rfPermute: Estimation of predictor importance significance in Random Forest models

Frederick I. Archer

Abstract

One of the strengths of the ensemble machine learning algorithm, Random Forest, is its ability to estimate the relative importance of predictors within classification and regression models. However, there is no method for identifying which predictors are significantly more important than what would be expected by random chance. I present the *rfPermute* package which is a wrapper for the commonly used *randomForest* package that produces permutation-based estimates of predictor importance significance. The package also includes utility functions for summarizing and visualizing Random Forest model results.

Introduction

Since its inception, the ensemble machine learning algorithm, Random Forest (Breiman 2001), has been rapidly gaining popularity as a powerful tool for classification and uncovering patterns in complex data sets. Because it is non-parametric and produces internally-validated classification models, it has been used in a wide variety of fields (D. R. Cutler et al. 2007; Touw et al. 2012; Winham et al. 2012), and has proven to be robust in performance tests against other commonly used modelling algorithms (Berk 2006).

The most common implementation of Random Forest is the *randomForest* package (Liaw and Wiener 2002). This package allows easy specification of both classification and regression models and provides several useful functions for summarizing their results. One of the metrics that many users of Random Forest are interested in examining are the measures of variable or predictor importance. The importance of a predictor is estimated by measuring the decrease in prediction accuracy when that predictor is permuted in each tree in the forest. These values are often used to identify and rank those predictors that are most related to the response and critical for classification. More detail about how the various importance metrics are computed can be found in Breiman (2002), Liaw and Wiener (2002), as well as the help file for the `importance` function in *randomForest*.
However, *randomForest* only produces raw measures of predictor importance and it is up to the user to identify cutoff values above which predictors would be considered to have a significant impact on model predictions. In order to address this, I have implemented a permutation test called *rfPermute* which generates a null distribution of importance scores for each predictor against which the observed importance scores are compared. This null distribution is created by randomly permuting the response variable (class assignments in a classification model, or independent continuous response in a regression model) among cases, running the same Random Forest model on the permuted data, and storing the resulting importance scores.

Under this procedure, a predictor that is not adding any significant information to the model will have an observed importance score that is similar to those generated by a random shuffling of the response, while a significant predictor will have an importance score much larger than the null. Significance p-values are then calculated as the fraction of replicates in the null distribution that are greater than or equal to the observed value. *rfPermute* provides both the p-values and the null distribution which the user can use for variable selection and further exploration.

Execution

The function, `rfPermute`, provides a wrapper for the `randomForest` function that accepts all of its arguments, plus two additional ones: `nrep`, which is an integer specifying the number of permutation replicates to run, and `num.cores`, which is an integer specifying the number of CPU cores to distribute the permutations over on a multi-core system. If `nrep` is set to 0, then the return value is a list with the same elements as the equivalent call to `randomForest`. With one or more permutation replicates specified, the returned list contains two elements: `null.dist`, which is a list containing arrays of the null distributions for unscaled and scaled importance measures, and `pval`, which is an array containing the permutation p-values for unscaled and scaled importance measures.

As an example, we use `rfPermute` to create a classifier of four clades of the endosymbiont dinoflagellate *Symbiodinium* from a data set of metabolite profiles as published in Klueter et al (2015). Significance is measured with 1000 permutations. Because each *Symbiodinium* type is represented by 16 cases in this data set, we create a model where 8 cases from each type are used in each tree, and sampling is done without replacement. This scheme ensures that in each tree, half of the samples from each class are used for trainng, while the other half are retained as out-of-bag (OOB) for testing. `rfPermute` automatically sets `importance = TRUE` to force `randomForest` to calculate and return the matrix of importance scores, so we don't need to set that. However, we will be using the proximity matrix later, so we must specify `proximity = TRUE`.

```r
# Load the package
library(rfPermute)
```

```r
# Load the metabolomics data set
data(symb.metab)

# Create the randomForest model with 1000 permutations.
metab.rf <- rfPermute(type ~ ., data = symb.metab, sampsize = rep(8, 4), replace = FALSE,
    proximity = TRUE, nrep = 1000)
metab.rf
```

```

Call:
 randomForest(formula = type ~ ., data = symb.metab, sampsize = rep(8,      4), replace = FALSE, proxim
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 12

        OOB estimate of  error rate: 4.69%
Confusion matrix:
     A194 B184 B224 D206 class.error
A194   15    1    0    0      0.0625
B184    0   16    0    0      0.0000
B224    0    0   14    2      0.1250
D206    0    0    0   16      0.0000
```

