**Tree models in R**

Tree models are computationally intensive techniques for recursively partitioning response variables into subsets based on their relationship to one or more (usually many) predictor variables. *Classification* trees involve a categorical response variable and *regression* trees a continuous response variable; the approaches are similar enough to be referred to together as CART (Classification and Regression Trees). Predictors can be any combination of categorical or continuous variables.

All tree-based techniques produce one or more tree objects that represent a series of splits from the 'root' or top of the tree. In one popular implementation (from the *tree* package), each split is based on finding the one predictor variable (and a given threshold of that variable) that results in the greatest change in explained deviance (for Gaussian error, this is equivalent to maximizing the between-group sum of squares (i.e,. minimizing SSE), as in an ANOVA). Tree functions do this using an exhaustive search of all possible threshold values for each predictor. Once a split is made, the routine is repeated for each group separately until all deviance (or some low threshold) is explained or there are too few samples in the remaining subset.

When would you use a CART model rather than a GLM or GAM? The recursive structure of CART models is ideal for uncovering complex dependencies among predictor variables. If the effect of, say, soil moisture content depends strongly on soil texture in nonlinear fashion, CART models of species occurrences have a better shot at detecting this than interaction terms in GLMs or even GAMs. When you have good reason to suspect non-additive interactions among variables, or have many variables and don't know what to expect, try a tree model. If you only have a few variables and generally expect simple linear or curvilinear functions with relatively simple (or no) interactions, tree models will only return approximations of the actual relationship (bumpy as opposed to smooth) using too many parameters. Tree models also have a tendency to overfit (i.e., error is fitted along with the data), and thus lead to over-interpretation. Finally, because they do not involve fitting a smooth function to data, tree model output can be overly sensitive to small changes in the input variables.

There are two common packages for CART models in R: *tree* and *rpart*. I find *tree* easier to use because it is based on our familiar deviance statistics; *rpart* output is more difficult to compare to GLM and GAM alternatives. As we'll see below, these routines can produce very different result for the same dataset, so use carefully!

We'll start with the "tree" package and focus on the distribution of red oak from our tree dataset. Since we'll compare presence-absence models with those based on abundance, we'll construct a dataset that includes absences, like we did for yellow birch last time.

dat = read.csv(file.choose()) #grab the [treedata.csv](http://plantecology.syr.edu/fridley/bio793/treedata.csv) file

dat2 = subset(dat,dat$species=="Quercus rubra") #a red oak subset

ro.plots = unique(as.character(dat2$plotID)) #plots with red oak

u.plots = unique(as.character(dat$plotID)) #all plots

nob.plots = u.plots[is.element(u.plots,ro.plots)==F] #plots without red oak

dat3 = subset(dat,is.element(as.character(dat$plotID),nob.plots)) #dataset of no red oak

dat4 = subset(dat3,duplicated(dat3$plotID)==F) #one row per plot

dat4$cover = 0 #cover of red oak is zero in these plots

rodat = rbind(dat2,dat4) #new dataframe of presences and absences

install.packages("tree")

library(tree)

The first model we'll consider is one using just one predictor variable of elevation to model red oak cover class (including zeros). As long as cover is a numeric variable, tree assumes we desire a regression tree model:

rt1 = tree(cover~elev,data=rodat)

rt1

node), split, n, deviance, yval

\* denotes terminal node

1) root 1039 7405.00 2.3600

2) elev < 1509.5 936 6762.00 2.6030

4) elev < 677.95 286 1055.00 1.7100 \*

5) elev > 677.95 650 5379.00 2.9950 \*

3) elev > 1509.5 103 87.51 0.1553 \*

summary(rt1)

Regression tree:

tree(formula = cover ~ elev, data = rodat)

Number of terminal nodes: 3

Residual mean deviance: 6.295 = 6521 / 1036

Distribution of residuals:

Min. 1st Qu. Median Mean 3rd Qu. Max.

