capensis          7 12.37 26.29 36.000
2       D.delphis        116 32.42 34.39  9.438
3       G.griseus          5 44.51 83.50 32.400
4 G.macrorhynchus          1 30.00 58.00 26.000
5   L.obliquidens         10 33.55 30.22 34.400
6       O.orcinus          1    NA    NA 38.000
7  S.coeruleoalba         13 44.03 41.77 16.615
8         Overall        153 32.12 35.33 13.980
```

You can then create and examine the Error Trace Plot to determine the stability of your model. You may want to modify the *sampsize* and *ntree* parameters in the model to improve performance and ensure a stable model. See the section on **Tune *BANTER* Model** for more information on interpreting these plots and tuning your model.

```
plotDetectorTrace(bant.mdl)
```



Once you are satisfied with the Detector Model, you are ready to run your final *BANTER* model. This model will include output from the Detector Models, as well as any event-level variables you may have.

This model also uses the *ntree* and *sampsize* parameters, which can be modified to improve performance and model stability. We have purposefully set these values to provide poor results. The next step will be to tune this model to improve performance (see **Tune *BANTER* Model**).

```
bant.mdl <- runBanterModel(bant.mdl, ntree = 10, sampsize = 1)
```

```
Warning: Event model: sampsize = 1 is >= species frequencies:
  G.macrorhynchus: 1
  O.orcinus: 1
These species will be used in the model:
  D.capensis: 7
  D.delphis: 115
  G.griseus: 5
  L.obliquidens: 10
  S.coeruleoalba: 13
```

The next step is to evaluate and tune your model.

2.3 Tune *BANTER* Model

The *BANTER* Models (Detector Models and Event Models) use Random Forest, which is an ensemble approach to classification using a large number of classification trees (`ntree`), where each tree consists of a random sample (n = `sampsize`) and a random number of variables to build a tree. Each tree gives a classification (or 'vote'), and the forest uses the classification having the most votes (trees in the forest). We can tune these two parameters, `ntree` and `sampsize` , to improve performance and ensure stability of the models. Here we will examine the parameters used in the model as well as the summary text and plots of the model, to evaluate the model and tune it to improve the results.

The arguments provided in the Detector and/or Event models include:

- **sampsize = number of samples to use in each tree** The sample size (`sampsize`) is the number of samples randomly selected (without replacement) to build each tree in the 'forest' (model). Increasing `sampsize` leads to a forest that trained on few unique random combinations of samples and may miss patterns in small subsets of the sample space. Decreasing `sampsize` increases the variation from tree to tree in the forest, which strengthens some of the built-in protections against overfitting. However, this may come at the expense of model performance which can be addressed by increasing the number of trees in the forest (`ntree`).

The model will use n = `sampsize` samples for creating each tree in the model, and the remaining samples will be used as out-of-bag (OOB) for model testing. At a maximum, `sampsize` should be half of the smallest sample size of all species, which ensure a balanced and unbiased model. Models will run faster for low sample sizes and large number of trees, rather than vice-versa (there is little computational cost to running a very large number of trees). Simulated tests showed that we can obtain the same performance with sample sizes as low as 1-2 per species and very large numbers of trees (F. Archer, unpublished methods).

- **ntree = number of trees** There is a low computational cost to increasing the number of trees, so we recommend increasing the number of trees until the classification results are extremely stable (see the `plotDetectorTrace()` function). Each tree is based on a random subset of variables, and therefore, the more trees you run in your model, the more you can reduce the variance. Therefore, you want to increase `ntree` until the classification results are stable. In the Error Trace plot below, you want any model variation (vertical movement in any lines) to occur in the first 1-5% of the trace, resulting in a trace that is primarily flat (stable).
- **importance = TRUE** Importance in Random Forest is a measure of the predictive power of a variable. This variable will be used in downstream processing, and we recommend setting `importance=TRUE` to save these values in your *BANTER* detector model (it is automatically saved in the event model). As a tree is trained, a permutation experiment is conducted that scrambles the

predictor values. If this scrambling increases the final error rate, then this variable is a relatively important predictor. However, if this experiment shows that changes to the value of this variable do not impact the overall error rate, then this variable is not as important.

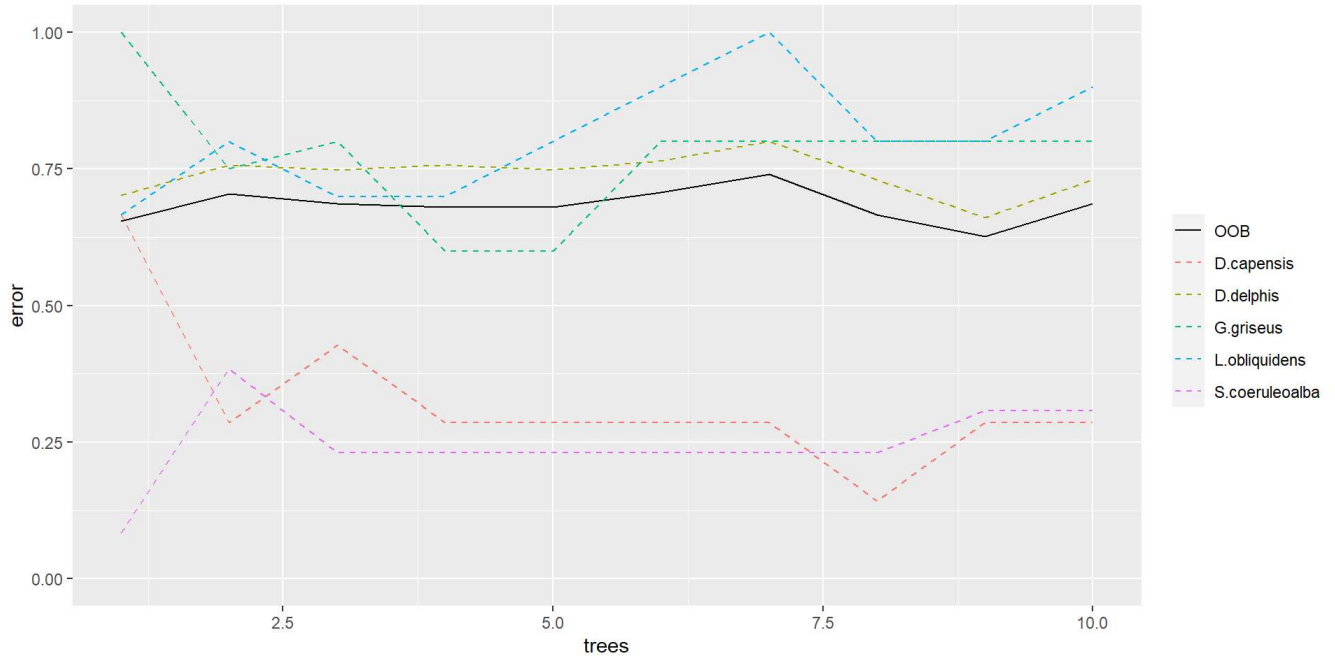
- **num.cores = number of cores to use for Random Forest model** `num.cores` refers to the number of cores used by your computer in processing data. The default is `num.cores = 1`, but it can be set to a maximum of 1 less than the number of cores available on your computer. If `num.cores` is set to >1, the importance variables cannot be saved. While there may be value in increasing the `num.cores` during preliminary processing (to 'tune' the model), we recommend reducing `num.cores = 1` for the final processing in order to allow for `importance = TRUE`.

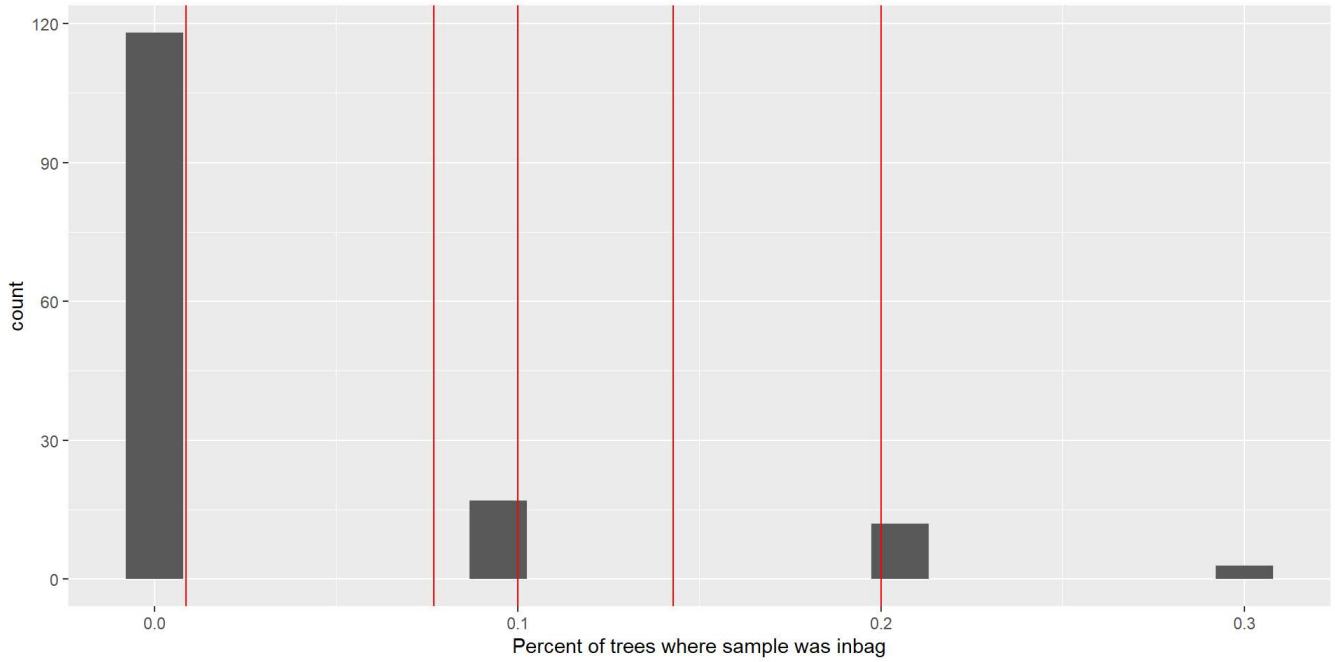
It is important that your *BANTER* model is stable: the results should not change when you rerun the model. We will explain how to tune the model using the case of the poor performing *BANTER* Event Model we created above. These same methods can be applied to the Detector Models, to ensure that your stage 1 models are stable (in this small case they are reasonably stable).

The first tool we have is the Error Trace plot (top plot after you run the summary function, below), which shows the error (y-axis) as we average across an increasing number of trees (x-axis). The goal is to have a stable Error Trace (flat lines). The second tool we have is the count of the percentage of trees where a samples was 'inbag'. You can get these plots by applying the summary function to your *BANTER* model after the Event model has been run.