The result is a `rfPermute` object that inherits from and has the same components as a `randomForest` object, plus `null.dist` and `pval`, as seen below.

```r
class(metab.rf)
```

```
[1] "rfPermute"            "randomForest.formula" "randomForest"
```

```r
names(metab.rf)
```

```
 [1] "call"            "type"            "predicted"
 [4] "err.rate"        "confusion"       "votes"
 [7] "oob.times"       "classes"         "importance"
[10] "importanceSD"    "localImportance" "proximity"
[13] "ntree"           "mtry"            "forest"
[16] "y"               "test"            "inbag"
[19] "null.dist"       "pval"            "terms"
```

```r
str(metab.rf$null.dist)
```

```
List of 2
 $ unscaled: num [1:155, 1:6, 1:1000] 0.00175 0.00025 0.00025 0.0005 -0.0005 -0.00175 -0.001 -0.0005 0.0
  ..- attr(*, "dimnames")=List of 3
  .. ..$ : chr [1:155] "Amino.Acid.2" "Amino.Acid.3" "Alanine" "Amino.Acid.4" ...
  .. ..$ : chr [1:6] "A194" "B184" "B224" "D206" ...
  .. ..$ : NULL
 $ scaled  : num [1:155, 1:6, 1:1000] 1.531 0.578 0.218 1.001 -0.447 ...
  ..- attr(*, "dimnames")=List of 3
  .. ..$ : chr [1:155] "Amino.Acid.2" "Amino.Acid.3" "Alanine" "Amino.Acid.4" ...
  .. ..$ : chr [1:6] "A194" "B184" "B224" "D206" ...
  .. ..$ : NULL
```

```r
str(metab.rf$pval)
```

```
 num [1:155, 1:6, 1:2] 0.827 0.232 0.734 0.321 0.569 ...
 - attr(*, "dimnames")=List of 3
  ..$ : chr [1:155] "Amino.Acid.2" "Amino.Acid.3" "Alanine" "Amino.Acid.4" ...
  ..$ : chr [1:6] "A194" "B184" "B224" "D206" ...
  ..$ : chr [1:2] "unscaled" "scaled"
```

Predictor significance

The permutation p-values for predictors in the `rfPermute` object can be accessed using the `rp.importance` function, which behaves in the same manner as `randomForest::importance`, returning a matrix of the scaled or unscaled importance scores along with their respective p-values.

```r
imp <- rp.importance(metab.rf)
head(imp)
```

```
                      A194   A194.pval     B184   B184.pval     B224
C30..5.Sterol     7.607126 0.000999001 9.512588 0.000999001 5.825483
C28..5.Sterol    10.405141 0.000999001 6.155306 0.000999001 6.325869
C29.Stanol.2      8.135186 0.000999001 5.115841 0.000999001 6.084142
Sugar.17          5.602500 0.000999001 5.631534 0.000999001 7.717763
Closed.Hexose.5   7.772355 0.000999001 7.303435 0.000999001 2.333828
C28..5.22.Sterol  3.057139 0.005994006 7.166943 0.000999001 4.075839
                   B224.pval      D206   D206.pval MeanDecreaseAccuracy
C30..5.Sterol    0.000999001 10.083884 0.000999001            10.299961
C28..5.Sterol    0.000999001  5.162145 0.000999001            10.190206
C29.Stanol.2     0.000999001  8.979959 0.000999001             9.650500
Sugar.17         0.000999001  6.586413 0.000999001             8.438290
Closed.Hexose.5  0.032967033  2.298043 0.035964036             8.333227
C28..5.22.Sterol 0.002997003  7.536672 0.000999001             8.117613
```

```
                  MeanDecreaseAccuracy.pval MeanDecreaseGini
C30..5.Sterol                   0.000999001        1.3762887
C28..5.Sterol                   0.000999001        1.3390682
C29.Stanol.2                    0.000999001        1.2566522
Sugar.17                        0.000999001        1.0370137
Closed.Hexose.5                 0.000999001        0.8017916
C28..5.22.Sterol                0.000999001        0.8865358
                  MeanDecreaseGini.pval
C30..5.Sterol               0.000999001
C28..5.Sterol               0.000999001
C29.Stanol.2                0.000999001
Sugar.17                    0.000999001
Closed.Hexose.5             0.000999001
C28..5.22.Sterol            0.000999001
```

A density plot of the null distribution of predictor importances along with the observed importance value can be visulized with the `plotNull` function (Fig. 1).

```
plotNull(metab.rf, preds = "Disaccharide.8", imp.type = "MeanDecreaseAccuracy")
```