-2.995e+00 -1.995e+00 -1.553e-01 -1.726e-17 2.005e+00 6.845e+00

Output of the fitted model shows the partition structure. The root level (no splits) shows the total number of observations (1039), the associated deviance (at the root equal to the null deviance, or the response variable sum of squares (SSY):

sum(sapply(rodat$cover,function(x)(x-mean(rodat$cover))^2))

[1] 7405.374

followed by the mean response value for that subset (for the root, this is the overall mean). Subsequent splits refer to these statistics for the associated data subsets, with final nodes (leaves) indicated by asterisks. The summary function associated with tree lists the formula, the number of terminal nodes (or leaves), the residual mean deviance (along with the total residual deviance and N-#nodes), and the 5-stat summary of the residuals. The total residual deviance is the residual sum of squares:

sum(sapply(resid(rt1),function(x)(x-mean(resid(rt1)))^2))

[1] 6521.413

To examine the tree, use the plot and text functions:

plot(rt1); text(rt1)

There are three terminal nodes produced by two splits. The first partitioned the data into two sets of above and below 1509.5 m elevation. The second took the latter group and partitioned it again into above and below 678 m elevation. Values at the leaves show the mean cover associated with each of the three final data subsets. The height of each branch is proportional to the reduction in deviance from each split. From this it appears that the tree model has uncovered a hump-shaped relationship: relatively high cover at the lowest elevations, followed by even higher cover between 678 and 1509 m, and then a steep drop to near-absence above 1509 m. We can address this easier by plotting our predictions against our observations:

with(rodat,plot(elev,cover))

x = seq(0,2000)

lines(x,predict(rt1,newdata=list(elev=x)),col="lightblue",lwd=3)

With this graph it is easy to interpret the above model output of associated deviances. The highest elevation set has virtually no variance because these are (almost) all absences: the residual deviance of this terminal node is low (87.51). The other two terminal nodes do a poor job of predicting red oak abundance, with residual deviances of 5379 (middle elevations) and 1055 (lowest elevations). We should expect additional variables to add the most to our model in these regions of elevation.

Why did our tree stop at 3 leaves? There are default values in **tree** that determine the stopping rules associated with few samples or splits that add little additional explained deviance. In this case we had plenty of remaining samples in each node but the added amount of explained deviance achieved with another split did not exceed the built-in threshold of 1% of the null deviance. We can examine this more closely by changing this default to a lower threshold, and then plotting residual deviance against tree size (# nodes) with the **prune.tree** function:

rt2 = tree(cover~elev,data=rodat,control=tree.control(1039,mindev=.003))

plot(prune.tree(rt2))

abline(v=3,col="red")

You can see that the deviance continues to decrease as we add splits (we can do this until each node equals one datum), but with generally decreasing returns.

Given the tendency for tree models to overfit data, how do we know when we have a good model? To use a data-based criterion of when to stop the splitting, tree models use a cross-validation technique that splits the data into 'training' set for model fitting and a validation set to evaluate goodness of fit. The default is a '10-fold' cross-validation, meaning that 10% of the data is left out of each training run. The function **cv.tree** automates the process:

plot(cv.tree(rt2))

What you typically get from this is that the minimum cross-validated deviance occurs with 3 splits. However, note the results of cross-validations are from randomizations; I tried this several times and occasionally got lower deviances for 6 splits. Finally, we can compare the tree model to a simple second-order GLM:

glm1 = glm(cover~elev+I(elev^2),data=rodat,family=poisson)

with(rodat,plot(elev,cover))

x = seq(0,2000)

lines(x,predict(rt1,newdata=list(elev=x)),col="lightblue",lwd=3)

lines(x,predict(glm1,newdata=list(elev=x),type="response"),col="orange",lwd=3)

and compare their pseudo-R2s:

1-(deviance(rt1)/7405)

[1] 0.1193230

1-(glm1$dev/glm1$null)

[1] 0.1070730

In this case the tree model is slightly more predictive, mostly due to its ability to match the zero abundances above 1500 m.

