rfPermute: Estimation of Predictor Importance Significance in Random Forest Models

by 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 (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 either the decrease in prediction accuracy or decrease in node purity 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. In a classification model, predictor importance can be measured for each class separately, as well as for the model overall. More detail about how these measures 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 dinoflag-

ellate *Symbiodinium* from a data set of metabolite profiles as published in Klueter et al. (2015). We build a Random Forest model with 2000 trees. Significance of the predictor importances 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 full 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`.

```
# Load the package
library(rfPermute)

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

# Create the randomForest model with 1000 permutations.
metab.rf <- rfPermute(type ~ ., data = symb.metab, ntree = 2000, sampsize = rep(8,
    4), replace = FALSE, proximity = TRUE, nrep = 1000)
metab.rf


Call:
 randomForest(formula = type ~ ., data = symb.metab, ntree = 2000,      sampsize = rep(8, 4),
 replace = FALSE, proximity = TRUE)
               Type of random forest: classification
                     Number of trees: 2000
No. of variables tried at each split: 12

        OOB estimate of  error rate: 3.12%
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   15    1      0.0625
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.

```
class(metab.rf)

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

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"
```

The `null.dist` element is a list with arrays of the unscaled and scaled null distributions. Each array is dimensioned as `[1:i, 1:j, 1:nrep]`, where `i` is the number of predictors, `j` is the number of importance metrics (in a classification model, the number of classes + 2), and `nrep` is the number of permutation replicates. The `pval` element is a three dimensional array of p-values given the null distribution. The array is dimensioned as `[1:i, 1:j, 2]`, where the last dimension specifies `"unscaled"` or `"scaled"` p-values.

Predictor significance

The permutation p-values for predictors in the `rfPermute` object can be accessed with the `rp.importance` function. This function behaves in a similar manner as `randomForest::importance`, returning a matrix of the scaled or unscaled predictor importances with a column of their respective p-values immediately following each importance metric.

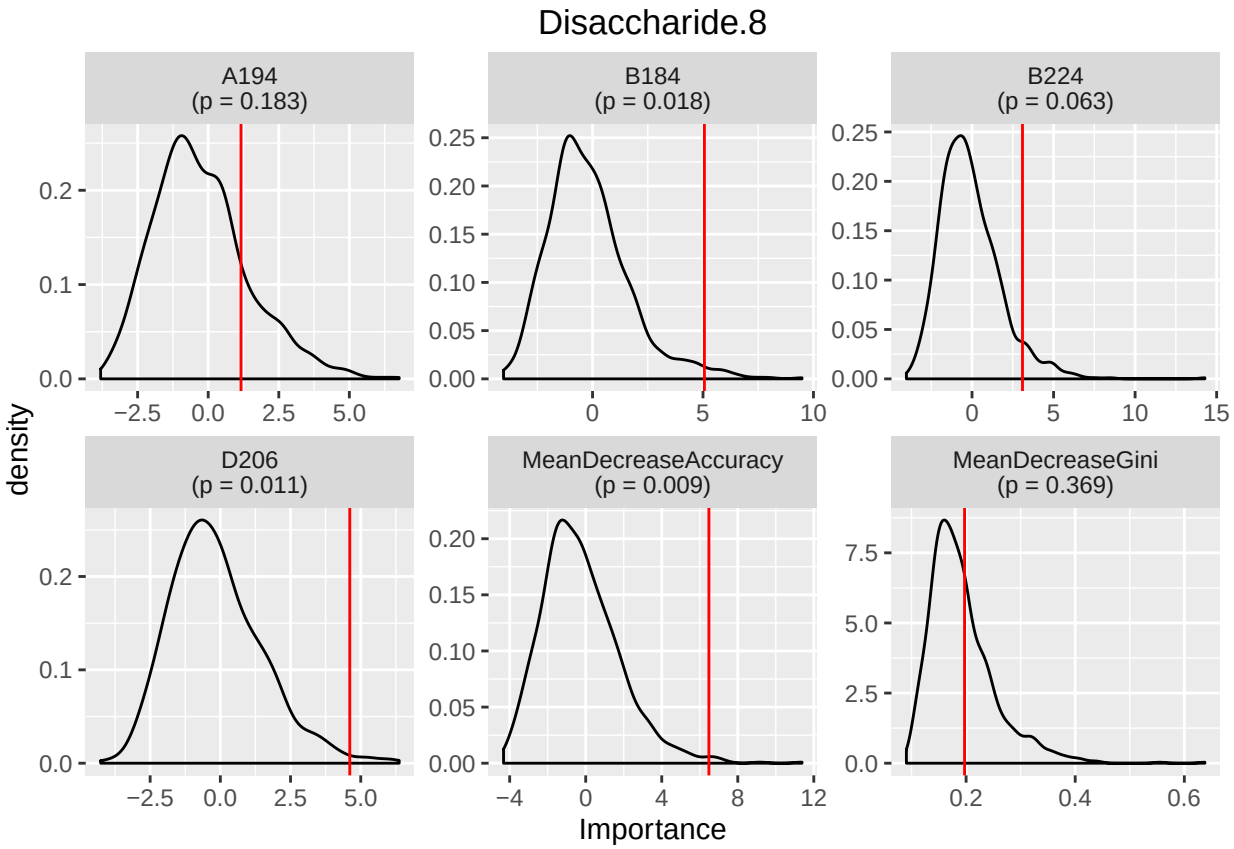

Figure 1: Null distributions of importance scores for Disaccharide.8.

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