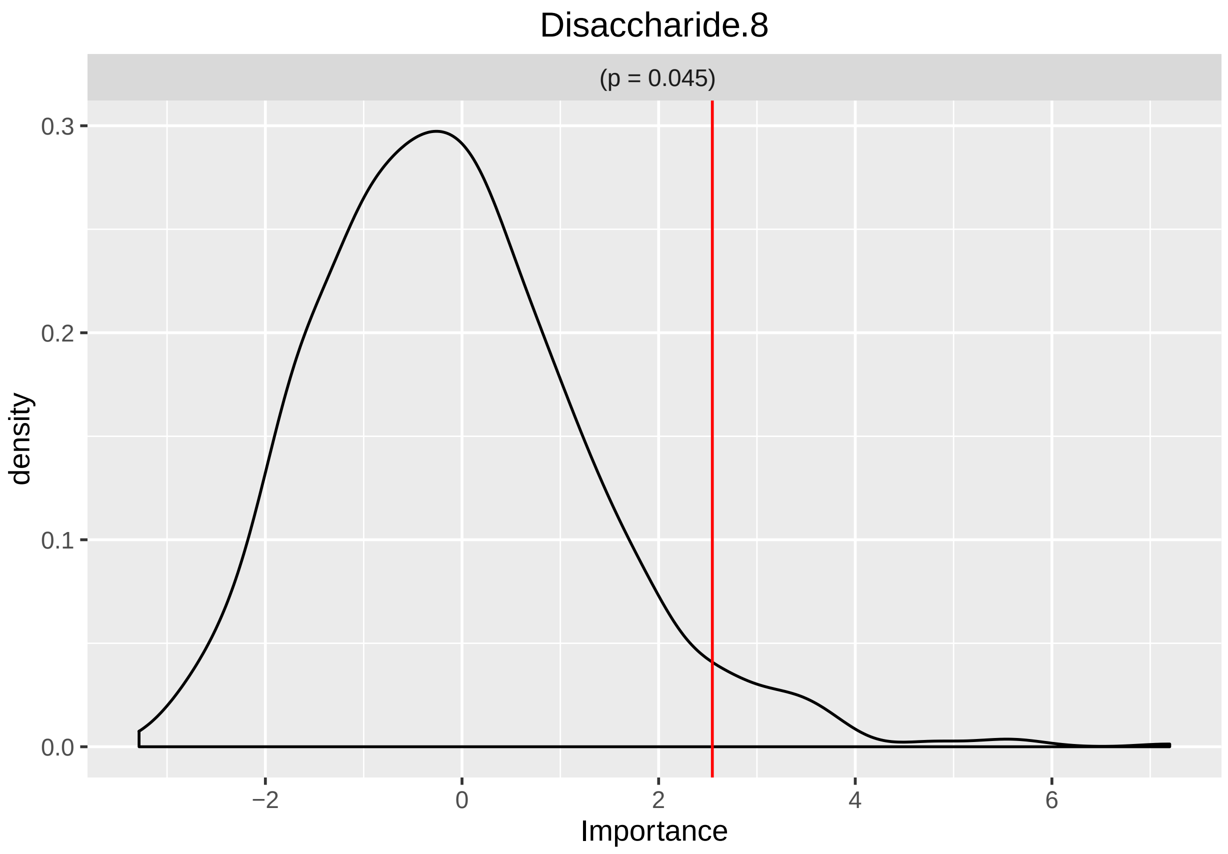

Figure 1: Null distribution of mean decrease in accuracy for the predictor metabolite, 'Disaccharide.8'.

An ordered bar chart of all predictor importances with designations of their significance at a specified critical α (traditionally 0.05), can be visualized with the `plot` function applied to the result of `rp.importance`. Figure 2 shows that approximately the top third of the predictor metabolites are deemed significant at p <=

0.05, even though the distributon of importance scores shows a steady decline with no obvious break in this region.

```r
plot(imp, type = "MeanDecreaseAccuracy")
```