Before we explore the power of tree models with many predictor variables, note the slight difference in output associated with using categorical predictor variables:

rt3 = tree(cover~disturb,data=rodat)

rt3

node), split, n, deviance, yval

\* denotes terminal node

1) root 1039 7405 2.360

2) disturb: SETTLE,VIRGIN 410 2632 1.822 \*

3) disturb: CORPLOG,LT-SEL 629 4577 2.711 \*

plot(rt3);text(rt3)

The technique is the same as with continuous variables, but here the levels of the disturbance factor are put into two exclusive groups that best minimize residual deviance. The output shows the grouping: oddly enough, red oak abundance from both the virgin forests and those highly disturbed from past settlement is more similar than those compared to sites of former selective cutting or corporate logging. Note that categorical with many levels can be a challenge to fit robustly due to the possible number of permutations of the membership of two groups.

Now let's examine a more complicated model of many predictor variables and a categorical variable of presences and absences. By converting the response variable to a factor, we create a classification model:

ct1 = tree(factor(cover>0)~utme+utmn+elev+tci+streamdist+beers,data=rodat)

ct1

node), split, n, deviance, yval, (yprob)

\* denotes terminal node

1) root 1038 1408.000 TRUE ( 0.41426 0.58574 )

2) elev < 1509.5 935 1215.000 TRUE ( 0.35401 0.64599 )

4) utme < 260887 640 740.900 TRUE ( 0.26562 0.73438 )

8) utmn < 3.93797e+006 241 188.800 TRUE ( 0.13278 0.86722 ) \*

9) utmn > 3.93797e+006 399 514.600 TRUE ( 0.34586 0.65414 )

18) utme < 245804 186 257.100 TRUE ( 0.46774 0.53226 ) \*

19) utme > 245804 213 234.500 TRUE ( 0.23944 0.76056 )

38) streamdist < 232.05 113 145.600 TRUE ( 0.34513 0.65487 ) \*

39) streamdist > 232.05 100 73.380 TRUE ( 0.12000 0.88000 ) \*

5) utme > 260887 295 406.500 FALSE ( 0.54576 0.45424 )

10) utme < 307402 263 353.000 FALSE ( 0.60456 0.39544 )

20) utmn < 3.93572e+006 57 63.550 TRUE ( 0.24561 0.75439 )

40) elev < 719 26 36.040 TRUE ( 0.50000 0.50000 ) \*

41) elev > 719 31 8.835 TRUE ( 0.03226 0.96774 ) \*

21) utmn > 3.93572e+006 206 250.300 FALSE ( 0.70388 0.29612 ) \*

11) utme > 307402 32 14.960 TRUE ( 0.06250 0.93750 ) \*

3) elev > 1509.5 103 33.830 FALSE ( 0.96117 0.03883 ) \*

plot(ct1); text(ct1)

Classification tree output is slightly different, in that in addition to information about the threshold of each split, the node sample size, the associated residual deviance, and the predicted value (TRUE=present), you also get the fraction of presences and absences (i.e., the frequency of each response class) for the samples of each node. For example, the root node has presences indicated about 41% of the time, which should be the fraction of presences in the data. The tree diagram shows what we already knew about elevation (over 1509.5 m there are virtually no presences of red oak), and then suggests red oak is slightly more likely to be found further west than UTM Easting 260887 (see also the above output, compare nodes 4 and 5). Let's see whether this tree needs pruning:

plot(cv.tree(ct1))

From this it appears several of the terminal nodes (those beyond 6 or 7) are unwarranted (again, try this several times; my runs are unstable). Our best tree is thus:

btree = prune.tree(ct1,best=7)

btree

node), split, n, deviance, yval, (yprob)

\* denotes terminal node

1) root 1038 1408.00 TRUE ( 0.41426 0.58574 )

2) elev < 1509.5 935 1215.00 TRUE ( 0.35401 0.64599 )

4) utme < 260887 640 740.90 TRUE ( 0.26562 0.73438 )

8) utmn < 3.93797e+006 241 188.80 TRUE ( 0.13278 0.86722 ) \*

9) utmn > 3.93797e+006 399 514.60 TRUE ( 0.34586 0.65414 )

