Boosted Regression Trees in R

Overview

This is a brief tutorial to accompany a set of functions that we have written to facilitate fitting BRT (boosted regression tree) models in R, and in particular extending Ridgeway's *'gbm'* library to make them easier to apply to typical ecological data, and to enhance interpretation. The functions are as follows:

1. `gbm.step` – fits a gbm model to one or more response variables, using cross-validation to estimate the optimal number of trees. This requires use of the utility functions `roc`, `calibration` and `calc.deviance`.

2. `gbm.fixed`, `gbm.holdout` – alternative functions for fitting gbm models, implementing options provided in Ridgeway's code;

3. `gbm.simplify` – code to perform backwards elimination of variables, to drop those that give no evidence of improving predictive performance

4. `gbm.plot` – plots the partial dependence of the response on one or more predictors;

5. `gbm.plot.fits` – plots the fitted values from a gbm object returned by any of the model fitting options – this can give a more reliable guide to the shape of the fitted surface than can be obtained from the individual functions, particularly when predictor variables are correlated and/or samples are unevenly distributed in environmental space.

6. `gbm.interactions` – tests whether interactions have been detected and modelled, and reports the relative strength of these. Results can be visualised with `gbm.perspec`

7. `gbm.predict.grids` – makes prediction to new data, and can output ascii grids if required.

8. `roc`, `calibration` and `calc.deviance` are utility variables used to calculate various performance statistics, and would not normally need to be called directly.

The tutorial is aimed at helping you to learn the mechanics of how to use the functions and to develop a BRT model in R. It does not explain what a BRT model is - for that, see the references at the end of the tutorial, and for an example application with similar data, see Elith, Leathwick & Hastie's article in *Journal of Animal Ecology*. Please note that we developed this code for our own research, and it simply builds on the base functions provided by Ridgeway. Read his documentation to find out more about that – we show you below how to find that documentation. We have not compiled our code into a formal package, nor written formal help files – but you will be able to learn what is in our functions within this tutorial. Whilst we are willing to answer simple questions and to fix code that doesn't work where possible, we apologize in advance that we will not be able to spend much time in this capacity. Also note that R is regularly updated, and this code and tutorial worked with the current versions in January 2008 (i.e. R 2.6.1 and gbm, version 1.6-3). As with all freeware, people using these functions do so entirely at their own risk.

If you use the code in published research, we would appreciate your acknowledgement by referring to us as its authors.

Jane Elith & John Leathwick

January 2008

Contact details:

Dr Jane Elith
School of Botany
The University of Melbourne
Parkville
Victoria 3010
Australia
Email: j.elith@unimelb.edu.au

Dr John Leathwick
National Institute of Water and Atmospheric Research
PO Box 11115
Hamilton
New Zealand
Email: j.leathwick@niwa.co.nz

Setting up R

We generally work with a different R workspace for each analysis project. The easiest way to set this up is to ask for an R short-cut to be placed on the desktop at the time of installation. This shortcut is then used as a template from which a copy is made and modified for each project. The key steps in making a project-specific shortcut under Windows XP are as follows:

1. select the original R shortcut, copy it by right-clicking / copy, and then right-click / paste in an unoccupied part of the desktop;

2. Right-click this copy and select the “properties” option at the bottom of the menu, which will open a dialogue box allowing modification of the shortcut properties;

3. On the shortcut tab, go to the field labelled "Start in:" and type the name of the working directory in which you want the data to be stored, e.g., "c:\brt";

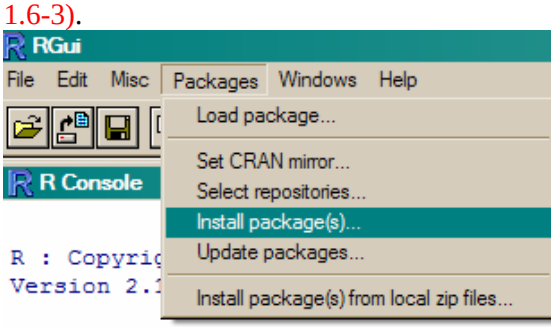
4. Click ok, and the dialogue will disappear, leaving a modified shortcut – double clicking on this will open R with the working directory set to that specified in step 3, and all data and output from the R session will be saved there by default.

Learning about R

In using our functions you will need to understand how to use R, and in particular how to adjust the calls to functions. We don’t cover that here. You could read any of a number of books and tutorials to develop your knowledge - see the list on the R homepage (http://www.r-project.org/) under "Documentation".
As at January 2008 there is also an excellent book draft on the web:
(http://www.zoo.ufl.edu/bolker/emdbook/index.html).
There is also a tutorial describing the use of GLMs and GAMs for habitat modelling in R (http://www.botany.unimelb.edu.au/envisci/brendan/model.html) that may have some useful tips; it accompanies Wintle *et al.* 2005.
In the following sections all R code is in blue, courier new font.