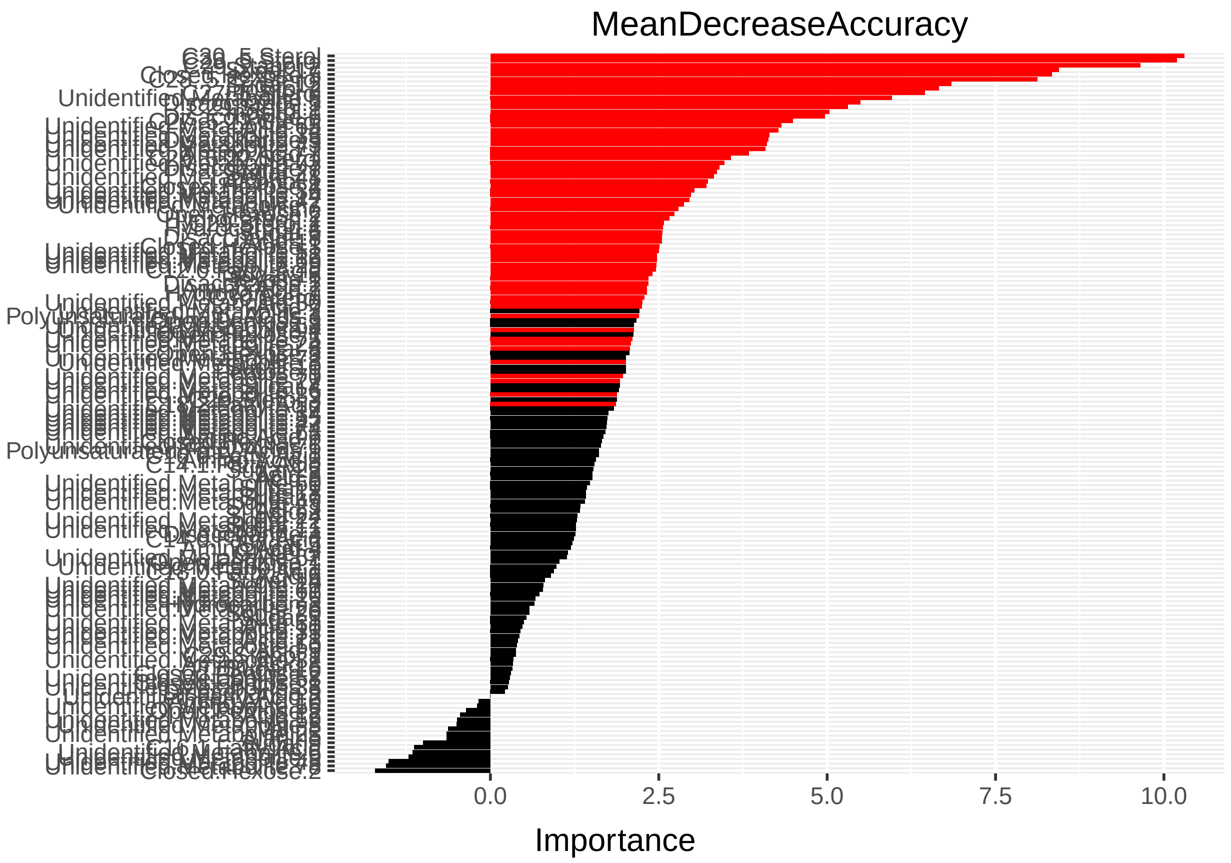

Figure 2: Bar chart of mean decrease in accuracy. Bars colored in red have p-values <= 0.05.

Given that we can't see much detail about specific predictor metabolites, we can also show just the top 10 most important predictors, but this time for all importance types (Fig. 3). The figure can also be restricted to show only significant predictors by specifying `sig.only = TRUE` to the `plot` function.

```r
plot(imp, n = 10)
```

Figure 3 shows that many of the same metabolites are significantly important for all classes. To better visualize the relative distribution of predictor importance across classes we can also produce a heatmap (Fig. 4). Although many of the same metabolites rank high for all classes, there are several lower down in the list that are significant in some classes, but not in others, suggesting that they have strong diagnostic properties for those subsets of classes.

```r
impHeatmap(metab.rf, n = 30)
```

Model summary functions

`rfPermute` also provides a set of convenient summary functions for *randomForest* models, regardless of if the model was run in *randomForest* or *rfPermute*. In order to evaluate how well a classification model is