```
                 A194 A194.pval B184 B184.pval  B224 B224.pval  D206
C30..5.Sterol   17.54  0.000999 19.3  0.000999 13.29  0.000999 20.60
C28..5.Sterol   19.99  0.000999 13.7  0.000999 11.25  0.001998  9.99
Sugar.17        12.15  0.000999 11.3  0.000999 16.15  0.000999 12.61
C29.Stanol.2    13.01  0.000999 11.3  0.001998  8.54  0.001998 16.97
C28..5.22.Sterol 8.02  0.002997 13.1  0.000999  8.69  0.001998 15.20
Closed.Hexose.5 15.50  0.000999 13.7  0.000999  6.74  0.007992  6.20
                D206.pval MeanDecreaseAccuracy MeanDecreaseAccuracy.pval
C30..5.Sterol    0.000999                 22.2                   0.000999
C28..5.Sterol    0.000999                 19.9                   0.000999
Sugar.17         0.000999                 17.6                   0.000999
C29.Stanol.2     0.000999                 17.4                   0.000999
C28..5.22.Sterol 0.000999                 16.5                   0.000999
Closed.Hexose.5  0.003996                 16.4                   0.000999
                MeanDecreaseGini MeanDecreaseGini.pval
C30..5.Sterol              1.548               0.000999
C28..5.Sterol              1.202               0.000999
Sugar.17                   1.128               0.000999
C29.Stanol.2               1.010               0.000999
C28..5.22.Sterol           0.818               0.000999
Closed.Hexose.5            0.837               0.000999
```

The null distribution along with the observed importance of a predictor can be visualized as either a density plot or histogram with the `plotNull` function. This function acepts vectors of any number of predictors and importance types, so that multiple figures can produced in one call. If no predictors or importance types are specified, all null distributions will be produced by default. In Figure 1, we show the null distributions for the metabolite Disaccharide.8, which is a significant predictor overall in the model with p = 0.009.