```
summary(bant.mdl)
```

```
Event model run completed at 2021-04-12 11:57:36
Number of events and model classification rate:
           species num.events    bp    dw     ec event
1      D.capensis          7 12.37 26.29 36.000 71.43
2       D.delphis        116 32.42 34.39  9.438 26.96
3       G.griseus          5 44.51 83.50 32.400 20.00
4 G.macrorhynchus          1 30.00 58.00 26.000    NA
5   L.obliquidens         10 33.55 30.22 34.400 10.00
6       O.orcinus          1    NA    NA 38.000    NA
7  S.coeruleoalba         13 44.03 41.77 16.615 69.23
8         Overall        153 32.12 35.33 13.980 31.33

Distribution of percent correctly classified overall in last 'n' trees:
      n    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  100.0    31.3    31.3    31.3    31.3    31.3    31.3

Sample inbag rate distribution:
          Min. 1st Qu. Median  Mean 3rd Qu. Max.
expected 0.009   0.077    0.1 0.106   0.143  0.2
observed 0.000   0.000    0.0 0.033   0.000  0.3

Confusion matrix:
               D.capensis D.delphis G.griseus L.obliquidens S.coeruleoalba
D.capensis              5         0         0             0              2
D.delphis              75        31         0             0              9
G.griseus               0         1         1             0              3
L.obliquidens           1         1         4             1              3
S.coeruleoalba          3         0         1             0              9
Overall                NA        NA        NA            NA             NA
               pct.correct LCI_0.95 UCI_0.95 Prior
D.capensis            71.4     29.0     96.3   4.7
D.delphis             27.0     19.1     36.0  76.7
G.griseus             20.0      0.5     71.6   3.3
L.obliquidens         10.0      0.3     44.5   6.7
S.coeruleoalba        69.2     38.6     90.9   8.7
Overall               31.3     24.0     39.4  60.3
```





The top plot is Error Trace, or the trace of the error by the number of trees. This gives you an idea of the stability of the model. This plot is created using the `plotRFtrace()` function from `rfPermute`.The lower plot is Inbag distribution plot, or a count by trees where the sample is 'In Bag' (used in the training dataset), the red lines are the expected 'in bag' frequency for this model. This plot provides information on the minimum % of trees that every sample should have been in, and this gives a representation of the samples in the model. We want to use all of the samples, so we want enough trees that the majority of the trees were used and that they were used in the appropriate rate. If too few trees were used, this plot would show peaks at zero and the distributions will not be centered around the red lines. Ideally, the distributions should be tightly centered on the red lines.

To tune the model, you want to run enough trees that the error trace is flat, with the noise occuring in the first 5% of the error trace plot, and you want the Inbag distribution to show the frequency of inbag samples centered around the red lines.

Remember that for our *BANTER* model, we used `sampsize = 1` and `ntree = 10` (`bant.mdl <- run_BANTER_Model(bant.mdl, ntree = 10, sampsize = 1`). Clearly these were insufficient. We will need to increase the sample size and/or the number of trees in our model to improve performance. We suggest first increasing `ntree` until the trace is flat (or close), and then increasing `sampsize` incrementally until you are satisfied with the performance. Remember that it is best to keep `sampsize` less than or equal to half of the smallest species frequency.

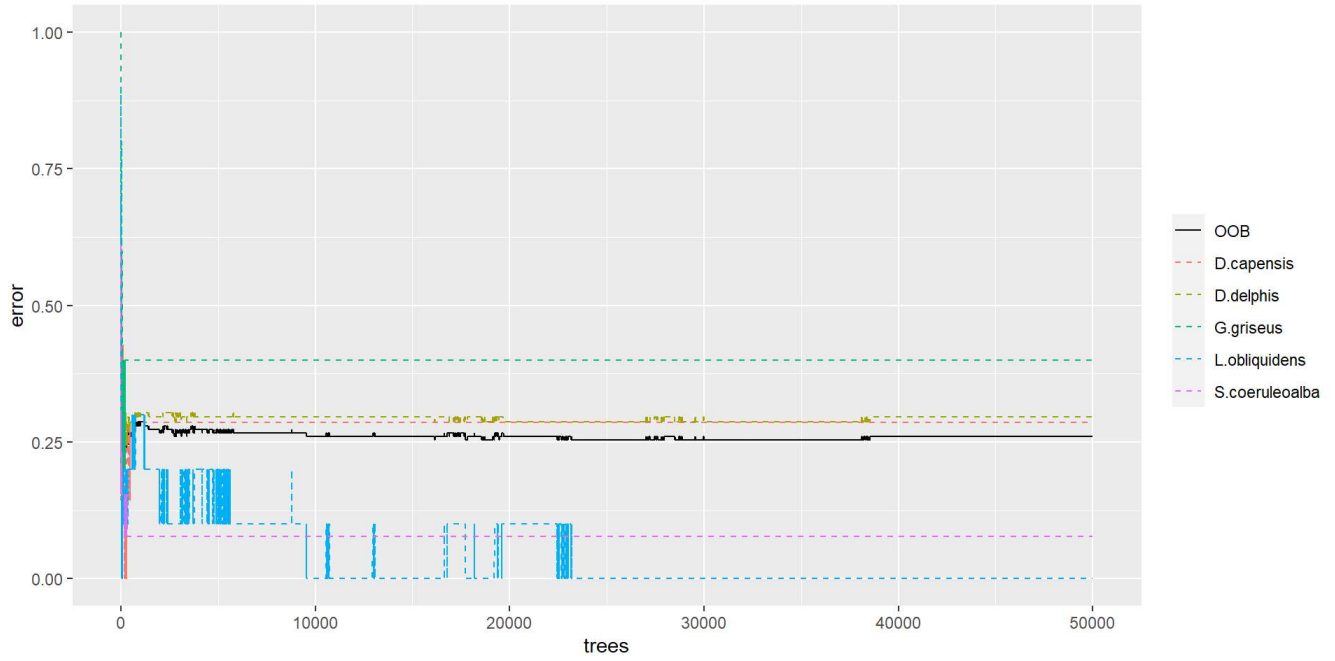
Here we will rerun our model with an improved set of parameters and examine the difference in the results and summary information.

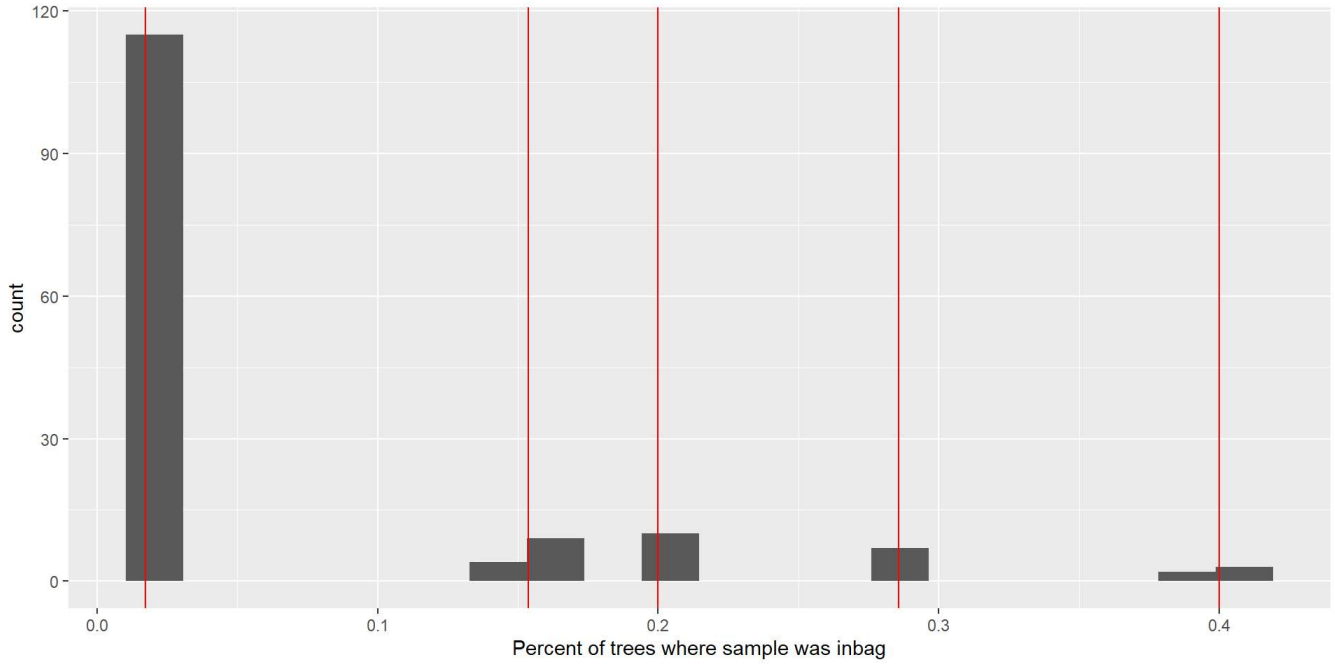
```
bant.mdl <- runBanterModel(bant.mdl, ntree = 50000, sampsize = 2)
```