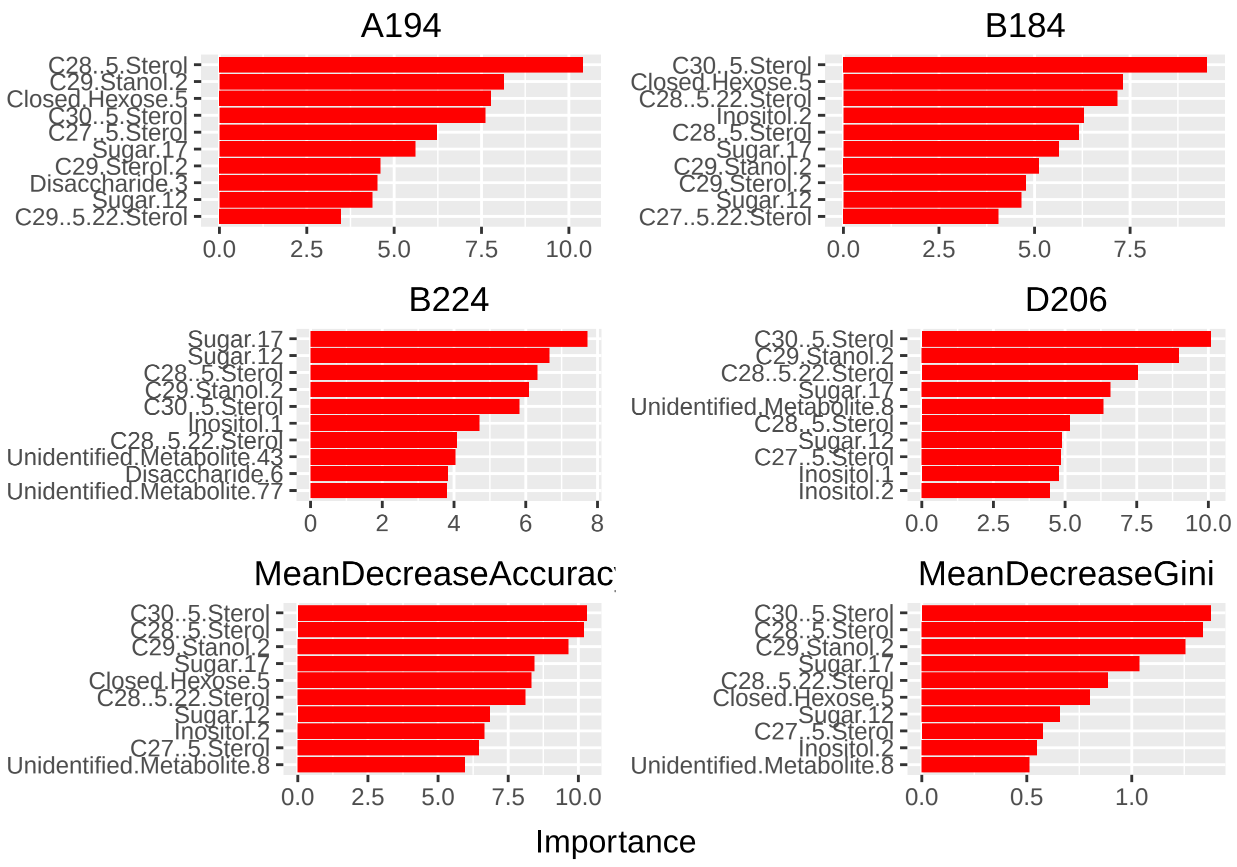

Figure 3: Bar chart of top ten most important predictors.

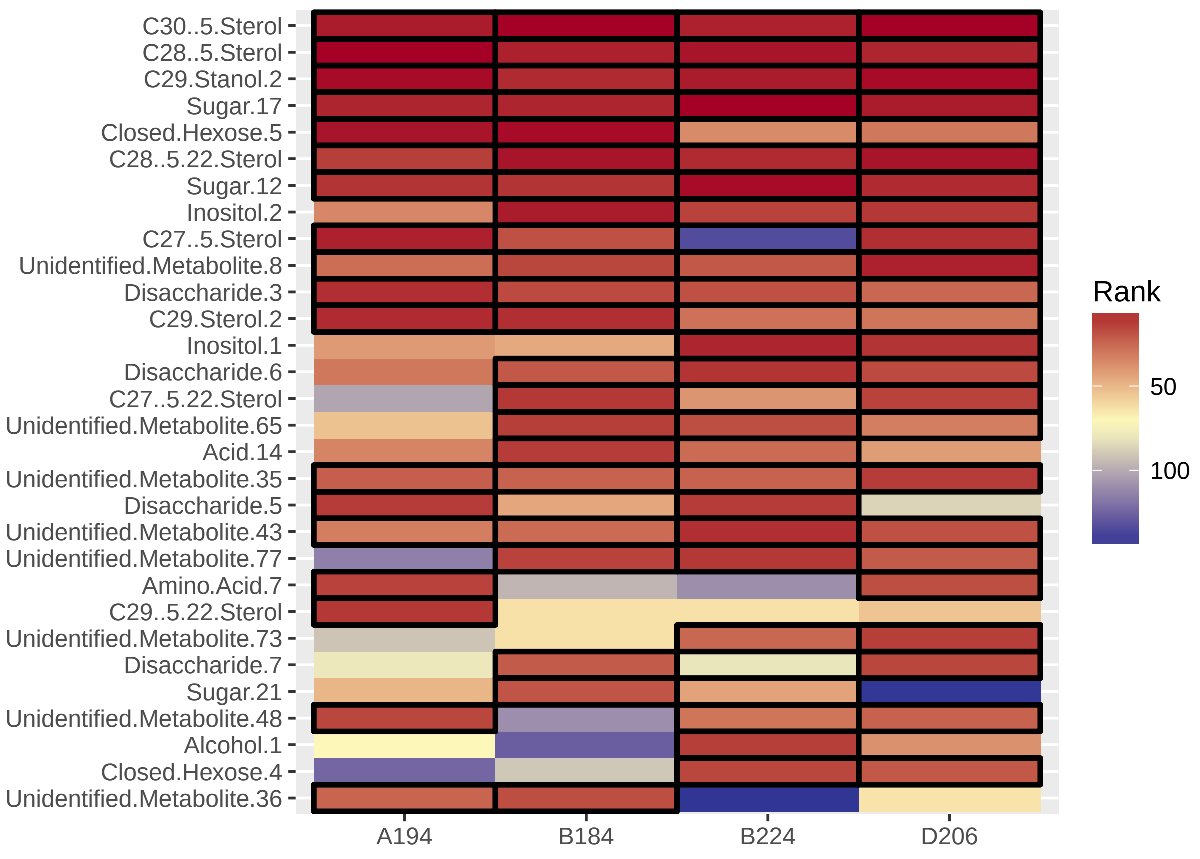

Figure 4: Heatmap of top 30 most important predictor metabolites, color coded by rank order. Cells encircled by black have p-values <= 0.05.