Install the gbm library

The source zip file for the gbm library can either be downloaded from CRAN and saved on the local computer, or loaded into R direct from CRAN. We typically use the second option, and simply select the appropriate option from the packages menu as shown below. Note that there are help files and pdf available for the gbm library; type, for example, `?gbm` for help (once you've loaded the gbm package) and download the pdf from: http://www.cran.r-project.org/doc/packages/gbm.pdf Please note that R is regularly updated, and this code and tutorial worked with the current versions in January 2008(i.e. R 2.6.1 and gbm version 1.6-3).



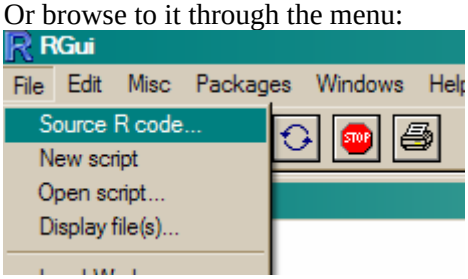
Install our additional BRT functions

These are provided in a file called "brt.functions.R" and can be loaded into R using the `source` command, e.g.,

```
source("brt.functions.R")
```

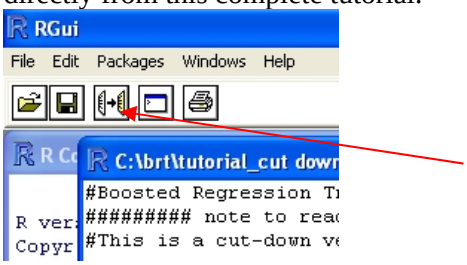
This assumes that the brt functions file is located in the working directory.

Or browse to it through the menu:



Our functions load the gbm library automatically, but if you want to load it yourself use:
`library(gbm)`
or through the menu: `Packages/load package….`

Note: from here on if you prefer to run code from within R, you can use the file:
tutorial_cut down for running.R
Open it within R (File/Open script..)
And when you come on a line you want to run, just highlight the line and send it to R (see red arrow). See R documentation on running scripts if you can't work this out. Alternatively, you can copy and paste directly from this complete tutorial.



Importing and checking example data

Data stored in a text file are typically read in using the `read.table` or `read.csv` command, the help material for which can be accessed, for example, using the following command:

```
?read.csv
```

If reading in the example data used in our examples here, and which is supplied as a comma-delimited file ("model.data.csv") with a header, the appropriate command to read the data from a directory "brt" on your C drive, to R object that we will call "model.data" would be:

```
model.data <- read.csv("c:/brt/model.data.csv")
```

Note that the data had one column with characters in it – using the above syntax, this will have been automatically changed into a "factor" variable, which is how we want to model it. To stop that automatic conversion, look at the help file for use of `as.is`

You can look at the data to check it:
`model.data[1:3,]`

The above is standard R syntax, asking for all columns and the first 3 rows. Alternatively you could have used the function `"head"`

```
> model.data[1:3,]

  Site Angaus SegSumT SegTSeas SegLowFlow DSDist DSMaxSlope USAvgT USRainDays USSlope USNative DSDam   Method LocSed
1    1      0    16.0    -0.10      1.036  50.20       0.57   0.09      2.470     9.8     0.81     0 electric    4.8
2    2      1    18.7     1.51      1.003 132.53       1.15   0.20      1.153     8.3     0.34     0 electric    2.0
3    3      0    18.3     0.37      1.001 107.44       0.57   0.49      0.847     0.4     0.00     0      spo    1.0
```

You can see that by default it prints the row number on the far left. The site number (in this case, simply 1 to 1000) is in column 1. Presence-absence data for *Anguilla australis* (`Angaus`) are in column 2. The environmental variables are in columns 3 to 14. This is the set of data used in all the figures and results in Elith, Leathwick & Hastie (2008) that rely on 1000 sites.

Fitting a model

You need to decide what settings to use – the article associated with this tutorial gives you information on what to use as rules of thumb. These data have 1000 sites, comprising 202 presence records for the short-finned eel (the command `sum(model.data$Angaus)` will give you the total number of presences). As a first guess you could decide:

1. there are enough data to model interactions of reasonable complexity
2. a *lr* of about 0.01 could be a reasonable starting point.