```
Warning: Event model: sampsize = 2 is >= species frequencies:
  G.macrorhynchus: 1
  O.orcinus: 1
These species will be used in the model:
  D.capensis: 7
  D.delphis: 115
  G.griseus: 5
  L.obliquidens: 10
  S.coeruleoalba: 13
```

```
summary(bant.mdl)
```

```
Event model run completed at 2021-04-12 11:57:43
Number of events and model classification rate:
           species num.events    bp    dw     ec  event
1      D.capensis          7 12.37 26.29 36.000  71.43
2       D.delphis        116 32.42 34.39  9.438  70.43
3       G.griseus          5 44.51 83.50 32.400  60.00
4 G.macrorhynchus          1 30.00 58.00 26.000     NA
5   L.obliquidens         10 33.55 30.22 34.400 100.00
6       O.orcinus          1    NA    NA 38.000     NA
7  S.coeruleoalba         13 44.03 41.77 16.615  92.31
8         Overall        153 32.12 35.33 13.980  74.00

Distribution of percent correctly classified overall in last 'n' trees:
      n    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 500000      74      74      74      74      74      74

Sample inbag rate distribution:
          Min. 1st Qu. Median  Mean 3rd Qu.  Max.
expected 0.017   0.154  0.200 0.211   0.286 0.400
observed 0.016   0.017  0.018 0.067   0.019 0.404

Confusion matrix:
               D.capensis D.delphis G.griseus L.obliquidens S.coeruleoalba
D.capensis              5         1         0             0              1
D.delphis              25        81         1             0              8
G.griseus               0         0         3             1              1
L.obliquidens           0         0         0            10              0
S.coeruleoalba          1         0         0             0             12
Overall                NA        NA        NA            NA             NA
               pct.correct LCI_0.95 UCI_0.95 Prior
D.capensis            71.4     29.0     96.3   4.7
D.delphis             70.4     61.2     78.6  76.7
G.griseus             60.0     14.7     94.7   3.3
L.obliquidens        100.0     69.2    100.0   6.7
S.coeruleoalba        92.3     64.0     99.8   8.7
Overall               74.0     66.2     80.8  60.3
```





Once you are satisfied with your model, you can extract the Random Forest model (and model data) as separate objects for further analysis.

```
bant.rf <- getBanterModel(bant.mdl)
bantData.df <- getBanterModelData(bant.mdl)
```

You can also save each Detector Model for downstream processing.

```
bant.dw.rf <- getBanterModel(bant.mdl, "dw")
bant.bp.rf <- getBanterModel(bant.mdl, "bp")
bant.ec.rf <- getBanterModel(bant.mdl, "ec")
```

You are now ready to summarize and interpret your models and results.

2.4 Interpret *BANTER* Results

The `summary()` function provides information regarding your model results; however, conducting a 'deep dive' into these results will give you a better understanding of the strengths and limitations of your results and may guide you towards improving those results. Here we explain a number of options for interpreting your *BANTER* results.

Model Information

Detector Names & Sample Sizes Show the Detector Names and Sample Sizes

```
# Get detector names for your _BANTER_ Model
getDetectorNames(bant.mdl)
```

```
[1] "bp" "dw" "ec"
```

```
# Get Sample sizes
getSampSize(bant.mdl)
```

```
    D.capensis      D.delphis      G.griseus  L.obliquidens S.coeruleoalba
             2              2              2              2              2
```

Number of Calls & Events, Proportion of Calls Number of calls (`numCalls()`), proportion of calls (`propCalls()`) and number of events (`numEvents()`) in your *BANTER* detector models (or specify by event/species)

```
# number of calls in detector model
numCalls(bant.mdl)
```

```
          species num.bp num.dw num.ec
1      D.capensis    283    350    350
2       D.delphis   4837   5444   5732
3       G.griseus    182    103    250
4 G.macrorhynchus     50     50     50
5   L.obliquidens    307    182    500
6       O.orcinus      0      0     50
7  S.coeruleoalba    134    486    650
```

```
# number of calls by species (can also do by event)
numCalls(bant.mdl, "species")
```

```
          species num.bp num.dw num.ec
1      D.capensis    283    350    350
2       D.delphis   4837   5444   5732
3       G.griseus    182    103    250
4 G.macrorhynchus     50     50     50
5   L.obliquidens    307    182    500
6       O.orcinus      0      0     50
7  S.coeruleoalba    134    486    650
```

```
# proportion of calls in detector model
propCalls(bant.mdl)
```

```
          species   prop.bp   prop.dw   prop.ec
1      D.capensis 0.2878942 0.3560529 0.3560529
2       D.delphis 0.3020671 0.3399738 0.3579592
3       G.griseus 0.3401869 0.1925234 0.4672897
4 G.macrorhynchus 0.3333333 0.3333333 0.3333333
5   L.obliquidens 0.3104146 0.1840243 0.5055612
6       O.orcinus 0.0000000 0.0000000 1.0000000
7  S.coeruleoalba 0.1055118 0.3826772 0.5118110
```

```
# proportion of calls by event (can also do by species)
#propCalls(bant.mdl, "event")
#[this is commented out as printout is long]

# number of events, with default for Event Model
numEvents(bant.mdl)
```

```
          species num.events
1      D.capensis          7
2       D.delphis        116
3       G.griseus          5
4 G.macrorhynchus          1
5   L.obliquidens         10
6       O.orcinus          1
7  S.coeruleoalba         13
```

```
# number of events for a specific detector
numEvents(bant.mdl, "bp")
```

```
          species num.events
1      D.capensis          7
2       D.delphis        113
3       G.griseus          5
4 G.macrorhynchus          1
5   L.obliquidens         10
6       O.orcinus          0
7  S.coeruleoalba         11
```

Confusion Matrix The Confusion Matrix is the most commonly used output for a Random Forest model, and is provided by `summary()`. The output includes the percent correctly classified for each species, the lower and upper confidence levels, and the priors (expected classification rate).

By default, `summary()` reports the 95% confidence levels of the percent correctly classified. By using the `confusionMatrix()` function, we can specify a different confidence level if desired. However, unlike `summary()`, `confusionMatrix()` takes a `randomForest` object like the one we extracted above.