18) utme < 245804 186 257.10 TRUE ( 0.46774 0.53226 ) \*

19) utme > 245804 213 234.50 TRUE ( 0.23944 0.76056 ) \*

5) utme > 260887 295 406.50 FALSE ( 0.54576 0.45424 )

10) utme < 307402 263 353.00 FALSE ( 0.60456 0.39544 )

20) utmn < 3.93572e+006 57 63.55 TRUE ( 0.24561 0.75439 ) \*

21) utmn > 3.93572e+006 206 250.30 FALSE ( 0.70388 0.29612 ) \*

11) utme > 307402 32 14.96 TRUE ( 0.06250 0.93750 ) \*

3) elev > 1509.5 103 33.83 FALSE ( 0.96117 0.03883 ) \*

summary(btree)

Classification tree:

snip.tree(tree = ct1, nodes = c(19, 20))

Variables actually used in tree construction:

[1] "elev" "utme" "utmn"

Number of terminal nodes: 7

Residual mean deviance: 1.012 = 1043 / 1031

Misclassification error rate: 0.2418 = 251 / 1038

We learn that only three variables are used to predict red oak occurrence (elevation, lat, long), that ca. 24% of the predicted values are misclassified in this model, and that the residual deviance is 1043 (pseudo-R2 of about 26%).

*CART with* ***rpart***

You may want to experiment with the other major CART function in R, **rpart** available in the *rpart* library. The input and output to **rpart** are sufficiently different that it's worth going over an example similar to what we produced with the above classification tree. Here's the analogous model with **rpart**:

library(rpart)

rp1 = rpart(factor(cover>0)~utme+utmn+elev+tci+streamdist+beers,dat=rodat,method="class")

The formula specification is the same, but rpart has different fitting routines specified with the method argument; "class" indicates a classification tree, while "anova" indicates a regression tree. There are also options for survival (exponential="exp") data and binomial (2-column) response variables ("poisson"). If y is a factor a classification tree is assumed.

rp1

n= 1039

node), split, n, loss, yval, (yprob)

\* denotes terminal node

1) root 1039 430 TRUE (0.4138595 0.5861405)

2) utme>=260886.5 399 139 FALSE (0.6516291 0.3483709)

4) elev>=1247 202 31 FALSE (0.8465347 0.1534653)

8) utme< 307270 195 25 FALSE (0.8717949 0.1282051) \*

9) utme>=307270 7 1 TRUE (0.1428571 0.8571429) \*

5) elev< 1247 197 89 TRUE (0.4517766 0.5482234)

10) utmn>=3945351 77 25 FALSE (0.6753247 0.3246753) \*

11) utmn< 3945351 120 37 TRUE (0.3083333 0.6916667)

22) utme< 269131 22 5 FALSE (0.7727273 0.2272727) \*

23) utme>=269131 98 20 TRUE (0.2040816 0.7959184) \*

3) utme< 260886.5 640 170 TRUE (0.2656250 0.7343750)

6) utmn>=3937966 399 138 TRUE (0.3458647 0.6541353)

12) utme< 245804 186 87 TRUE (0.4677419 0.5322581)

24) beers< 1.480724 123 56 FALSE (0.5447154 0.4552846)

48) tci< 4.884236 48 12 FALSE (0.7500000 0.2500000) \*

49) tci>=4.884236 75 31 TRUE (0.4133333 0.5866667) \*

25) beers>=1.480724 63 20 TRUE (0.3174603 0.6825397) \*

13) utme>=245804 213 51 TRUE (0.2394366 0.7605634) \*

7) utmn< 3937966 241 32 TRUE (0.1327801 0.8672199) \*

The format of the output table is similar as that for **tree**, but note we have very different results from the **tree** model; our first split here is on the UTM Easting axis rather than elevation. And we have lost information about deviance (SSE); **rpart** uses instead a measure of statistical inequality called the Gini coefficient (default) or a metric based on information theory (parms=list(split="information")). Clearly the choice of splitting depends upon the metric used. The cost of adding another split to the tree is assessed with the 'complexity' parameter (cp). The minimum cp has a default value of 0.01 and can be accessed in a similar way to the 'mindev' parameter of tree. For example:

rp2 = rpart(factor(cover>0)~utme+utmn+elev+tci+streamdist+beers,dat=rodat,method="class", control=rpart.control(cp=0.005))