To use our function that steps forward and identifies the optimal number of trees (*nt)* use this call:

```
angaus.tc5.lr01 <- gbm.step(data=model.data,
    gbm.x = 3:14,
    gbm.y = 2,
    family = "bernoulli",
    tree.complexity = 5,
    learning.rate = 0.01,
    bag.fraction = 0.5)
```

(Note that this function is an alternative to the cross-validation one that Ridgeway provides). Here we are using the function `gbm.step` and have passed information to the function about data and settings. We have defined:

- the name of the dataframe containing the data – `data = model.data`;
- the predictor variables – `gbm.x = c(3:14)` – which we do using a vector consisting of the indexes for the data columns containing the predictors (i.e., here the predictors are columns 3 to 14 in model.data);
- the response variable – `gbm.y = 2` – indicating the column number for the species (response) data;
- the nature of the error structure – `family = "bernoulli"` – note that the value for this argument needs to surrounded by quotes. Alternatives are listed in the text at the start of the function – look at this with fix(gbm.step);
- the tree complexity – we are trying a tree complexity of 5 for a start;
- the learning rate – we are trying with 0.01;
- the bag fraction – our default is 0.75; here we are using 0.5.

Everything else – i.e. all the other things that we could change if we wanted to (again, see the header to the file, and Ridgeway's documentation) – are set at their defaults if they are not named in the call. If you want to see what else you could change, you can type `gbm.step` and all the code will write itself to screen, or type `fix(gbm.step)` and it will open in an editor window.

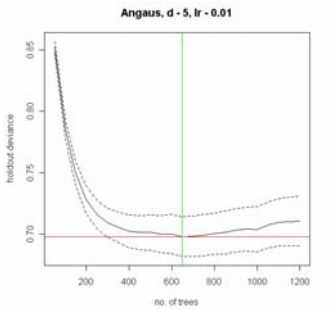
Running a model such as that described above writes progress reports to the screen, makes a graph, and returns an object containing a number of components

Firstly, the things you can see:
The R console will show something like this (not identical, because remember that these models are *stochastic* and therefore slightly different each time you run them, unless you set the `seed` or make them deterministic by using a bag fraction of 1):

```
fitting final gbm model with a fixed number of  650  trees for  Angaus

mean total deviance = 1.006
mean residual deviance = 0.443

estimated cv deviance = 0.698 ; se = 0.016

training data correlation = 0.796
cv correlation =  0.567 ; se = 0.015

training data ROC score = 0.965
cv ROC score = 0.863 ; se = 0.008
```

This reports a brief model summary – all these values are also retained in the model object, so they will be permanently kept (as long as you save the R workspace before quitting).

There will also be a graph like this:



This model was built with the default 10-fold cross-validation (CV) – the solid black curve is the mean, and the dotted curves ± 1 standard error, for the changes in predictive deviance (ie as measured on the excluded folds of the CV). The red line shows the minimum of the mean, and the green line the number of trees at which that occurs. The final model that is returned in the model object is built on the full data set, using the number of trees identified as optimal.

The returned object is a list (see R documentation if you don't know what that is), and the names of the components can be seen by typing:

```
names(angaus.tc5.lr01)
```

Here are some of them; there will be 38 in all:

```
.....
[21] "verbose"             "data"                "Terms"               "cv.folds"
[25] "gbm.call"            "fitted"              "fitted.vars"         "residuals"
[29] "contributions"       "self.statistics"     "cv.statistics"       "weights"
.....
```

To pull out one component of the list, use a number (`angaus.tc5.lr01[[29]]`) or name (`angaus.tc5.lr01$cv.statistics`) – but be careful – some are as big as the dataset – e.g. there will be 1000 fitted values – find this by typing `length(angaus.tc5.lr01$fitted)`

The way we organise our functions is to return exactly what Ridgeway's function returned, *plus* extra things that are relevant to our code. You will see by looking at the final parts of the gbm.step code that we have added components 25 onwards – i.e. from gbm.call on. See Ridgeway's documentation for what his parts comprise. Ours are:

- `gbm.call` - a list containing the details of the original call to gbm.step
- `fitted` – the fitted values from the final tree, on the response scale
- `fitted.vars` – the variance of the fitted values, on the response scale
- `residuals` – the residuals for the fitted values, on the response scale
- `contributions` – the relative importance of the variables, produced from Ridgeway's `summary` function
- `self.statistics` – the relevant set of evaluation statistics, calculated on the fitted values – i.e. this is only interesting in so far as it demonstrates "evaluation" (i.e. fit) on the training data. It should NOT be reported as the model predictive performance.
- `cv.statistics` – these are the most appropriate evaluation statistics. We calculate each statistic within each fold (at the identified optimal number of trees that is calculated on the mean change in predictive deviance over all folds), then present here the mean and standard error of those fold-based statistics.
- `weights` – the weights used in fitting the model (by default, "1" for each observation – i.e. equal weights).
- `trees.fitted` – a record of the number of trees fitted at each step in the stagewise fitting; only relevant for later calculations
- `training.loss.values` – the stagewise changes in deviance on the training data
- `cv.values` – the mean of the CV estimates of predictive deviance, calculated at each step in the stagewise process – this and the next are used in the plot shown above
- `cv.loss.ses` – standard errors in CV estimates of predictive deviance at each step in the stagewise process
- `cv.loss.matrix` – the matrix of values from which cv.values were calculated – as many rows as folds in the CV
- `cv.roc.matrix` – as above, but the values in it are area under the curve estimated on the excluded data, instead of deviance in the cv.loss.matrix.