It can be useful to visualize the distribution of importance scores across all predictors and easily

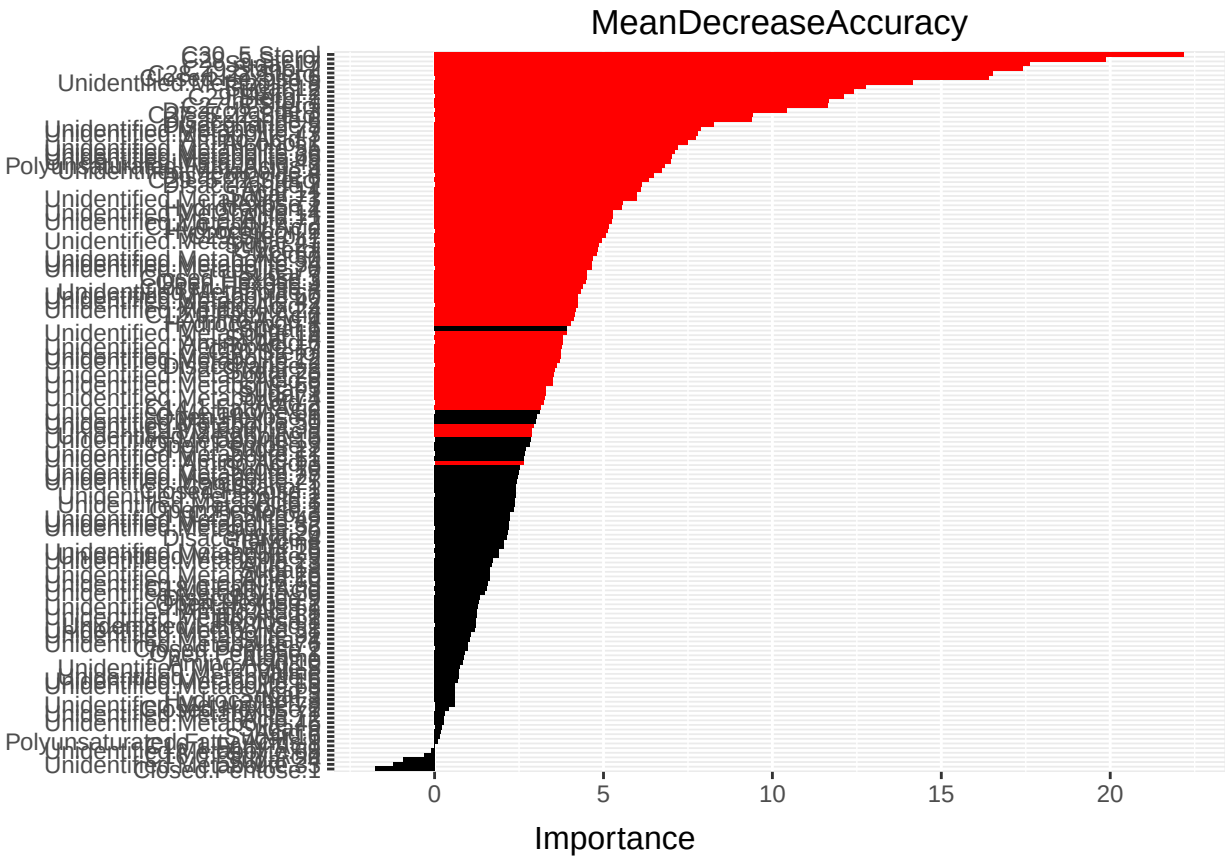

Figure 2: Bar chart of mean decrease in accuracy. Bars colored in red have p-values <= 0.05.

identify those which are significant. In *rfPermute*, this is accomplished with the `plot` function applied to the result of `rp.importance`, which produces an ordered bar chart of all predictor importances with those significant at a specified critical α (traditionally 0.05) identifed in red. Figure 2 shows that approximately the top half 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.

Given that we can't see much detail about specific predictor metabolites in Figure 2 due to overlap of all of the metabolite names, we can also use the `plot` function to just show a subset, say only the top ten most important predictors, but this time for all importance types (Fig. 3). This figure suggests that many of the same metabolites are significantly important for all classes. One can also restrict the output to only significant predictors by specifyig the argument, `sig.only = TRUE`.

To better visualize the relative distribution of predictor importance across classes we can also produce a heatmap as in Figure 4 with the `impHeatmap` function. Although many of the same metabolites rank high for all classes, there are several lower in the list that are significant in some classes, but not in others, suggesting that they might have strong diagnostic properties for those subsets of classes.

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*. One such function is used to evaluate how well a classification model is performing, by calculating the OOB error rates that would expected if samples were just randomly allocated to bins in the absence of information from predictor variables. These rates turn 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)
```