A call to 'rp2' generates a very large tree. You can plot it will the same plot and text functions:

plot(rp2); text(rp2)

Pruning the tree is accomplished with cross-validation as before, where the relative error rate of 'dropping' the out-of-bag samples through the existing tree is plotted with standard errors for each split:

plotcp(rp2)

Here it looks as though relative error is minimized for a tree of 5 nodes. We return the pruned tree with **prune** and the cp associated with our desired tree size:

printcp(rp2)

Classification tree:

rpart(formula = factor(cover > 0) ~ utme + utmn + elev + tci +

streamdist + beers, data = rodat, method = "class", control = rpart.control(cp = 0.005))

Variables actually used in tree construction:

[1] beers elev streamdist tci utme utmn

Root node error: 430/1039 = 0.41386

n= 1039

CP nsplit rel error xerror xstd

1 0.2813953 0 1.00000 1.00000 0.036920

2 0.0534884 1 0.71860 0.72791 0.034393

3 0.0279070 3 0.61163 0.64186 0.033109

4 0.0139535 4 0.58372 0.61860 0.032716

5 0.0116279 8 0.52791 0.64186 0.033109

6 0.0093023 9 0.51628 0.63023 0.032915

7 0.0081395 12 0.48837 0.62326 0.032796

8 0.0069767 16 0.45581 0.63721 0.033032

9 0.0054264 19 0.43488 0.63721 0.033032

10 0.0050000 24 0.40698 0.63721 0.033032

rp2.pruned = prune(rp2,cp=.02) #cp=.02 is the desired CP threshold in the above table

rp2.pruned

n= 1039

node), split, n, loss, yval, (yprob)

\* denotes terminal node

1) root 1039 430 TRUE (0.4138595 0.5861405)

2) utme>=260886.5 399 139 FALSE (0.6516291 0.3483709)

4) elev>=1247 202 31 FALSE (0.8465347 0.1534653) \*

5) elev< 1247 197 89 TRUE (0.4517766 0.5482234)

10) utmn>=3945351 77 25 FALSE (0.6753247 0.3246753) \*

11) utmn< 3945351 120 37 TRUE (0.3083333 0.6916667)

22) utme< 269131 22 5 FALSE (0.7727273 0.2272727) \*

23) utme>=269131 98 20 TRUE (0.2040816 0.7959184) \*

3) utme< 260886.5 640 170 TRUE (0.2656250 0.7343750) \*

We have a much different result than that produced by **tree**; **rpart** suggests stream distance is not relevant, and claims Easting is more important than elevation. The lack of a residual deviance statistic makes comparing these model fits tricky. However, because this is a classification tree, we can always compare misclassification rates. I'll do this here manually because I couldn't find a simple misclassification rate output for **rpart**:

exp = predict(rp2.pruned,type="class") #predicted values from our pruned rp2 tree

obs = factor(rodat$cover>0) #actual presences of red oak

sum(as.numeric(exp!=obs)) #total misclassifications

[1] 251

Oddly enough, this is identical to the misclassification rate of the ct1 model of 7 terminal nodes fit with **tree**. The take home message is to use these techniques with care, as the same initial group of variables can be used in different ways to produce the same model fit.

**Using multiple trees for enhanced prediction: Bagging Trees and Random Forests**

We noticed that the above cross-validation technique for both standard tree techniques yielded pruned tree results that were not perfectly stable. Bagging Trees (BT) and Random Forests (RF) are two bootstrapping-based techniques that improve the prediction accuracy of tree models by constructing a set of trees by re-sampling from the data and 'averaging' the predictions of each. As with any other model averaging technique, it comes at a cost of model interpretability. For both methods, if there is high tree diversity (splits are based on different variables and/or using different thresholds), the averaged model says little about the underlying relationships between variables.