Choosing the settings

The above was a first guess at settings, using rules of thumb discussed in Elith *et al.* (2008). It made a model with only 650 trees, so our next step would be to reduce the *lr* - e.g., try *lr* = 0.005, to aim for over 1000 trees – i.e.:

```
angaus.tc5.lr005 <- gbm.step(data=model.data,
    gbm.x = 3:14,
    gbm.y = 2,
    family = "bernoulli",
    tree.complexity = 5,
    learning.rate = 0.005,
    bag.fraction = 0.5)
```

- this fit 1600 trees.

To more broadly explore whether other settings perform better, and assuming that these are the only data available, you could either split the data into a training and testing set or use the CV results. You could

systematically alter *tc*, *lr* and the bag fraction and compare the results. See the later section on prediction to find out how to predict to independent data and calculate relevant statistics.

Alternative ways to fit models

The step function above is slower than just fitting one model and finding a minimum. If this is a problem, you could use our `gbm.holdout` code – this includes Ridgeway's functions, but combines them in ways we find useful. We tend to prefer `gbm.step`, especially when modelling many species, because it automatically finds the optimal number of trees.

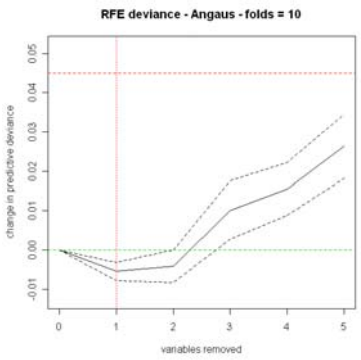
Alternatively, the `gbm.fixed` code allows you to fit a model of a set number of trees; this can be used, as in Elith *et al.* (2008), to predict to new data (see later section).

Simplifying the model

For a discussion of simplification see Appendix 2 of the online supplement to Elith *et al* (2008). Simplification builds many models, so it can be slow. For example, the code below took a few minutes to run on a modern laptop. In it we assess the value in simplifying the model built with a *lr* of 0.005, but only test dropping up to 5 variables (the "n.drop" argument; the default is an automatic rule so it continues until the average change in predictive deviance exceeds its original standard error as calculated in gbm.step).

```
angaus.simp <- gbm.simplify(angaus.tc5.lr005, n.drops = 5)
```

For our run, this estimated that the optimal number of variables to drop was 1; yours could be slightly different:



You can use the number indicated by the red vertical line, or look at the results in the `angaus.simp` object

Now make a model with 1 predictor dropped, by indicating to the gbm.step call the relevant number of predictor(s) from the predictor list in the angaus.simp object – see highlights, below, in which we indicate we want to drop 1 variable by calling the second vector of predictor columns in the pred list, using a `[[1]]`:

```
angaus.tc5.lr005.simp <- gbm.step(model.data, gbm.x =
angaus.simp$pred.list[[1]], gbm.y = 2, tree.complexity = 5, learning.rate =
0.005)
```

This has now made a new model (`angaus.tc5.lr005.simp`) with the same list components as described earlier. We could continue to use it, but given that we don't particularly want a more simple model (our view is that, in a dataset of this size, included variables that contribute little are acceptable), we won't use it further.