performing, one should compare the observed OOB error rates to what would be expected if samples were just randomly allocated to bins in the absence of information from predictor variables. This rate turns out to simply be the fraction of the total sample size represented by each class. In a simple two-class model, with equal sample sizes in each class, this is then 50% for both classes. However, if the sample sizes are skewed, say 80 in class A, and 10 in class B, then one would expect on average to correctly classify 80 / 90 of the class A samples and 10 / 90 of the class B samples. Thus the expected error rates would be 0.11 for class A and 0.89 for class B. These values, referred to as "priors" are calculated by the `exptdErrRate` function. Since this data set has equal sample sizes, all priors are the same:

```
exptdErrRate(metab.rf)
```

```
 OOB A194 B184 B224 D206
0.75 0.75 0.75 0.75 0.75
```

The `classConfInt` function provides confidence intervals for the class-specific and overall model classification scores based on a binomial distribution. This helps provide a measure of uncertainty of the true classification rate given the observed sample sizes.

```
classConfInt(metab.rf)
```

```
        pct.correct  LCI_0.95  UCI_0.95 Pr.gt_0.8
A194       0.937500 0.6976793 0.9984189 0.9718525
B184       1.000000 0.7940928 1.0000000 1.0000000
B224       0.875000 0.6165238 0.9844864 0.8592625
D206       1.000000 0.7940928 1.0000000 1.0000000
Overall    0.953125 0.8690643 0.9902269 0.9999102
```

The first column in the output is the oberved percent correct (1 - OOB error rate). The next two columns are the bounds of the central 95-percentile of the classification score given the observed sample sizes. The final column provides the fraction of the binomial distribution that is greater than a given threshold. The default is 0.8, but it can be specified using the `threshold` argument.

A complete summary of the *randomForest* classification model can be produced by combining the results of `classConfInt` and `exptdErrRate` along with the full confusion matrix as in the function, `confusionMatrix`.

```
confusionMatrix(metab.rf)
```

```
        A194 B184 B224 D206 pct.correct LCI_0.95  UCI_0.95 Pr.gt_0.8 Prior
A194      15    1    0    0     93.7500 69.76793  99.84189  97.18525    25
B184       0   16    0    0    100.0000 79.40928 100.00000 100.00000    25
B224       0    0   14    2     87.5000 61.65238  98.44864  85.92625    25
D206       0    0    0   16    100.0000 79.40928 100.00000 100.00000    25
Overall   NA   NA   NA   NA     95.3125 86.90643  99.02269  99.99102    25
```

Note that in this matrix, the columns deriving from the `classConfInt` function are multiplied by 100, and `Prior` is 100 * (1 - expected error rate), as I have found that this scale to be more easily communicated in publications.

Vote distributions

When evaluating a Random Forest model, it can be useful to visualize the distribution of votes for classes within the forest. From these distributions, we can tell if individual cases are tending to be correctly classified with large probabilities (have a high fraction of trees voting for the correct class), or have equivocal assignments (vote probabilities are roughly equal). We can also see if misclassified samples were strongly or weakly misclassified. The `plotVotes` function will produce such a distribution, either as a bar chart, which is useful when few samples are present, or as an area chart, which is more readable with many samples.

```
plotVotes(metab.rf)
```