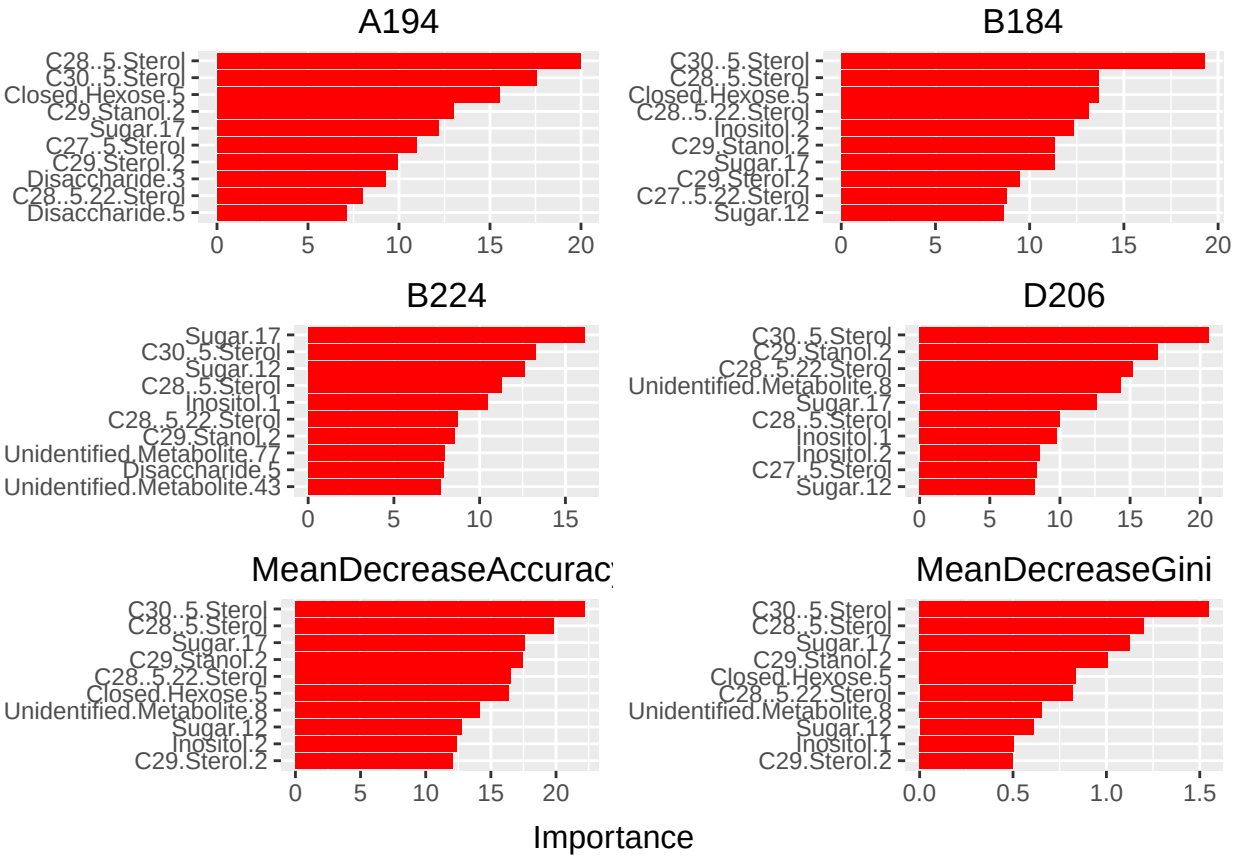

Figure 3: Bar chart of top ten most important predictors.