Keeping the workspace at a reasonable size

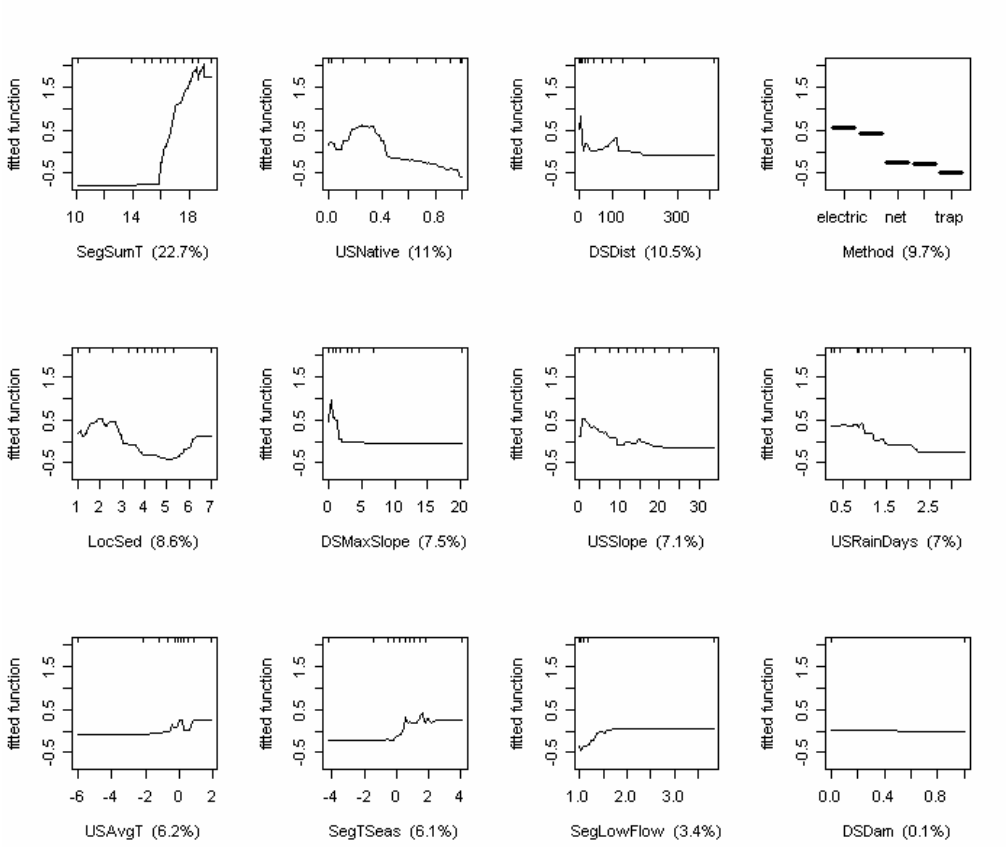
We find that the R workspaces can get very large with BRT models, and that the usual cleanup procedures don't reduce their size. Dumping the models out of the workspace and re-reading them in fixes the problem. If your models all had "angaus" in their name, this code would work:

```
dump(ls(pattern="angaus"), "scratch.R") #this dumps up all objects with "angaus" in the name
source("scratch.R")
save.image("C:/brt/.RData")
```

Plotting the functions and fitted values from the model

The fitted functions from a BRT model created from any of our functions can be plotted using `gbm.plot`. If you want to plot all variables on one sheet first set up a graphics device with the right set-up - here we will make one with 3 rows and 4 columns:

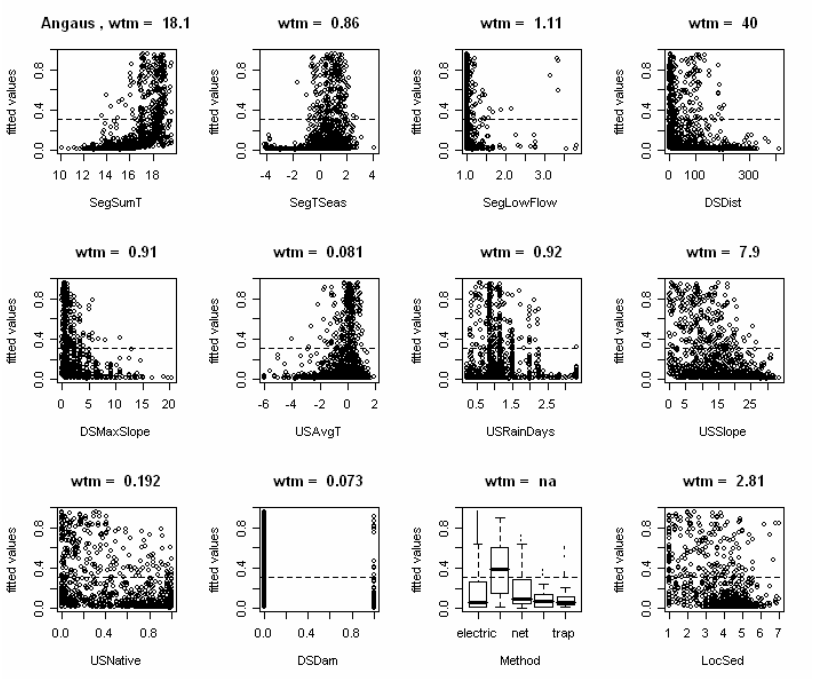
```
par(mfrow=c(3,4))
gbm.plot(angaus.tc5.lr005, n.plots=12, write.title = F)
```



Additional arguments to this function allow for making a smoothed representation of the plot, allowing different vertical scales for each variable, omitting (and formatting) the rugs, and plotting a single variable.

Depending on the distribution of observations within the environmental space, fitted functions can give a misleading indication about the distribution of the fitted values in relation to each predictor. The function `gbm.plot.fits` has been provided to plot the fitted values in relation to each of the predictors used in the model.

```
gbm.plot.fits(angaus.tc5.lr005)
```



This has options that allow for the plotting of all fitted values or of fitted values only for positive observations, or the plotting of fitted values in factor type graphs that are much quicker to print. Values above each graph indicate the weighted mean of fitted values in relation to each non-factor predictor.