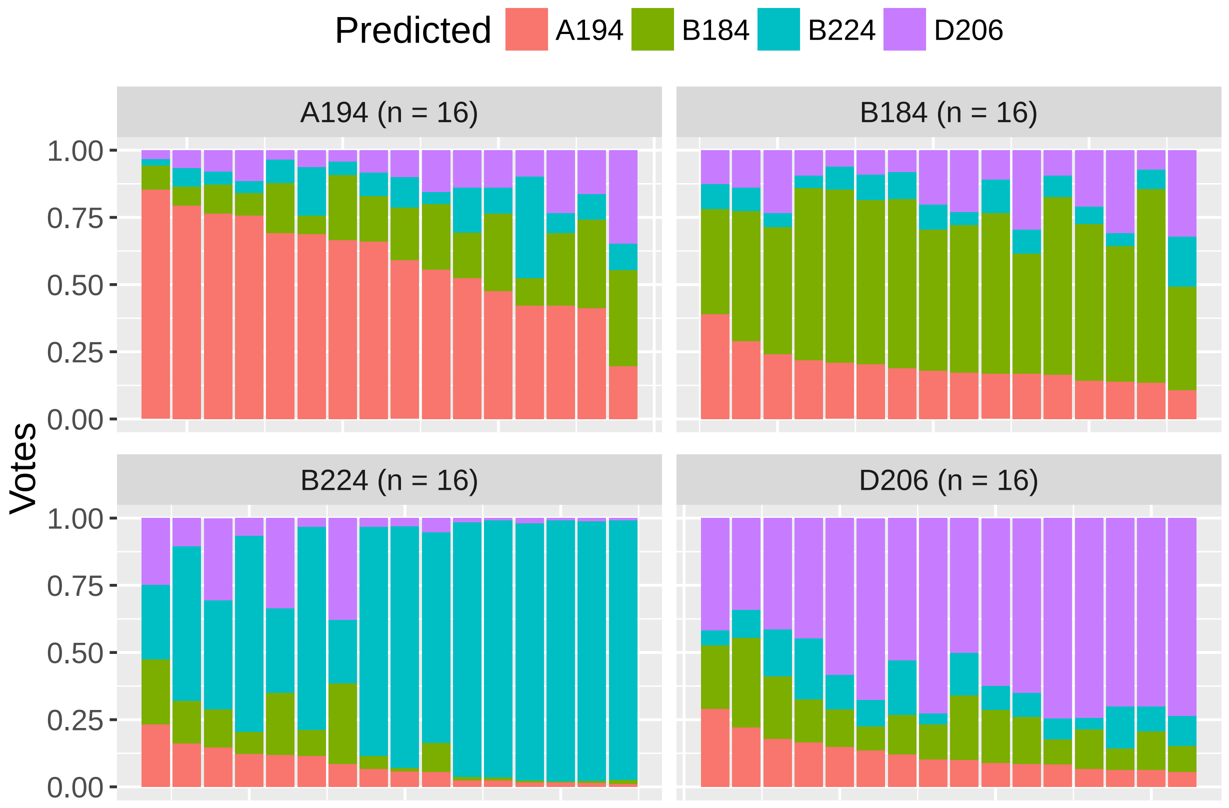

Figure 5: Distribution of votes in the Random Forest for samples of each of the four *Symbiodinium* types.

Figure 5 shows that *Symbiodinium* types A194 and D206 have many of their members classified with a relatively large fraction of votes. We also see that the two B224 samples that were misclassified to D206 only had about 30% of the votes to D206, which is not a strong misclassification.

Proximity plots

For visualizing the relative distribution of samples in the Random Forest, multi-dimensional scaling (MDS) is applied to the proxmity matrix computed by `randomForest`. The *randomForest* package provides the `MDSplot` function which uses *base* graphics to visualize these points. In *rfPermute* there is the `proximityPlot` function which uses *ggplot* graphics (Wickham 2009) and provides a few additional useful features. The result of `proximityPlot` depicts the MDS projection of cases, with classes encircled by a shaded convex hull. Additionally, each case is represented by a color-coded dot and circle. The color of the interior dot corresponds to the original case, while the color of the exterior circle corresponds to the predicted case. Thus, correctly classified cases will have the same color, while misclassified cases will have different colors.

```
proximityPlot(metab.rf)
```