Let's first examine a BT model for our red oak occurrences. BT is available in the *ipred* library and is built off of the *rpart* implementation.

install.packages("ipred")

library(ipred)

bt1 = bagging(factor(cover>0)~utme+utmn+elev+tci+streamdist+beers,

dat=rodat,nbagg=30,coob=T)

The tree creating function for BT is **bagging**, requiring the standard formula (classification with a factor y variable, regression otherwise) and the number of desired bootstrapped replicate trees (here 30). The function then does what **rpart** does, except it first generates 30 (*nbagg*) new datasets randomly selected (with replacement) from the original data, and creates a separate tree for each. If the model is a classification tree, the model grows the maximum number of leaves; if a regression tree, the size of the tree is controlled by the *rpart.control* arguments. The *coob=T* argument tells R to use the 'out-of-bag' (left aside) values to calculate the misclassification rate (or, if a regression model, the mean squared error). Including this argument means we can ask what the misclassification rate of the 'averaged' model is, which is included as the 'err' component of the fitted model:

bt1$err

[1] 0.2196532

In this case we have a slight improvement of predictive ability with our BT model, as the original **tree** and **rpart** versions gave a 24% misclassification rate. (Your values will differ because this is a stochastic procedure.) We can calculate this ourselves using the **predict** function with our BT object as we did with **rpart**:

exp = predict(bt1,type="class") #predicted values from our pruned rp2 tree

obs = factor(rodat$cover>0)[-31] #actual presences; row 31 excluded due to NA

sum(as.numeric(exp!=obs)) #total misclassifications

[1] 232

We've reduced misclassified values by 19. Unfortunately, BT is only a predictive tool; there is no 'average tree' to examine. For classification models, a single vector of predictions is created based on a 'majority rules' criterion, but this can't be deconstructed into a single 'best guess' tree. You can look at each tree separately using the **summary** function, and you can access trees individually using the $mtrees component of the model. For example, here is the first bootstrapped tree:

bt1$mtrees[1]

It is possible but not convenient to calculate the importance of each predictor in creating the trees; for a regression tree Prasad et al. (2006) tally the total amount of error sum of squares due to each split produced by a given variable and then average this overall all trees. You could do the same here in a classification tree using instead the reported decrease in the Gini coefficient, but to do this for *nbagg* trees is tedious.

Here's the same overall routine with the more convenient Random Forests technique:

install.packages("randomForest")

library(randomForest)

rf1 = randomForest(factor(cover>0)~utme+utmn+elev+tci+streamdist+beers,

dat=rodat,importance=T,na.action=na.exclude)

rf1

Call:

randomForest(formula = factor(cover > 0) ~ utme + utmn + elev + tci + streamdist + beers, data = rodat, importance = T, na.action = na.exclude)

Type of random forest: classification

Number of trees: 500

No. of variables tried at each split: 2

OOB estimate of error rate: 22.16%

Confusion matrix:

FALSE TRUE class.error

FALSE 294 136 0.3162791

TRUE 94 514 0.1546053

Random Forests is similar to Bagged Trees in that many trees (here a forest of 500) are created with bootstrapped samples. With RF, however, the trees are less related to one another because only a subset of predictor variables are available for each split, and these are chosen randomly. This creates a more diverse forest, and many more trees are needed to get a robust result compared to BT. Among many other options, you can control the number of permuted trees (*ntree* argument, default is 500) and number of variables selected randomly at each split (the *mtry* argument; the default is the square-root of the number of total predictors [or 1/3 for a regression tree]).