Interrogate and plot the interactions

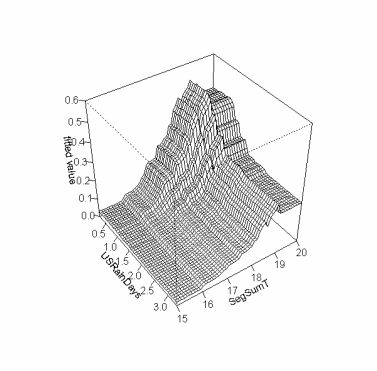
This code assesses the extent to which pairwise interactions exist in the data.

```
find.int <- gbm.interactions(angaus.tc5.lr005)
```

The returned object, here named test.int, is a list. The first 2 components summarise the results, first as a ranked list of the 5 most important pairwise interactions, and the second tabulating all pairwise interactions. The variable index numbers in $rank.list can be used for plotting.

```
> find.int
$rank.list
  var1.index var1.names var2.index var2.names int.size
1          7 USRainDays          1    SegSumT    22.24
2          4     DSDist          1    SegSumT    12.43
3          4     DSDist          2   SegTSeas     8.25
.....

$interactions
           SegSumT SegTSeas SegLowFlow DSDist DSMaxSlope USAvgT USRainDays USSlope
SegSumT          0     5.42       0.05  12.43       0.91   6.20      22.24    6.63
SegTSeas         0     0.00       0.81   8.25       0.95   1.57       1.12    0.11
SegLowFlow       0     0.00       0.00   0.46       0.47   0.90       0.03    1.11
.....
```

You can plot pairwise interactions like this:




```
gbm.perspec(angaus.tc5.lr005,7,1,y.range =
c(15,20), z.range=c(0,0.6))
```

Additional options allow specifications of label names, rotations of the 3D graph and so on.

Predicting to new data

We provide code for making predictions, and this has an option for outputting gridded maps of predictions.

If you want to predict to a set of sites (rather than to a whole map), the general procedure is to set up a data frame with rows for sites and columns for the variables that are in your model. R is case sensitive; the names need to exactly match those in the model. Other columns such as site ids etc can also exist in the dataframe.

Our dataset for predicting to sites is in a file called eval.data.csv – read it in:

```
eval.data <- read.csv("eval.data.csv", as.is=T)
```

The "Method" column needs to be converted to a factor, **with levels matching those in the modelling data**:

```
eval.data$Method <- factor(eval.data$Method, levels =
levels(model.data$Method))
```

To make predictions to sites from the BRT model use Ridgeway's code:

```
library(gbm)
preds <- predict.gbm(angaus.tc5.lr005, eval.data,
n.trees=angaus.tc5.lr005$gbm.call$best.trees, type="response")
```

or use our code with the defaults, which means it just makes predictions to the workspace:

```
gbm.predict.grids(angaus.tc5.lr005, eval.data, want.grids = F, sp.name =
"preds")
```

In both cases the predictions will be in a vector called `preds`, because that's what we named them.

These are evaluation sites, and have observations in column 1 (named `Angaus_obs`). They are independent of the model building set and could be used for an independent evaluation. For example here is code for calculating the deviance and the AUC (area under the ROC curve):

```
calc.deviance(eval.data$Angaus_obs,preds,calc.mean=T)
roc(eval.data$Angaus_obs,preds)
```

Note that the calc.deviance function has different formulae for different distributions of data; the default is binomial, so we didn't specify it in the call

Code for a figure like figure 2, Elith *et al.* (2008)

One useful feature of prediction in *gbm* is you can predict to a varying number of trees. See the highlighted code below to how to predict to a vector of trees. The full set of code here shows how to

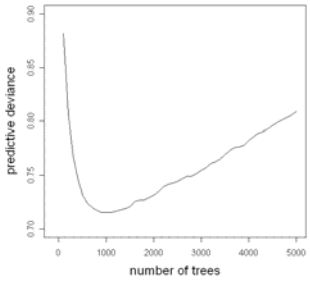
make one of the graphed lines from Fig. 2 in our paper, using a model of 5000 trees developed with `gbm.fixed`

```
angaus.5000 <- gbm.fixed(data=model.data, gbm.x = 3:14, gbm.y = 2,
learning.rate = 0.005, tree.complexity = 5, n.trees = 5000)

tree.list <- seq(100, 5000, by = 100)

pred <- predict.gbm(angaus.5000, eval.data, n.trees = tree.list,
"response")
```

Note that the code above makes a matrix, with each column being the predictions from the model angaus.5000 to the number of trees specified by that element of tree.list – for example, the predictions in column 5 are for `tree.list[5]` = 500 trees.