```
# Confusion Matrix
confusionMatrix(bant.rf, conf.level = 0.75)
```

```
               D.capensis D.delphis G.griseus L.obliquidens S.coeruleoalba
D.capensis              5         1         0             0              1
D.delphis              25        81         1             0              8
G.griseus               0         0         3             1              1
L.obliquidens           0         0         0            10              0
S.coeruleoalba          1         0         0             0             12
Overall                NA        NA        NA            NA             NA
               pct.correct LCI_0.75 UCI_0.75 Prior
D.capensis            71.4     42.7     91.0   4.7
D.delphis             70.4     64.9     75.5  76.7
G.griseus             60.0     26.9     87.2   3.3
L.obliquidens        100.0     81.2    100.0   6.7
S.coeruleoalba        92.3     74.9     99.0   8.7
Overall               74.0     69.4     78.2  60.3
```

The `confusionMatrix()` function also has a `threshold` argument that provides the binomial probability that the true classification probability (given infinite data) is greater than or equal to this value. For example, if we want to know what is probability that the true classification probability for each species is >= 0.80, we set `threshold = 0.8`:

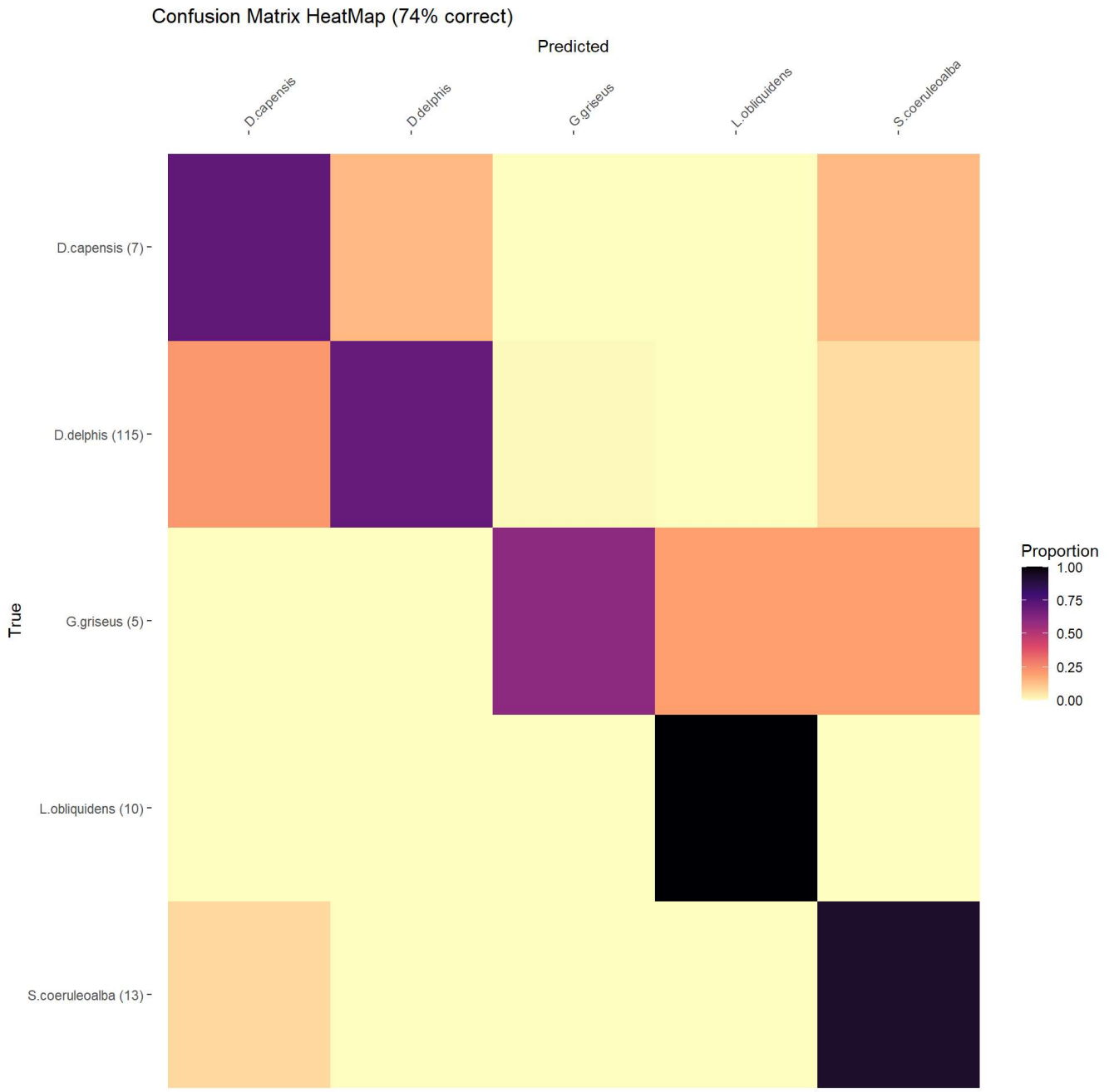
```
# Confusion Matrix with medium threshold
confusionMatrix(bant.rf, threshold = 0.8)
```

```
               D.capensis D.delphis G.griseus L.obliquidens S.coeruleoalba
D.capensis              5         1         0             0              1
D.delphis              25        81         1             0              8
G.griseus               0         0         3             1              1
L.obliquidens           0         0         0            10              0
S.coeruleoalba          1         0         0             0             12
Overall                NA        NA        NA            NA             NA
               pct.correct LCI_0.95 UCI_0.95 Pr.gt_0.8 Prior
D.capensis            71.4     29.0     96.3      42.3   4.7
D.delphis             70.4     61.2     78.6       0.9  76.7
G.griseus             60.0     14.7     94.7      26.3   3.3
L.obliquidens        100.0     69.2    100.0     100.0   6.7
S.coeruleoalba        92.3     64.0     99.8      94.5   8.7
Overall               74.0     66.2     80.8       4.5  60.3
```

This shows that *D. capensis* has a high probability of having a true classification score above 0.8 (Pr.gt_0.8 = 79.0). Conversely, the probability that the classification rate for *D.delphis* is above 0.8 is very low (Pr.gt_0.8 = 6.8).

And alternative view of the confusion matrix comes in the form of a heat map.

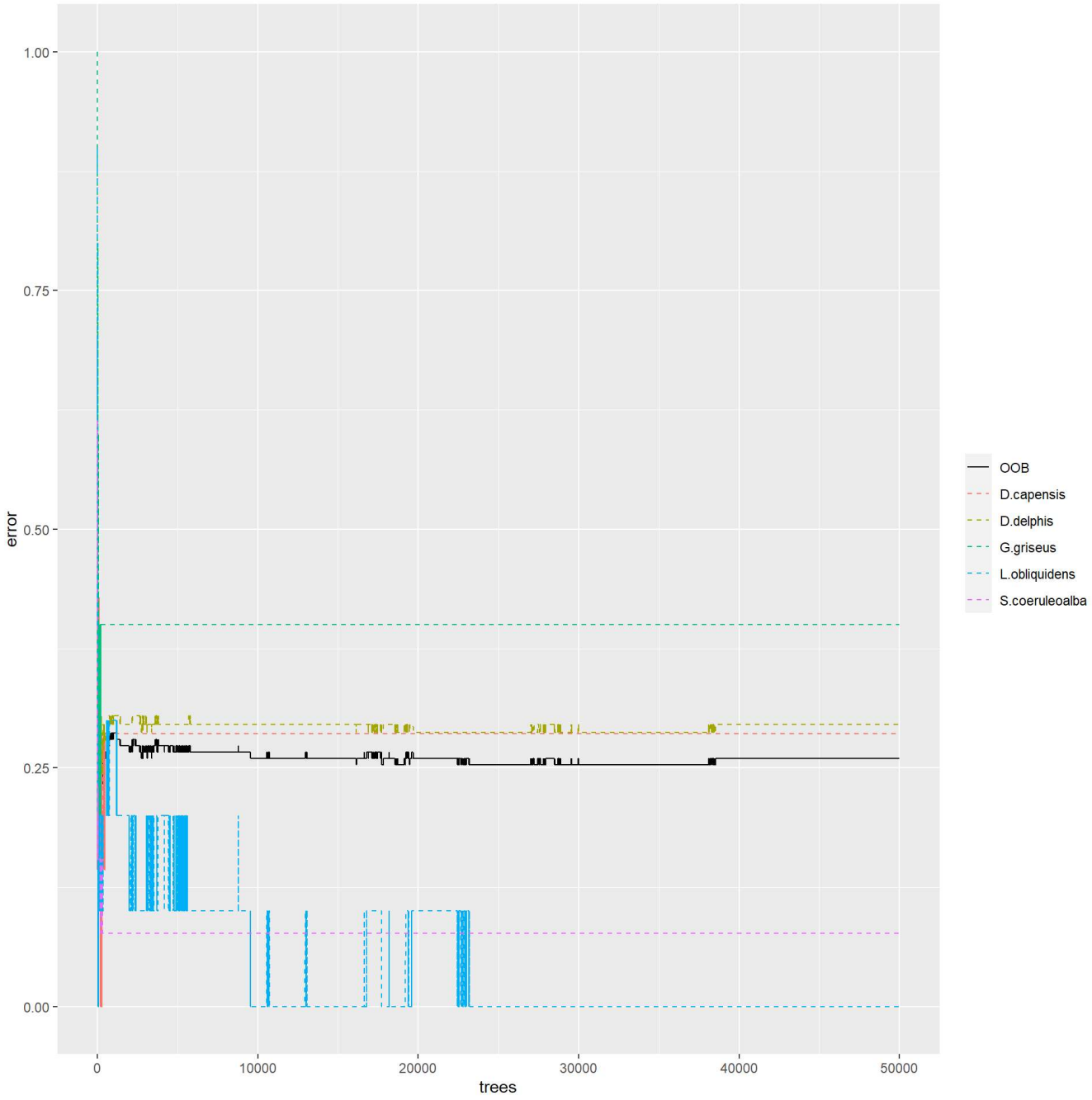
```
# Plot Confusion Matrix Heatmap
plotConfMat(bant.rf, title="Confusion Matrix HeatMap")
```



Plot Random Forest Trace

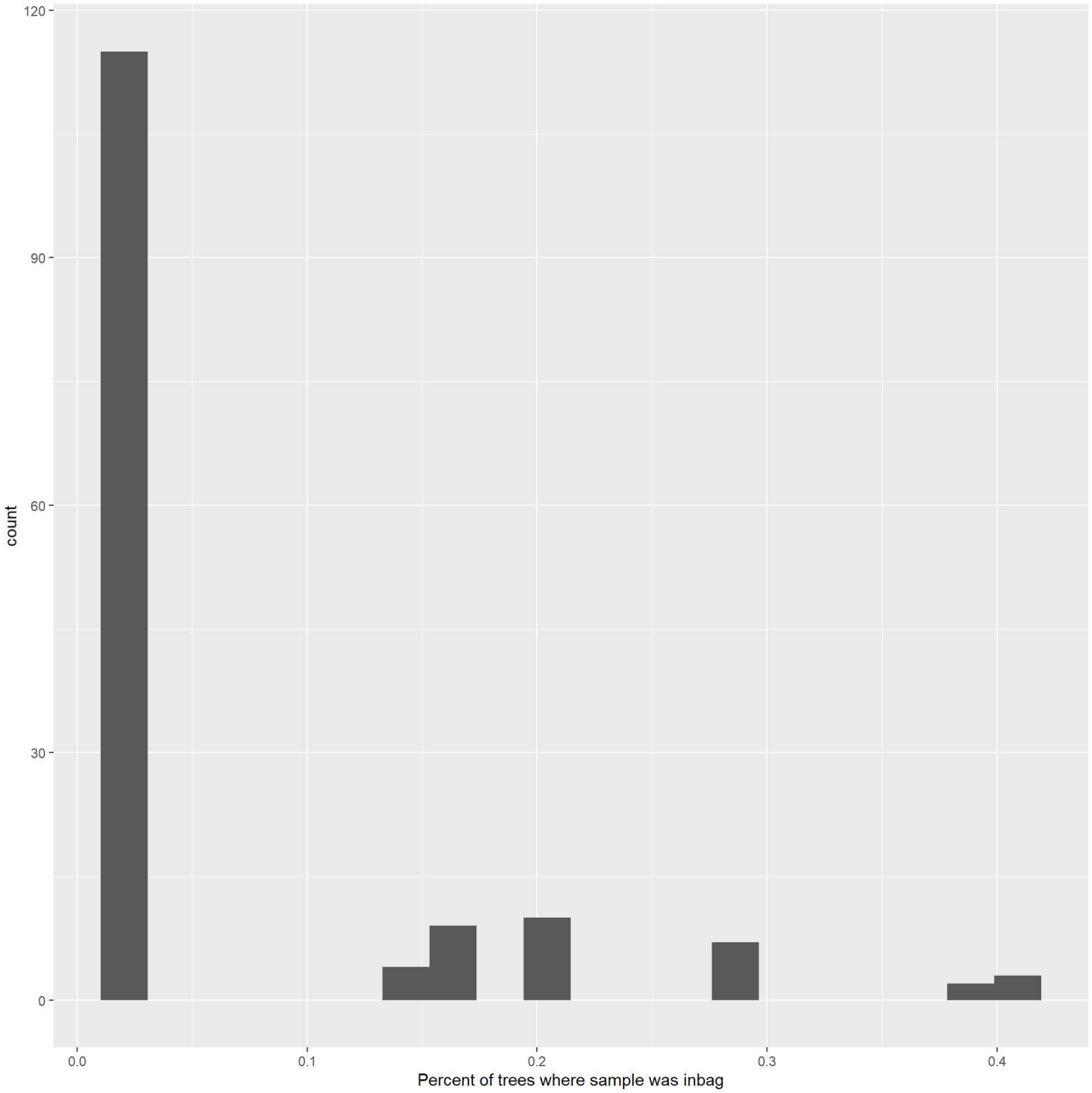
The `plotRFtrace()` function allows us to plot the Error Trace directly.