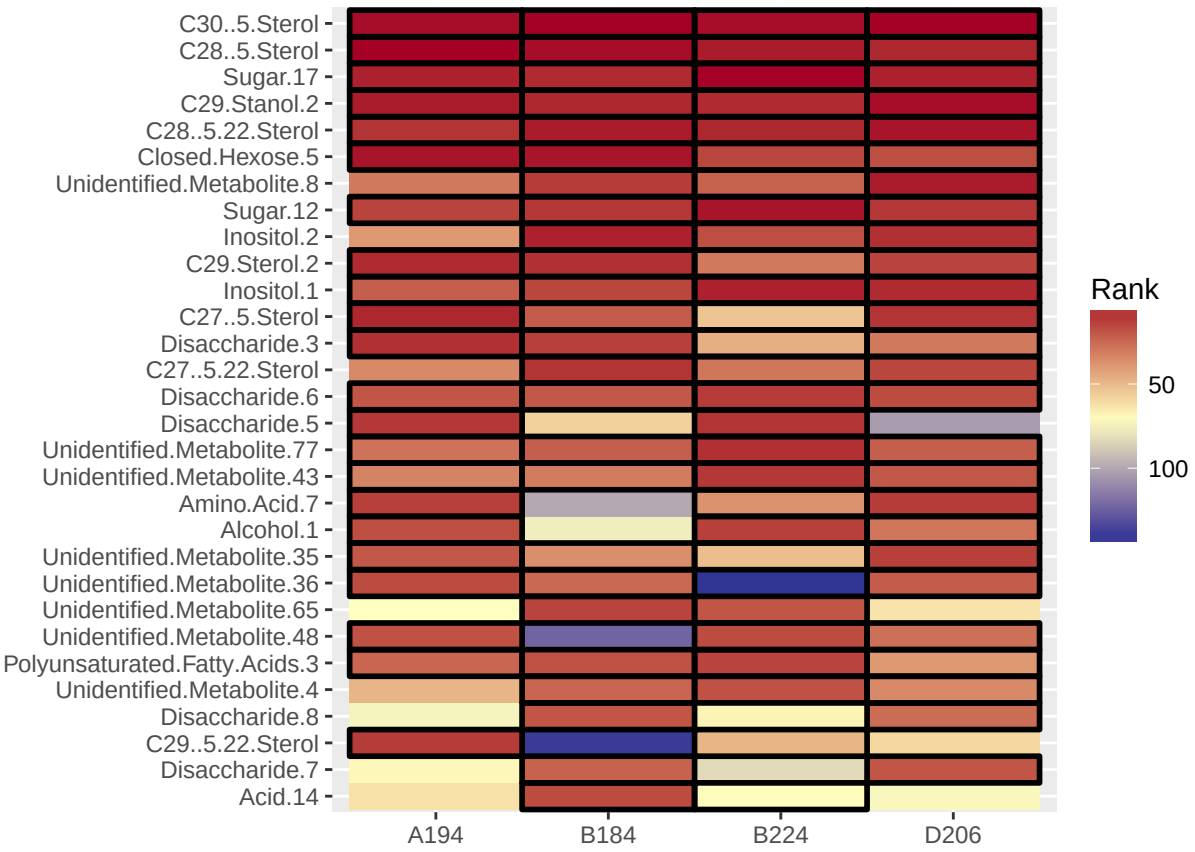

Figure 4: Heatmap of top 30 most important predictor metabolites, color coded by rank order. Cells encircled by black have p-values <=0.05.

```
 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           0.938    0.698    0.998     0.972
B184           1.000    0.794    1.000     1.000
B224           0.938    0.698    0.998     0.972
D206           1.000    0.794    1.000     1.000
Overall        0.969    0.892    0.996     1.000
```

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 to any other value in `0:1` 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        93.8     69.8     99.8      97.2    25
B184        0   16    0    0       100.0     79.4    100.0     100.0    25
B224        0    0   15    1        93.8     69.8     99.8      97.2    25
D206        0    0    0   16       100.0     79.4    100.0     100.0    25
Overall    NA   NA   NA   NA        96.9     89.2     99.6     100.0    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 this scale to be more easily communicated in publications.

Vote distributions

When evaluating a Random Forest classification model, it can also 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 stacked bar chart, which is useful when few samples are present, or as an area chart, which is more readable with many samples.

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 most of the B224 samples have a very large fraction of trees voting for the correct species, but about a third have more votes spread throughout other species, matching the misclassification rate for this species.

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 *ggplot2* 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 be composed of the same color, while misclassified cases will have different colors.