In our case the misclassification rate is about the same as it was for BT (22%; the 'confusion matrix' says the model does a better job predicting presences than absences), so we haven't gained any predictive power. But there are other benefits of RF, including a convenient way of examining the relative importance of predictor variables in the 'averaged' result (specified in the model formulation with the *importance=T* argument):

rf1$importance

FALSE TRUE MeanDecreaseAccuracy MeanDecreaseGini

utme 0.10450383 0.12314612 0.11534891 125.65908

utmn 0.05470402 0.11071643 0.08748901 92.71591

elev 0.08511044 0.10527131 0.09696550 110.75186

tci 0.01710043 0.02952482 0.02436907 56.03905

streamdist 0.01967501 0.05296555 0.03921863 60.25971

beers 0.01208206 0.02233350 0.01808597 54.09376

The first three columns are in units of percentages: how much does each factor contribute to our ability to predict absences (FALSE), presences (TRUE), or both (MeanDecreaseAccuracy)? RF does this by running the model with and without each factor. Here, Easting is providing about half (11.5%) of our total predictive power, followed by elevation at just under 10%. The Beers index adds less than 2%. It is interesting that models that omit distance-from-stream would have little effect on the rate of false presences but a relatively big effect (>5%) on false absences—that is, more actual presences would be misclassified. If this were a regression table, the importance output would include the difference in mean squared error (MSE) of models with and without each predictor. The final column refers to the Gini index, which is a measure of 'node impurity'—that is, the inverse of how homogeneous data subsets become once a particular variable is used for the split. The higher this number, the more a variable is able to group presences and absences into exclusive groups when it is chosen as the splitting factor. Simple graphs of predictor variable importance are available with:

varImpPlot(rf1)

If you want access to each tree in the forest, you can use the $forest component of the model fit with a subscript for the desired tree (if the *keep.forest* argument is set to TRUE), as above with $mtrees component of **bagging**. Like BT, however, there is no simple way of comparing differences among trees, although **randomForest** does keep a running tally of the misclassification rate as more trees are added to the 'averaged' sample:

plot(rf1)

We have good stability in the misclassification rate after about 70 trees (black line; the green and red lines show accuracies for TRUE and FALSE subsets, resp.). If this plot doesn't show convergence, you need to increase *ntree*. Finally, you can also determine which input values were most difficult to predict accurately by the $votes component, which returns a matrix of each response element and the proportion of times the model determined it was absent (FALSE) or present (TRUE).

A regression tree example of the RF routine shows the slight difference in model output, based on the MSE rather than misclassification rates:

rf2 = randomForest(cover~utme+utmn+elev+tci+streamdist+beers,dat=rodat,

importance=T,na.action=na.exclude)

rf2

Call:

randomForest(formula = cover ~ utme + utmn + elev + tci + streamdist + beers, data = rodat, importance = T, na.action = na.exclude)

Type of random forest: regression

Number of trees: 500

No. of variables tried at each split: 2

Mean of squared residuals: 4.109320

% Var explained: 42.4

varImpPlot(rf2)

The overall MSE is 4.1, and it would increase by more than 60% if we left of elevation. Distance-from-stream is also of greater importance in the regression (abundance) version of the model. Let's compare the BT version:

bt2 = bagging(cover~utme+utmn+elev+tci+streamdist+beers,dat=rodat,nbagg=30,coob=T)

bt2.resid = rodat$cover[-31]-predict(bt2)

SSE = sum(sapply(bt2.resid,function(x)(x-mean(bt2.resid))^2))

SSY = sum(sapply(rodat$cover[-31],function(x)(x-mean(rodat$cover[-31]))^2))

1-SSE/SSY

[1] 0.3226598

The BT variance explained is only 32%, far worse than the equivalent RT model (42.4%). Finally, we can compare the simple regression tree result:

rt = tree(cover~utme+utmn+elev+tci+streamdist+beers,data=rodat)

summary(rt)

Regression tree:

tree(formula = cover ~ utme + utmn + elev + tci + streamdist + beers, data = rodat)

Variables actually used in tree construction:

[1] "utmn" "utme" "elev" "streamdist"

Number of terminal nodes: 13

Residual mean deviance: 4.661 = 4777 / 1025

Distribution of residuals:

Min. 1st Qu. Median Mean 3rd Qu. Max.

-6.067e+00 -1.701e+00 -6.742e-02 7.677e-17 1.133e+00 7.171e+00

The residual sum of squares is 4777, which gives an R2 of ~35% (1-4777/7405), which is similar to the BT R2. RF is the clear winner in this case when it comes to prediction.