```
# Plot trace of OOB error rate by number of trees
plotRFtrace(bant.rf)
```

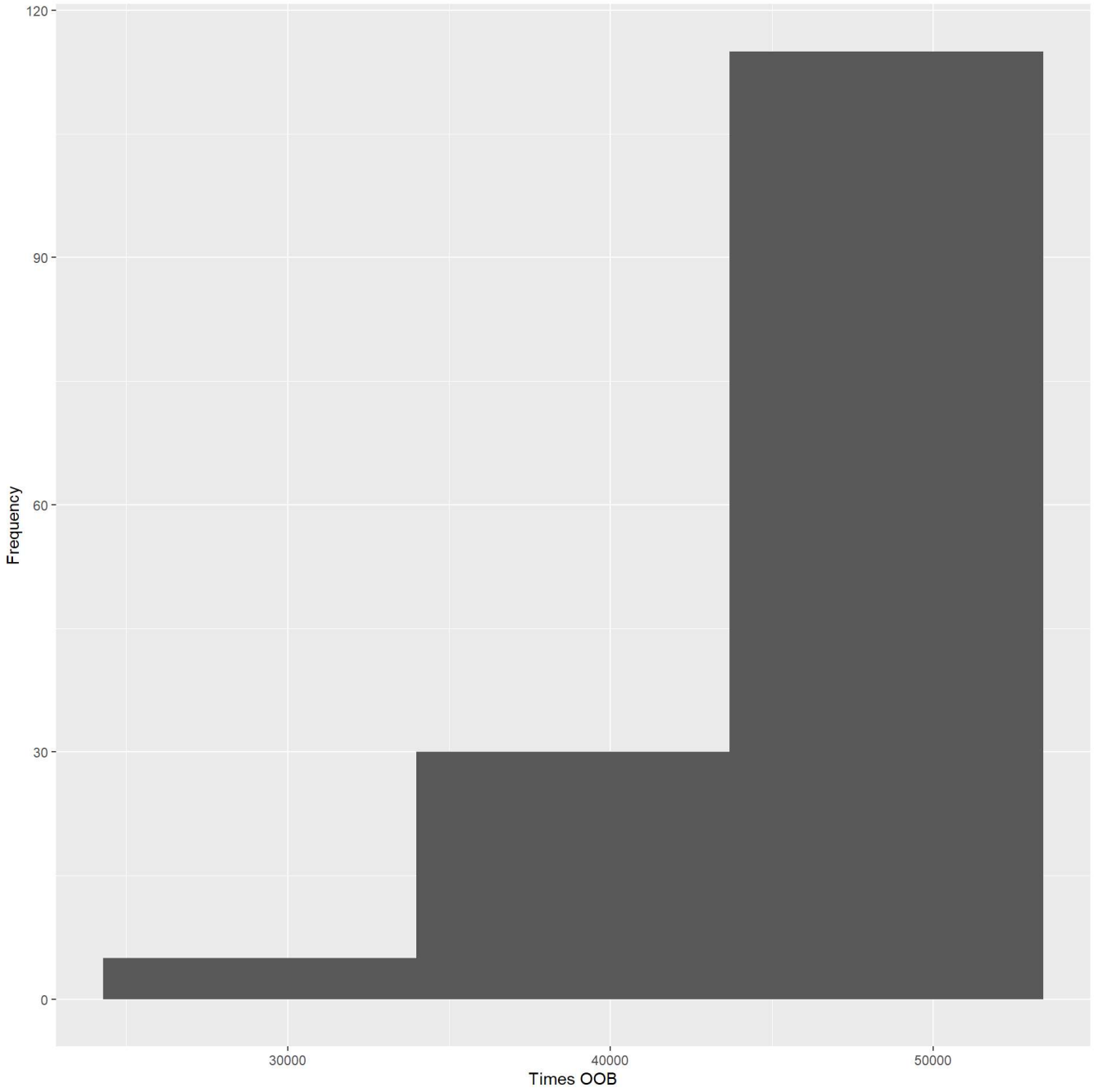


Plot In-Bag Distributions, Importance Null Distributions, and Histogram of OOB Samples The In-Bag samples are the events used in the model. The InBag distribution plot provides a visual for the percent of trees where the sample was in-ba The OOB plot provides a visual for the number of times a sample was out of bag. A high OOB and a low InBag suggest highly random sampling.

```
# Plot inbag distribution
plotInbag(bant.rf)
```




```
# Plot histogram of times samples were OOB
plotOOBtimes(bant.rf)
```



Percent Correct and Expected Error Rate A measure of how well a classifier works is to compare the percent correct score for a given threshold (specified percent of trees in the forest voting for that species) with the error rate you would expect based on random assignment and class sizes.

```
# Percent Correct for a series of thresholds
pctCorrect(bant.rf, pct = c(seq(0.2, 0.6, 0.2), 0.95))
```

```
           class pct.correct_0.2 pct.correct_0.4 pct.correct_0.6
1     D.capensis        71.42857        71.42857       14.285714
2      D.delphis        70.43478        64.34783        9.565217
3      G.griseus        60.00000        60.00000        0.000000
4  L.obliquidens       100.00000        50.00000        0.000000
5 S.coeruleoalba        92.30769        84.61538        7.692308
6        Overall        74.00000        65.33333        8.666667
  pct.correct_0.95
1                0
2                0
3                0
4                0
5                0
6                0
```