Now to calculate the deviance of all these results, and plot them:

```
angaus.pred.deviance <- rep(0,50)

for (i in 1:50) {
      angaus.pred.deviance[i] <- calc.deviance(eval.data$Angaus_obs,
      pred[,i],calc.mean=T)
      }

plot(tree.list,angaus.pred.deviance,ylim
=c(0.7,1),xlim = c(-100,5000),type='l',
xlab = "number of trees",ylab =
"predictive deviance", cex.lab = 1.5)
```



Making grids

Alternatively, you can predict to grids (i.e., to a whole map). To do this, export the grids for your variables of interest from a GIS program, as ascii grids (note - these must all have the same cell size and the same number of rows and columns). We provide our data as ascii grids, plus a mask, that has "1" for a river and nodata elsewhere. These are rasterized versions of the line data we usually work with. You can view them by importing them into a GIS program. You can alternatively view just the header by opening the ascii file a text reading program; the first six lines contain information you will need if you want to make a grid of predictions.

Import them into R:

```
grid.names <- c("mask.asc", "segsumt.asc",
"segtseas.asc","seglowflow.asc","dsdist.asc","dsmaxslope.asc","usavgt.asc",
"usraindays.asc","usslope.asc", "usnative.asc", "dsdam.asc", "locsed.asc")

variable.names <- c("mask", names(model.data)[c(3:12,14)]) #here make sure
the order is the same as above, if you're using different data

for(i in 1:length(grid.names)){
assign(variable.names[i],scan(grid.names[i], skip=6, na.string = "-9999"),
pos=1)
}
```

Make them into a data frame, adding a column for the Method – we will predict the probability of catch using electric fishing:

```
preddat <- data.frame
(SegSumT,USNative,DSDist,LocSed,DSMaxSlope,USSlope,USRainDays,USAvgT,SegTSe
as,SegLowFlow,DSDam,rep("electric", length(SegSumT)))

names(preddat)[12] <- "Method"

preddat$Method <- factor(preddat$Method, levels =
levels(model.data$Method))
```

#This data frame has 49000 rows, because there were 49000 cells in each grid. Whilst you could predict to all sites, for very large grids you might want to reduce it to the sites you are interested in:

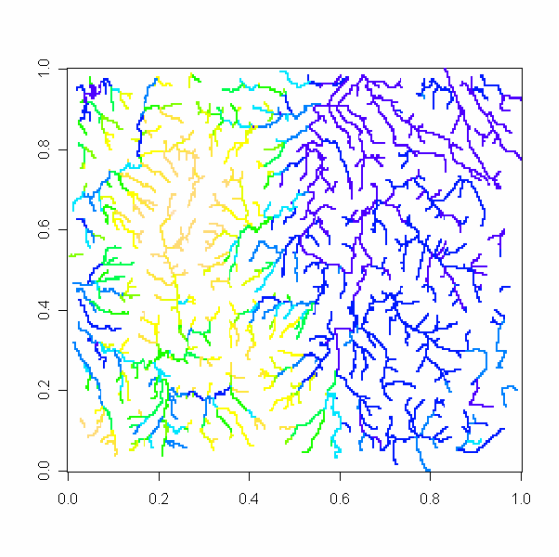
```
preddat<- preddat[!is.na(mask),]
```

You will now have have 8058 rows in the data.

gbm will predict to sites even if there is no data there, so in your own data make sure you mask out those sites with no data, or for which there are insufficient predictors. Our code will re-form these reduced data frames into a full grid, so long as you form the data frame first then reduce it (as above) and give the function a vector the original length of one of the scanned grids. This is how to make a grid (it also returns the predictions to the R workspace too) :

```
gbm.predict.grids(angaus.tc5.lr005, preddat, want.grids = T, sp.name =
"angaus_preds",pred.vec = rep(-9999,49000), filepath = "c:/brt/", num.col =
250, num.row = 196, xll = 0, yll = 0, cell.size = 100, no.data = -9999,
plot=T)
```

The information from the header file is included here (highlighted in aqua), and the `pred.vec` argument is where you give information on the value to be NO DATA (here, -9999) and the number of grid cells in the original grids (here, 49000). It will make a map in R like this:



The highest probabilities are blue and the lowest, tan

It will also have made a file in the directory you specified – here, it will be c:/brt/angaus_preds.csv (see yellow highlights). This could be read into in a GIS program.