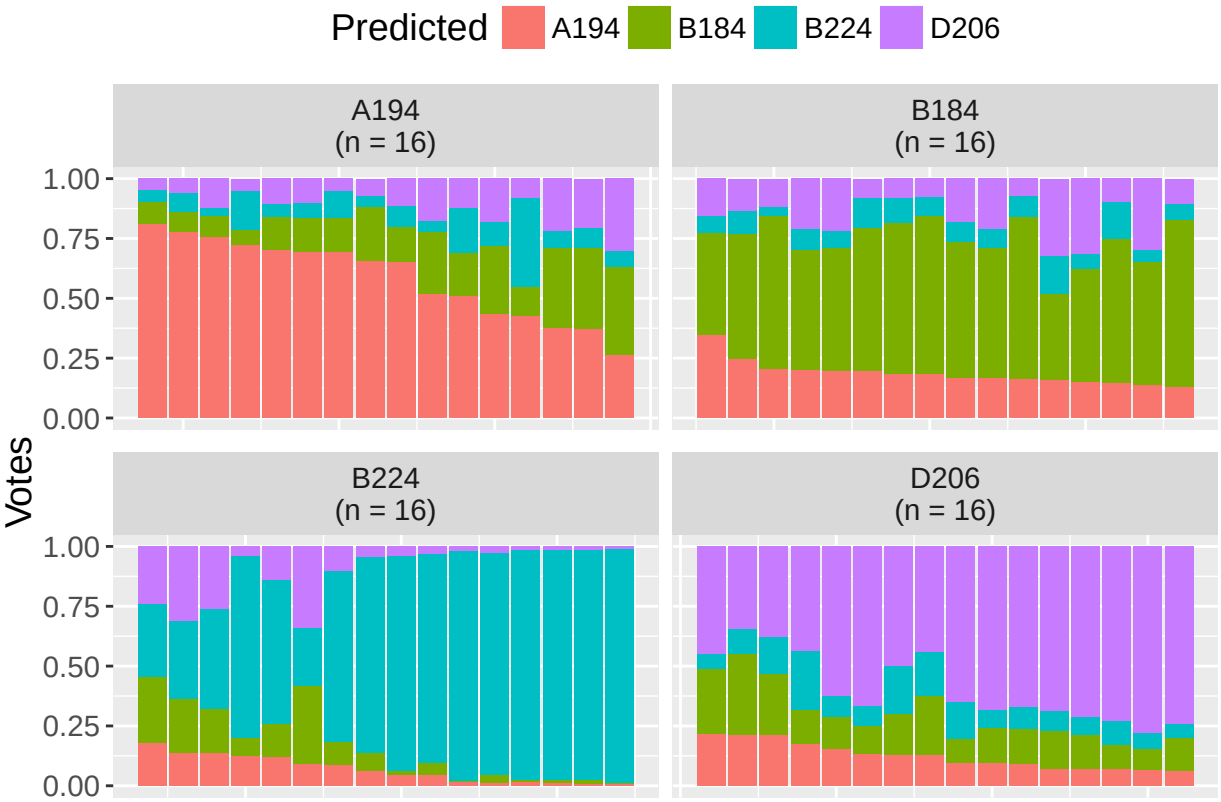

Figure 5: Distribution of votes in the Random Forest model for samples of each of the four *Symbiodinium* types.