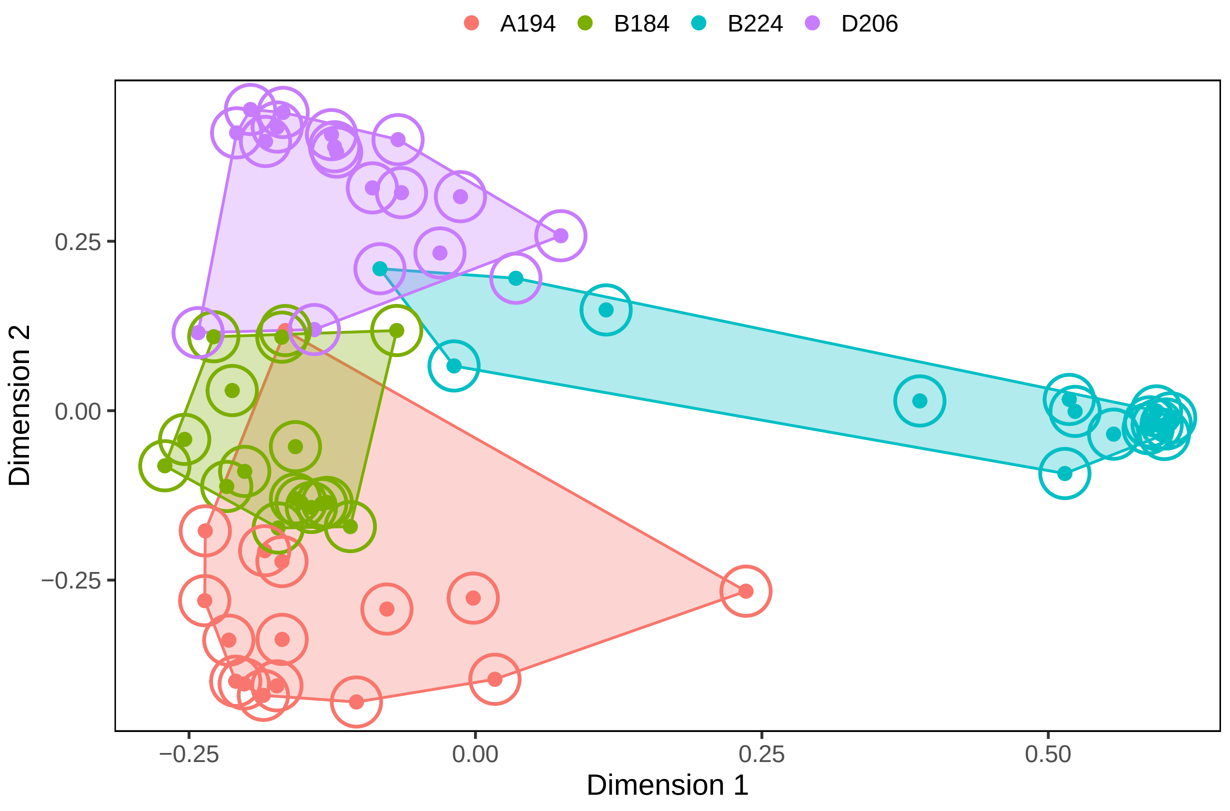

Figure 6: Proximity plot of *Symbiodinium* samples.

In Figure 6, the two misclassified B224 samples are near the periphery of the D206 convex hull, some distance away from the majority of the other B224 samples, but also separated from the majority of the D206 samples. Given the variability in B224, we might then view these as aberrant B224 samples rather than mislabelled D206 samples.

All graphical output in *rfPermute* is produced using the *ggplot* package. Functions that produce figures will also invisibly return the *ggplot* object generated so that they can be further modified if desired. The `proximityPlot` function also returns the MDS matrix resulting from a call to `cmdscale`.

Performance

Because the primary function, `rfPermute`, is a wrapper that calls `randomForest` multiple times, execution time can be lengthy for large data sets and will scale relatively linearly with the number of permutations. However, execution time can be reduced if a multi-core machine is used and the `num.cores` argument set appropriately. For example, on a MacBook Pro with a 2.8GHz Intel Core i7 chip and 16GB of 1600 MHz DDR3 RAM, one `randomForest` run of the *Symbiodinium* data set took approximately 0.1 seconds. The same `rfPermute` model with 100 replicates took 14.4 seconds, and 1000 replicates took 142.3 seconds. When the number of cores used was increased from one to three, the 1000 replicate model took 46 seconds.

Efficiency can also be enhanced by running the same *rfPermute* model can on multiple systems and combining the results later using the `rp.combine` function. This function is the equivalent of `randomForest::combine` and takes a set of the same `rfPermute` models as its arguments.

Installation

The stable version of *rfPermute* can be installed from CRAN via:

```r
install.packages("rfPermute")
```

To install the latest development version from GitHub, use:

```r
# make sure you have Rtools installed
if (!require("devtools")) install.packages("devtools")

# install from GitHub
devtools::install_github("EricArcher/rfPermute")
```

References

Berk, R. 2006. "An Introduction to Ensemble Methods for Data Analysis." Journal Article. *Sociological Methods and Research* 34 (3): 263–95.

Breiman, L. 2001. "Random Forests." Journal Article. *Machine Learning* 45 (1): 5–32.

———. 2002. "Manual on Setting up, Using, and Understanding Random Forests V3.1." Journal Article. http://oz.berkeley.edu/users/breiman/Using_random_forests_V3.1.pdf.

Cutler, D. R., T.C. Jr. Edwards, K. H. Beard, A Cutler, K.T. Hess, J. Gibson, and J.J. Lawler. 2007. "Random Forests for Classification in Ecology." Journal Article. *Ecology* 88 (11): 2783–92.

Liaw, A., and M. Wiener. 2002. "Classification and Regression by RandomForest." Journal Article. *R News* 2/3: 18–22.

Touw, W.G., J.R. Bayjanov, L Overmars, L Backus, J Boekhorst, M Wels, and S. A. F. T. van Hijum. 2012. "Data Mining in the Life Sciences with Random Forest: A Walk in the Park or Lost in the Jungle?" Journal Article. *Briefings in Bioinformatics* 14 (3): 316–26. doi:10.1093/bib/bbs034.

Wickham, Hadley. 2009. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. http://ggplot2.org.

Winham, S.J., C.L. Colby, R.R. Freimuth, X. Wang, M. de Andrade, M. Huebner, and J.M. Biernacka. 2012. "SNP Interaction Detection with Random Forests in High-Dimensional Genetic Data." Journal Article. *BMC Bioinformatics* 13: 164.