Dealing with large grids

If you have very large files you can do the above in a loop. For example, pretending that these ascii files are large and that even with changes to the memory size in R you can't predict, our code can be used in a loop. Let's say we have to do it in 4 repetitions; you would do a first run to make an output with a header file, then do the rest in a loop:

First detail how many rows you will have in each run; here we'll do ¼ at a time

```
rep.rows <- c(49,49,49,49)
```

Then do the first run:

```
for(i in 1:length(grid.names)){
assign(variable.names[i],scan(grid.names[i], skip=6, na.string = "-9999",
nlines = rep.rows[1]), pos=1)
}

preddat <- data.frame
(SegSumT,USNative,DSDist,LocSed,DSMaxSlope,USSlope,USRainDays,USAvgT,SegTSeas,SegLo
wFlow,DSDam,rep("electric", length(SegSumT)))

names(preddat)[12] <- "Method"

preddat$Method <- factor(preddat$Method, levels = levels(model.data$Method))
preddat<- preddat[!is.na(mask),]
```

This version of preddat (above) only has 2116 rows, and before we masked it it was 12250 rows. You can use our code similar to before; it will write the header file, plus the predictions to these 2116 rows. We highlight the new parts of the commands, for clarity

```
gbm.predict.grids(angaus.tc5.lr005, preddat, want.grids = T, sp.name =
"angaus_preds2",pred.vec = rep(-9999,250 * rep.rows[1]), filepath = "c:/brt/",
num.col = 250, num.row = 196, xll = 0, yll = 0, cell.size = 100, no.data = -9999,
plot=T, full.grid = F, part.number = 1, part.row = rep.rows[1])
```

Then do the rest in a loop:

```
for(i in 2:4){
      for(j in 1:length(grid.names)){
      assign(variable.names[j],scan(grid.names[j], skip=(6 +
      sum(rep.rows[1:(i-1)])), na.string = "-9999", nlines = rep.rows[i]),
      pos=1)
      }

preddat <- data.frame
(SegSumT,USNative,DSDist,LocSed,DSMaxSlope,USSlope,USRainDays,USAvgT,SegTSeas,SegLo
wFlow,DSDam,rep("electric", length(SegSumT)))
names(preddat)[12] <- "Method"
preddat$Method <- factor(preddat$Method, levels = levels(model.data$Method))
preddat<- preddat[!is.na(mask),]

gbm.predict.grids(angaus.tc5.lr005, preddat, want.grids = T, sp.name =
"angaus_preds2",pred.vec = rep(-9999,250 * rep.rows[i]), filepath = "c:/brt/",
num.col = 250, full.grid = F, part.number = i, part.row = rep.rows[i], header = F)
}
```

Note that the function is currently also saving the predictions in the R workspace – for large grids you probably want to turn that off; use `pred2R = F` within the call above

Further reading
The following includes a mix of both statistical papers and ecological applications:

Elith, J., Leathwick, J.R., & Hastie, T. (2008). Boosted regression trees - a new technique for modelling ecological data. *Journal of Animal Ecology*

Friedman, J.H. (2001) Greedy function approximation: a gradient boosting machine. *The Annals of Statistics*, **29**, 1189–1232.

Friedman, J.H. (2002) Stochastic gradient boosting. *Computational Statistics and Data Analysis*, **38**, 367-378.

Friedman, J.H., Hastie, T., & Tibshirani, R. (2000) Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, **28**, 337-407.

Friedman, J.H. & Meulman, J.J. (2003) Multiple additive regression trees with application in epidemiology. *Statistics in Medicine*, **22**, 1365-1381.

Hastie, T., R. Tibshirani, and J. H. Friedman. 2001. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer-Verlag, New York.

Leathwick, J.R., Elith, J., Francis, M.P., Hastie, T., & Taylor, P. (2006a) Variation in demersal fish species richness in the oceans surrounding New Zealand: an analysis using boosted regression trees. *Marine Ecology Progress Series*, **321**, 267-281.

Moisen, G.G., Freeman, E.A., Blackard, J.A., Frescino, T.S., Zimmermann, N.E., & Edwards, T.C.. (2006) Predicting tree species presence and basal area in Utah: a comparison of stochastic gradient boosting, generalized additive models, and tree-based methods. *Ecological Modelling*, **199**, 176-187.

Ridgeway, G. (1999) The state of boosting. *Computing Science and Statistics*, **31**, 172-181.

Ridgeway, G. (2006) Generalized boosted regression models. Documentation on the R package "gbm", version 1.5-7. http://www.i-pensieri.com/gregr/gbm.shtml.

Schapire, R. (2003). The boosting approach to machine learning - an overview. In *MSRI Workshop on Nonlinear Estimation and Classification, 2002.* (eds D.D. Denison, M.H. Hansen, C. Holmes, B. Mallick & B. Yu), Springer, NY.

Wintle, B. A., J. Elith, and J. Potts. 2005. Fauna habitat modelling and mapping in an urbanising environment; A case study in the Lower Hunter Central Coast region of NSW. Austral Ecology **30**:729-748.