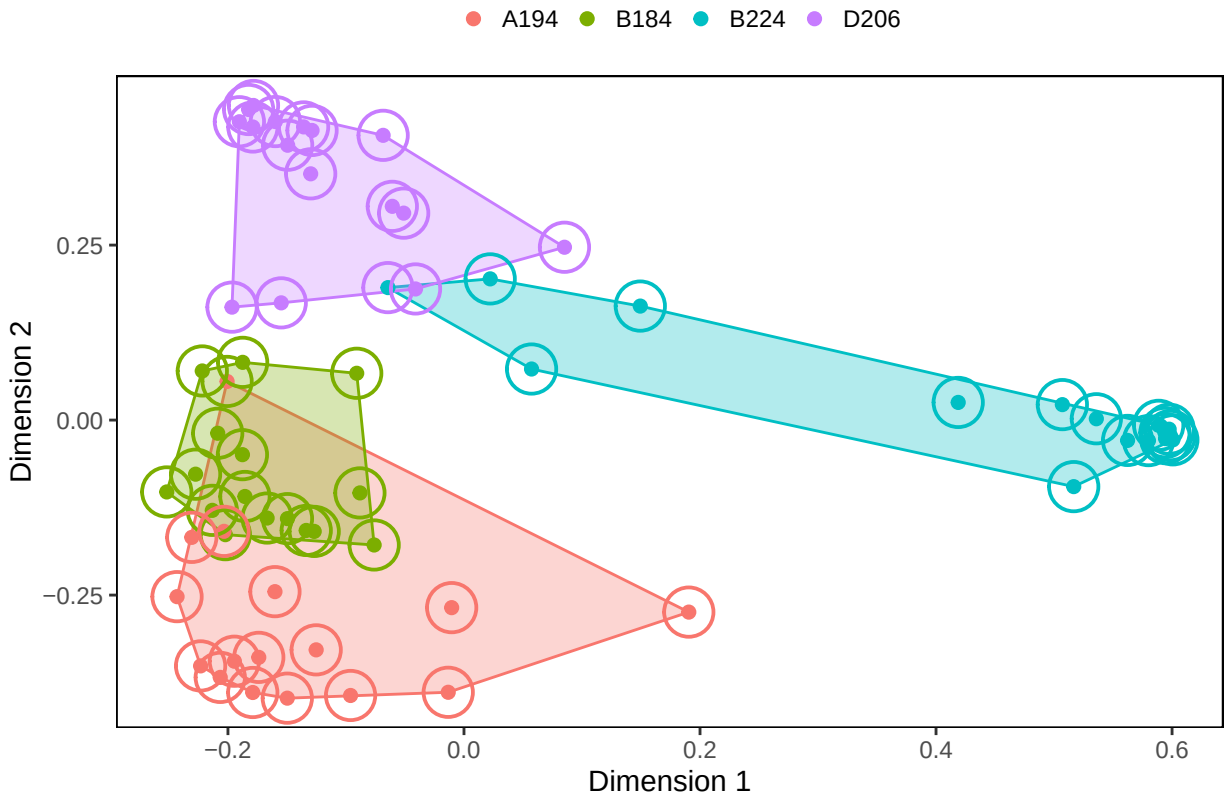

Figure 6: Proximity plot of *Symbiodinium* samples.

In Figure 6, we can see that most of the difference in metabolite profiles is driven by the differences between B224 and the other three species. A194, B184, and D206 are then distributed along the second dimension, with overlap between A194 and D206 on this dimension. However, given that there are no misclassifications between these two species, we suppose that they are differentiated on another dimension. We can also see that the misclassified B224 samples are near the periphery of the B184 and D206 convex hull, some distance away from the majority of the other B224 samples, but also separated from the majority of the B184 and D206 samples. This is reflected in the vote distribution in Figure 5, and given the overall variability in B224, we might then view these as aberrant B224 samples rather than mislabelled samples of another species.

All graphical output in *rfPermute* is produced using the *ggplot2* package. Functions that produce figures will also invisibly return the `ggplot2` 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 0.4 seconds. The same *rfPermute* model with 100 replicates took 52 seconds, and 1000 replicates took 537 seconds (9 minutes) on one core. When the number of cores used was to three, the 1000 replicate model took 193 seconds (3.2 minutes).

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.

Nevertheless, with models of any size, it is advised to first run one model with no replicates to ensure that a sufficient number of trees are being built to produce stable OOB estimates. Stability can be verified using the `plot` function on the model output and observing the trace of OOB estimates for each class as well as overall.

Installation

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

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

To install the latest development version from GitHub, use:

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

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

Bibliography

R. Berk. An introduction to ensemble methods for data analysis. *Sociological Methods and Research*, 34 (3):263–295, 2006. [p1]

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. [p1]

L. Breiman. Manual on setting up, using, and understanding random forests v3.1. 2002. URL http://oz.berkeley.edu/users/breiman/Using_random_forests_V3.1.pdf. [p1]

D. R. Cutler, T. J. Edwards, K. H. Beard, A. Cutler, K. Hess, J. Gibson, and J. Lawler. Random forests for classification in ecology. *Ecology*, 88(11):2783–2792, 2007. [p1]

A. Klueter, J. Crandall, F. I. Archer, M. Teece, and M. Coffroth. Taxonomic and environmental variation of metabolite profiles in marine dinoflagellates of the genus symbiodinium. *Metabolites*, 5:74–99, 2015. [p2]

A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2/3:18–22, 2002. [p1]

W. Touw, J. Bayjanov, L. Overmars, L. Backus, J. Boekhorst, M. Wels, and S. A. F. T. van Hijum. Data mining in the life sciences with random forest: a walk in the park or lost in the jungle? *Briefings in Bioinformatics*, 14(3):316–326, 2012. doi: 10.1093/bib/bbs034. [p1]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009. ISBN 978-0-387-98140-6. URL http://ggplot2.org. [p6]

S. Winham, C. Colby, R. Freimuth, X. Wang, M. de Andrade, M. Huebner, and J. Biernacka. Snp interaction detection with random forests in high-dimensional genetic data. *BMC Bioinformatics*, 13: 164, 2012. [p1]

Frederick I. Archer
Southwest Fisheries Science Center
8901 La Jolla Shores Drive
La Jolla, CA 92037 USA
eric.archer@noaa.gov