```
# Expected Error Rate
exptdErrRate(bant.rf)
```

```
    D.capensis      D.delphis      G.griseus  L.obliquidens S.coeruleoalba
     0.9533333      0.2333333      0.9666667      0.9333333      0.9133333
           OOB
     0.3969778
```

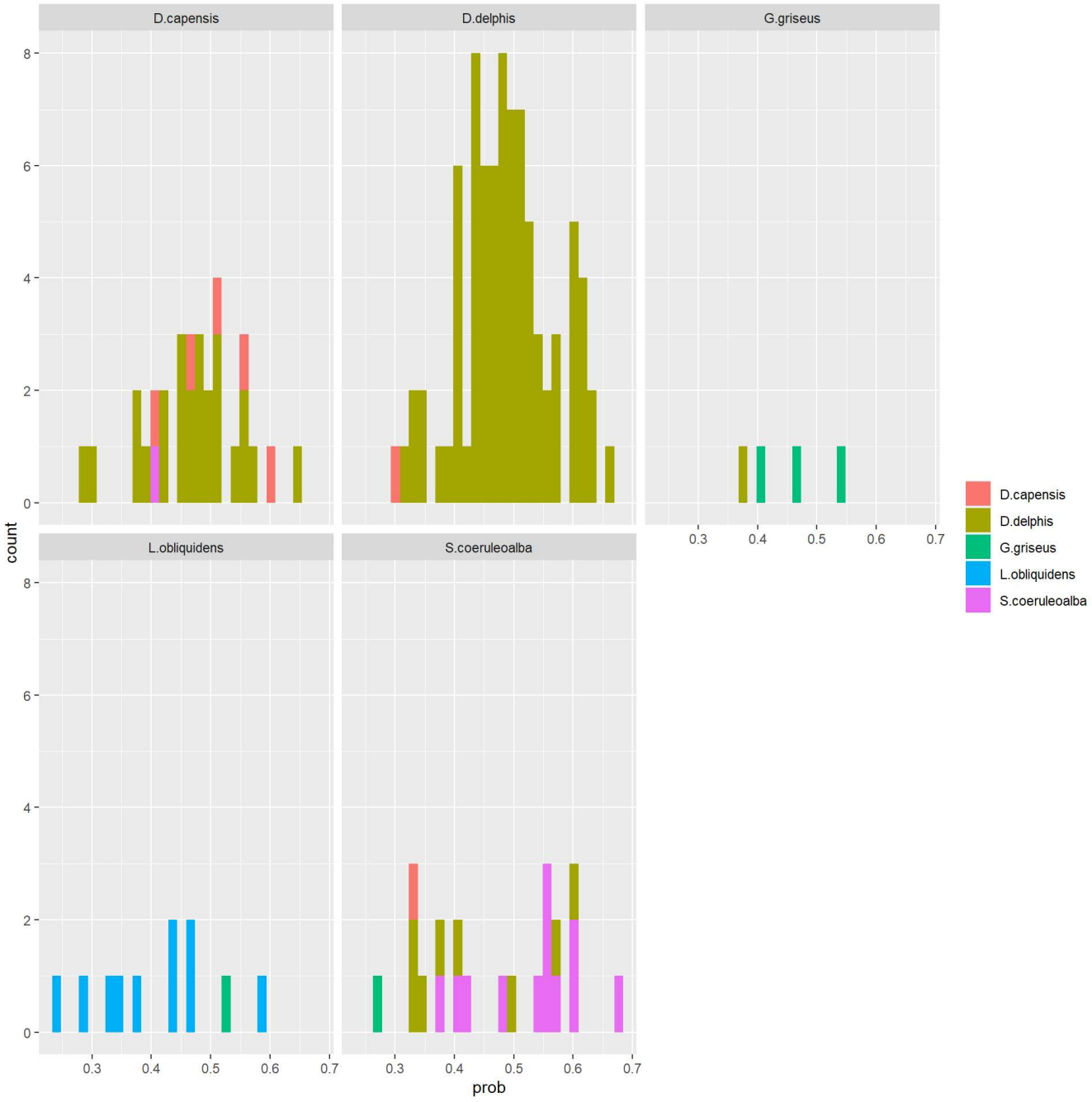
Model Percent Correct Provides a summary data frame with the % correctly classified for each detector model and the event model.

```
modelPctCorrect(bant.mdl)
```

```
# A tibble: 8 x 5
  species            bp    dw    ec event
  <fct>           <dbl> <dbl> <dbl> <dbl>
1 D.capensis       12.4  26.3 36     71.4
2 D.delphis        32.4  34.4  9.44  70.4
3 G.griseus        44.5  83.5 32.4   60
4 G.macrorhynchus  30    58.0 26     NA
5 L.obliquidens    33.6  30.2 34.4  100
6 O.orcinus        NA    NA   38     NA
7 S.coeruleoalba   44.0  41.8 16.6   92.3
8 Overall          32.1  35.3 14.0   74
```

Plot Predicted Probabilities Histograms of the assignment probabilities to the predicted species class. Ideally, all events would be classified to the correct species (identified by the color), and would be strongly classified to the correct species (higher probablity of assignment). This plot can be used to understand the distribution of these classifications, and how strong the misclassifications were, by species.

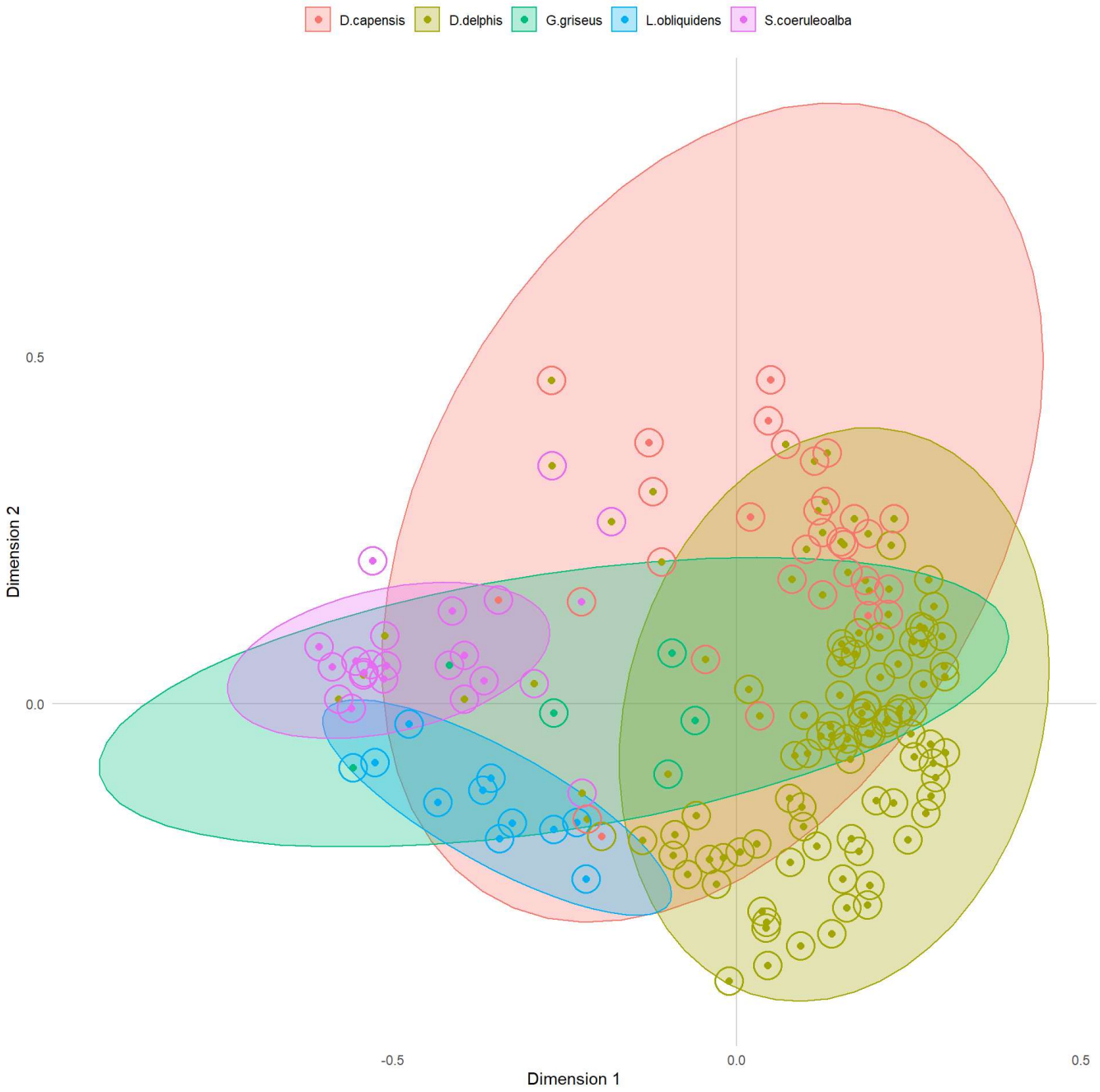
```
plotPredictedProbs(bant.rf, bins = 30, plot = TRUE)
```



Model Interpretation

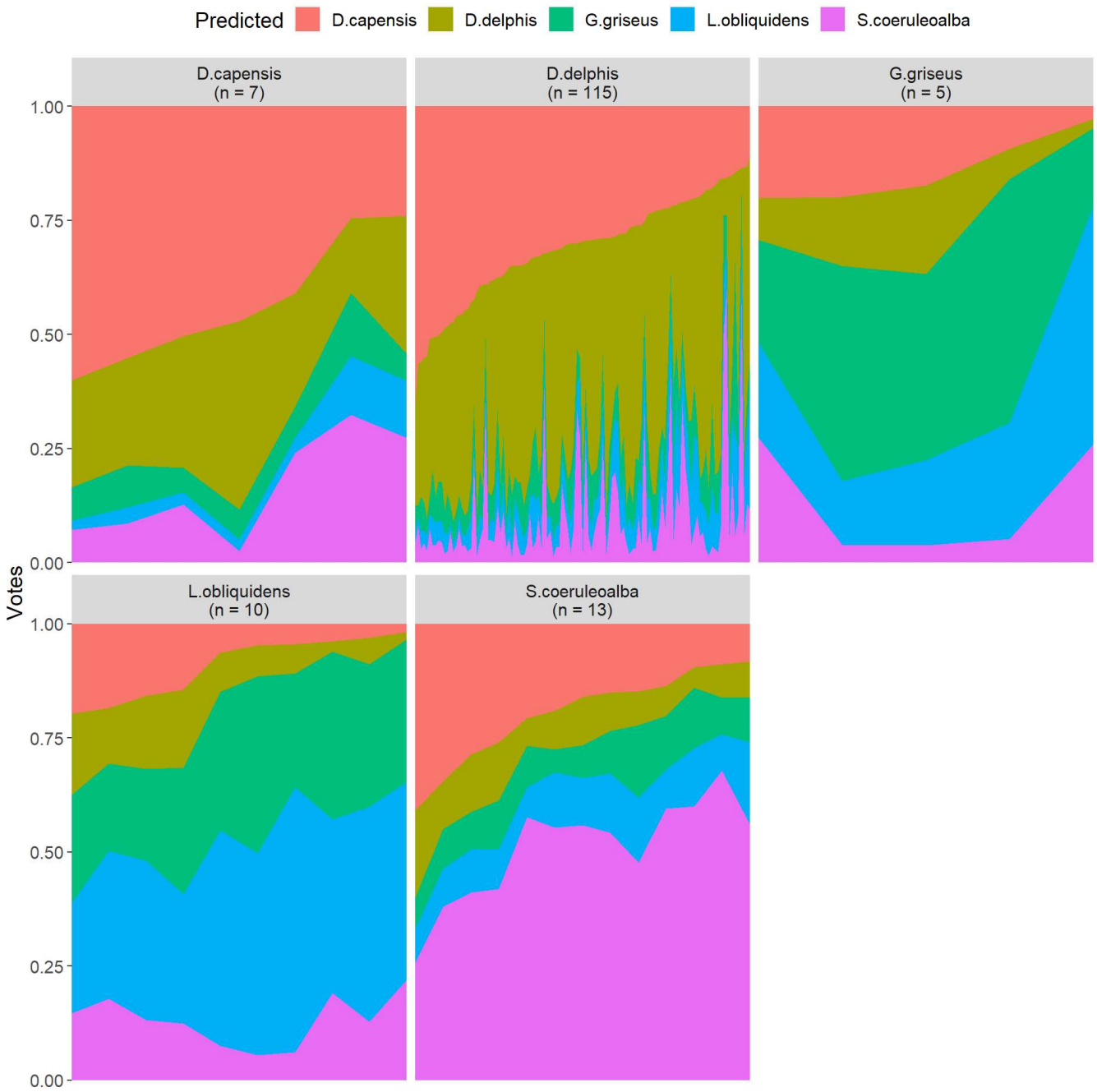
Proximity Plot The proximity plot provides a view of the distribution of events within the tree space. It shows the relative distance of events based on their average distance in nodes in the trees across the forest. For each event in the plot, the color of the central dot represents the true species identity, and the color of the circle represents the *BANTER* classification. Ideally, these would form rather distinct clusters, one for each species. The wider the spread of the events in this feature space, the more variation found in these predictors. Some species differentiation may be predicted by other predictors and may not be clear based on this pair of dimensions (those may be differentiated with different predictors).

```
# Proximity Plot
proximityPlot(bant.rf)
```



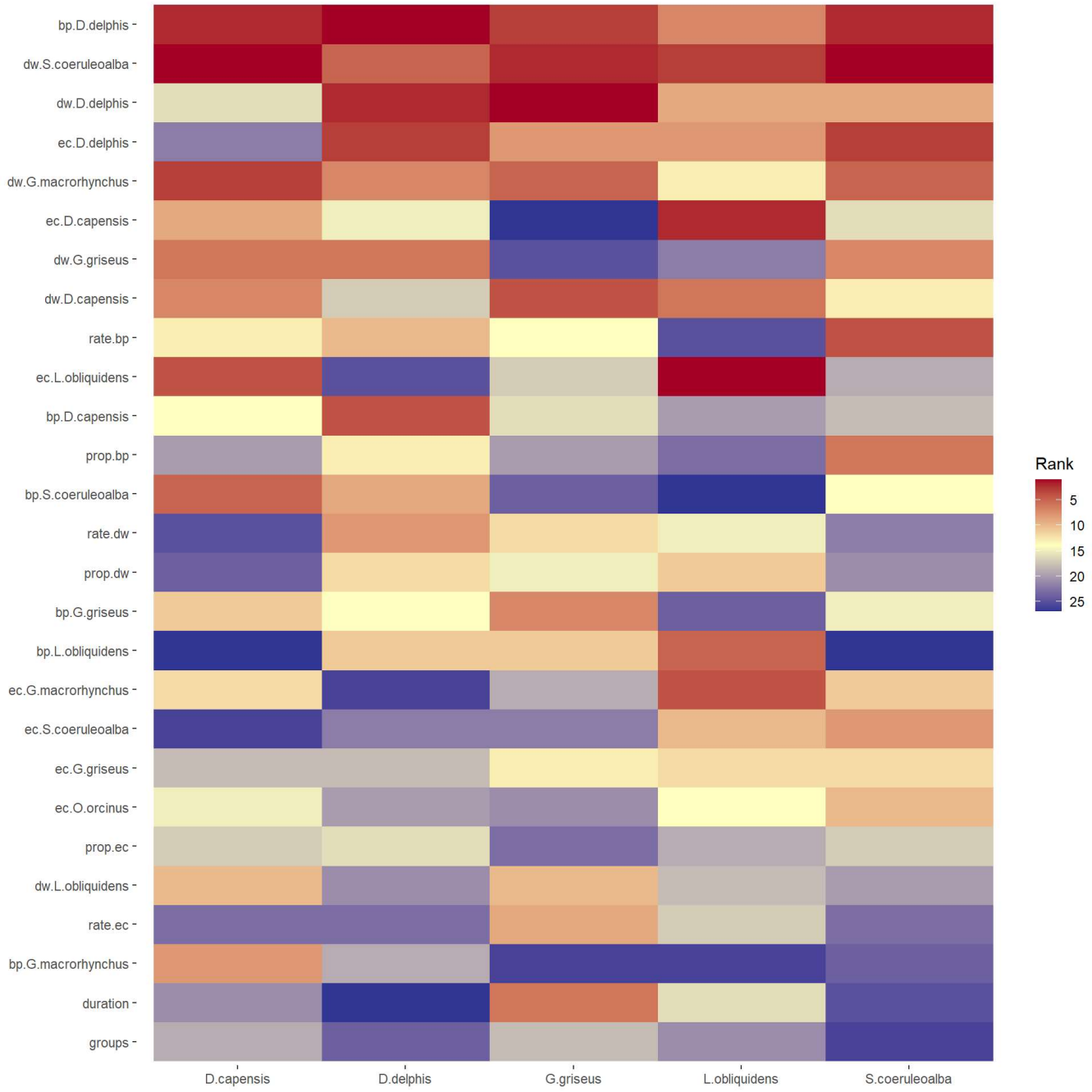
Plot Votes The strength of a classification model depends on the number of trees that 'voted' for the correct species. We can look at the votes from each of these 5,000 trees for an event to see how many of them were correct. This plot shows these votes where each vertical slice is an event, and the percentage of votes for each species is represented by their color. If a species were to be correctly classified by all of the trees (votes) in the forest, then the plot for that species would be solid in the color that represents that species.

```
# Plot Vote distribution
plotVotes(bant.rf)
```



Importance Heat Map The importance heat map provides a visual assessment of the important predictors for the overall model. The *BANTER* event model relies on the mean assignment probability for each of the detectors in our detector model, as well as any event level measures. For example, in this heat map, the first variable is 'dw.D.delphis', which is the mean probability that a detection was assigned to the species 'D.delphis' in the whistle detector. This requires extra steps to dig down to the specific whistle measures that are the important predictor variables for the whistle detector.

```
# Importance Heat Map
impHeatmap (bant.rf)
```



Mis-Classified Events

By segregating the misclassified events, you can dive deeper into these data to understand why the model failed. Perhaps they were incorrectly classified in the first place (inaccurate training data) or the misclassification could be due to natural variablity in the call characteristics. There are any number of possiblities, and by diving into the misclassifications, you can learn a lot about your data and your model. We do not recommend eliminating misclassifications simply because they are misclassifications. The point is to learn more about your data, not to cherry pick your data to get the best performing model. If it is important to only include strong classification results in your final model– then apply the appropriate threshold in the confusionMatrix model, above.

First, identify your misclassified events and save them as an R object and a separate csv file.

Case Predictions You can also save a separate data.frame for your training data that includes the vote distributions. This can be useful for downstream processing and summaries.

```
casePredict <- casePredictions(bant.rf)

misclass <- casePredict %>%
  filter(is.correct== FALSE) %>%
  select(case.id)
```

We can then look closer at these events to learn more about them.

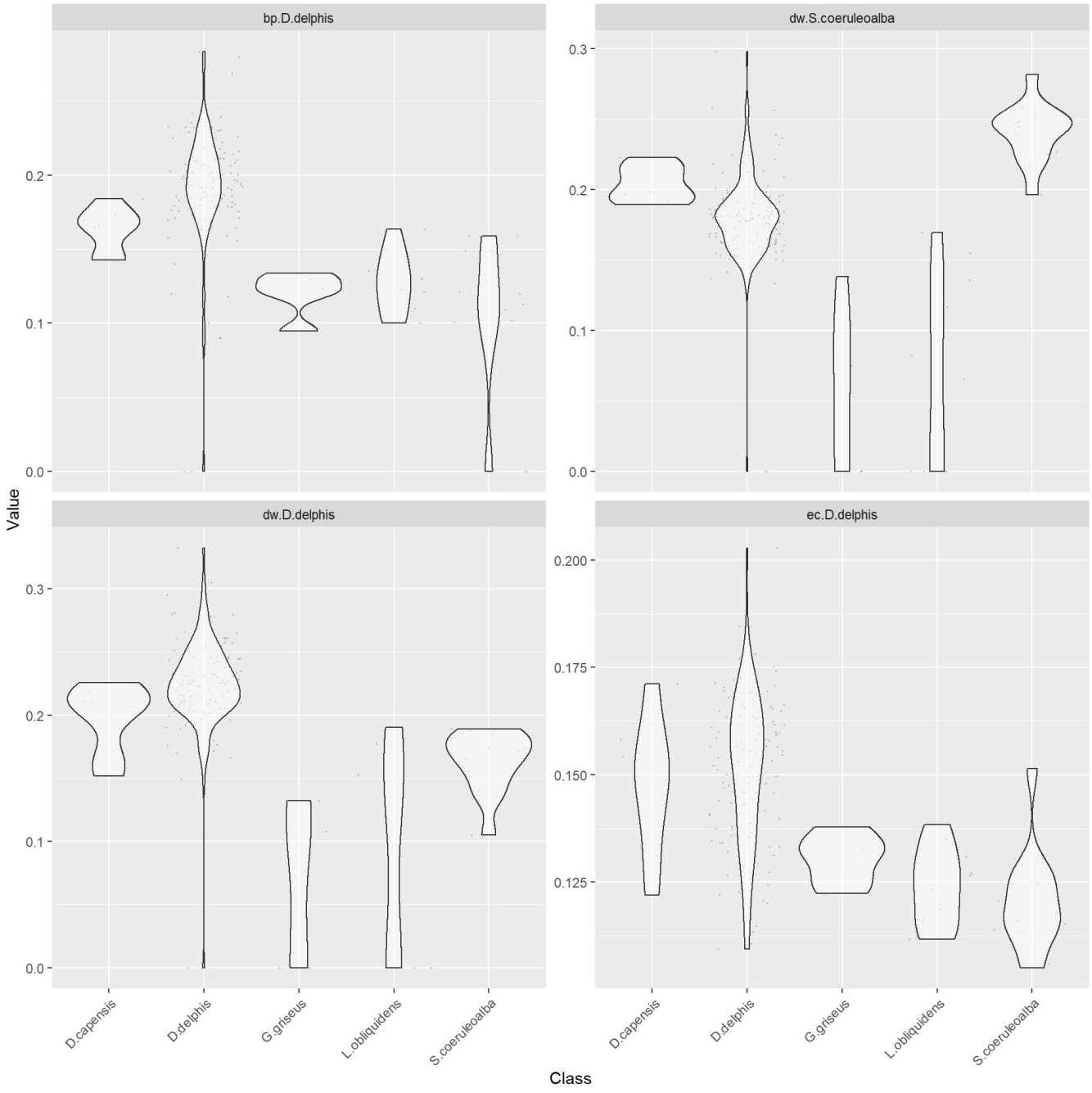
First, identify the most important variables in your event model

```
# Get importance scores and convert to a data frame
bant.imp <-  data.frame(importance(bant.rf))

# Select top 4 important event stage predictors
bant.4imp <- bant.imp[order(bant.imp$MeanDecreaseAccuracy, decreasing = TRUE), ][1:4, ]

# Look at the predictors to identify your next steps
bant.4imp
```

```
                 D.capensis D.delphis G.griseus L.obliquidens S.coeruleoalba
bp.D.delphis       45.905838  96.75902  63.15292      41.27392       70.16595
dw.S.coeruleoalba  54.115891  44.72047  71.54336      63.73765       75.84017
dw.D.delphis        2.039361  85.10194  78.55660      35.97041       48.79433
ec.D.delphis       -3.941655  54.18488  26.83915      38.10189       63.04086
                 MeanDecreaseAccuracy MeanDecreaseGini
bp.D.delphis                101.46769        0.5514270
dw.S.coeruleoalba            91.98777        0.5421580
dw.D.delphis                 90.88226        0.5451790
ec.D.delphis                 72.70526        0.4024866
```

The predictors that showed the greatest importance came from the whistle (dw) detector and the burst pulse (bp) detectors. We can plot the distribution of the predictor variables on these classes (in this case, a violin plot for each of these four most important variables).

```
plotImpVarDist(bant.rf, bantData.df, "species", max.vars = 4)
```



2.5 Predict

The goal of building an acoustic classifier is to ultimately apply this classifier to novel data. It is critical to understand that we should apply our *BANTER* classifier to data collected in the same manner. All variables (detectors, detector measures, event-level variables) must also be the same (with the same labels). For example, novel data collected using a different hydrophone with different sensitivity curves may result in different measurements from your original model (unless the data is calibrated). Even in the case where a classifier is applied to the appropriate data, it is wise to validate a subset of this novel data.

To run a prediction model, you must have your *BANTER* model, and new data. Here we will use the `bant.mdl` object we made previously, and apply it to the `test.data` provided in the *BANTER* package.

Predict The `predict()` function will apply your *BANTER* model to novel data and provide you with a data frame with the events used in the Event Model for predictions, and a data frame of predicted species and assignment probabilities for each event.

```
data(test.data)
predict(bant.mdl, test.data)
```

```
$events
  event.id     species duration    prop.bp   prop.dw   prop.ec groups
1  414_415  D.capensis 27.75000 0.33333333 0.3333333 0.3333333   drop
2      380   D.delphis 29.60000 1.00000000 0.0000000 0.0000000   drop
3      400 F.attenuata 30.98333 0.07936508 0.1269841 0.7936508   drop
  bp.D.capensis bp.D.delphis bp.G.griseus bp.G.macrorhynchus bp.L.obliquidens
1    0.18640000   0.17100000       0.1654             0.1558            0.173
2    0.09333333   0.08333333       0.1400             0.3000            0.190
3    0.14800000   0.10200000       0.1460             0.2360            0.186
  bp.S.coeruleoalba dw.D.capensis dw.D.delphis dw.G.griseus dw.G.macrorhynchus
1         0.1484000       0.19060       0.1922      0.13960             0.1180
2         0.1933333       0.00000       0.0000      0.00000             0.0000
3         0.1820000       0.22375       0.2200      0.03625             0.1725
  dw.L.obliquidens dw.S.coeruleoalba ec.D.capensis ec.D.delphis ec.G.griseus
1           0.1806            0.1790         0.153       0.1486       0.1636
2           0.0000            0.0000         0.000       0.0000       0.0000
3           0.1300            0.2175         0.123       0.1286       0.1360
  ec.G.macrorhynchus ec.L.obliquidens ec.O.orcinus ec.S.coeruleoalba   rate.bp
1             0.1144           0.1308       0.1298            0.1598 1.8018018
2             0.0000           0.0000       0.0000            0.0000 0.1013514
3             0.1320           0.1442       0.1668            0.1694 0.1613771
    rate.dw  rate.ec
1 1.8018018 1.801802
2 0.0000000 0.000000
3 0.2582033 1.613771

$predict.df
  event.id      predicted D.capensis D.delphis G.griseus L.obliquidens
1  414_415     D.capensis    0.44262   0.32158   0.09618       0.05556
2      380      G.griseus    0.11748   0.03958   0.40122       0.28408
3      400 S.coeruleoalba    0.15600   0.09380   0.14436       0.10184
  S.coeruleoalba    original correct
1        0.08406  D.capensis    TRUE
2        0.15764   D.delphis   FALSE
3        0.50400 F.attenuata   FALSE

$detector.freq
  detector num.events
1       bp          3
2       dw          2
3       ec          2

$validation.matrix
             predicted
original      D.capensis G.griseus S.coeruleoalba
  D.capensis           1         0              0
  D.delphis            0         1              0
  F.attenuata          0         0              1
```

3. Discussion

BANTER has been developed in a general manner such that it can be applied to a wide range of acoustic data (biological, anthropogenic). We have encouraged development of additional software (`PAMpal`) to facilitate *BANTER* classification of data analyzed in PAMGuard software. We encourage development of additional open source software to simplify *BANTER* classification of data analyzed using other signal processing software. While this classifier is easy to use, and can be powerful, we highly recommend that users examine their data and their results to ensure the data are appropriately applied. This is especially important when a classifier is applied to novel data for prediction purposes.

Acknowledgements

Many thanks to our original co-authors for their help in developing the original *BANTER* trial. Funding for development of *BANTER* was provided by NOAA's Advanced Sampling Technology Working Group.

References

Rankin, S., Archer, F., Keating, J. L., Oswald, J. N., Oswald, M., Curtis, A. and Barlow, J. (2017) Acoustic classification of dolphins in the California Current using whistles, echolocation clicks, and burst pulses. Mar Mam Sci, 33: 520-540.doi:10.1111/mms.12381 (doi:10.1111/mms.12381)