User's Guide for MN/Model

Phase 4 S-Plus Software

Prepared by

Gary W. Oehlert and Brian Shea

July 2007

Table of Contents

1. Conventions

This manual is intended for use with MN/Model Phase 4 S-Plus functions, dated May 2007. It describes some of the preparation necessary to use the S-Plus functions and the functions themselves. An outline of model evaluation and an example are provided. Conventions are that file names are shown in red, S-Plus output is shown in `green` monospaced font, and S-Plus commands and variable names are shown in `blue` monospaced font.

2. Organization of files

There are several kinds of files.
A. There are MN/Model data files. These are plain text (ASCII) files that are usually created by sampling from the GIS database. A text data file has one column for each variable and one row for each location. There is an additional first row that contains variable names. Variable names may contain periods but should not contain spaces or other punctuation. Data values in the file should be separated by white space, not commas.
MN/Model data files should contain variables named Phase3 and Phase4. These variables code the nature of each location in the data set. For Phase3 the codes are: 0 not used, 1 site centroid, 2 site secondary point, 5 negative survey, 7 site used as survey, 8 random point. For Phase4 the codes are: 0 not used, 1 site centroid, 2 site secondary point, 3 line site, 4 polygon site, 5 negative survey, 6 DOT survey, 7 site used as survey, 8 random point.
MN/Model data files may optionally contain variables named X and Y, which indicate the Easting and Northing of the locations in UTM coordinates. If these variables are present, they will be used to construct the subsets used in spatial cross-validation of the models.
All MN/Model data files are named REGION_DAT.TXT, where REGION is replaced with an abbreviation for the region name, for example, BGWD_DAT.TXT would be Big Woods. The S-Plus functions are expecting the _DAT.TXT to be upper case, but some operating systems are case insensitive.
B. There are files containing S-Plus scripts. At present, there are seven files with names phase4.core.S.txt, phase4.bmalogit.S.txt, phase4.tree.S.txt, phase4.bagging.S.txt, phase4.double.S.txt, phase4.bumping.S.txt, and phase4.naive.S.txt. These are plain text (ASCII) files which contain definitions of S-Plus functions and a table associating informative labels with variable names.

C. There are output/result files. These files are organized into directories named according to the region and a user-specified label. For example, a Phase 4 site model for Big Woods could have its output files placed in the directory BGWD.mod4.site.summaries , while the analogous survey model could use the directory BGWD.mod4.surv.summaries.
This directory will contain text files as well as plot files in pdf format. Some of these files summarize the variables in the data set, and others summarize the results of model fits. The contents and interpretation of the files will be discussed below.
D. There are internal S-Plus files that contain S-Plus variables. You will not ordinarily work directly with these files, but rather access them through S-Plus. S-Plus collects its files/variables into a data directory. Given the number of S-Plus files/variables that will accumulate, you may find it helpful to organize multiple S-Plus data directories.

3. Software Architecture

Phase 4 MN/Model S-Plus functions permit the use of seven different prediction methods: logistic regression with BIC variable selection, logistic regression with Bayesian model averaging, naïve Bayesian classification, tree-structured classification (recursive partitioning), bagging (bagged trees), double bagging, and bumping (bumped trees). See Chapter 3 of "Statistical Methods for MN/Model Phase IV" for more details. Users of the Phase 4 MN/Model S-Plus functions do not interact directly with the different prediction methods. Instead, there are four “front end” functions that users call. Two of these do not depend on the prediction method used, and the other two have an argument that indicates which prediction method to use. Thus, for example, to fit once using logistic regression with BIC model selection and once using bagging, the front end function is called twice, once with the method argument set to `“biclogit”` and once with it set to `“bagging”`.

The file phase4.core.S.txt contains the S-Plus functions that the user interacts with directly. The other S-Plus files contain functions that are used “behind the curtain” and are not called directly by the user. The discussion describes the functions called directly by the user, and not the method-specific functions that are called internally. (The method-specific function files contain comments that document the arguments should their direct use be required for some reason.)

4. Work flow

Before discussing specific S-Plus functions that will be used in MN/Model, it will be helpful to discuss overall work flow.

Modeling begins by selecting a directory in which to work. You may use a separate directory for each region, or you may put multiple regions in a single directory.

You should copy into this directory the S-Plus script files and the MN/Model data file(s). If you wish to use a separate S-Plus data directory for this region, you should create one here from within S-Plus.

Within S-Plus, work follows this pattern.

1. Copy the MN/Model S-Plus script files into the directory where you wish to work, and then read the script files into S-Plus. This need only be done once for each S-Plus data directory.

2. Read the data for the region into S-Plus.

3. Choose a subset of the data to work on (e.g., centroid site models, all site models, survey models, etc).

4. (Optional) Compute summary statistics and descriptive graphics for the variables in the data set and the subset selected.

5. Fit one or more prediction models using the modeling choices available.

6. Summarize the fit of the model to determine its accuracy.

7. Export the prediction model in a form suitable for use in GIS (tree or bagging).

8. (Optional) Make predictions from within S-Plus.

5. Modeling Steps

We now go through the eight modeling steps.

5.1 S-Plus Script Files

Your current directory should contain the files: phase4.core.S.txt, phase4.bmalogit.S.txt, phase4.tree.S.txt, phase4.bagging.S.txt, phase4.double.S.txt, phase4.bumping.S.txt, and phase4.naive.S.txt.

Start S-Plus. At the S-Plus command prompt, enter the following commands:

```
source("phase4.core.S.txt")
source("phase4.tree.S.txt")
source("phase4.bagging.S.txt")
source("phase4.double.S.txt")
source("phase4.bumping.S.txt")
source("phase4.naive.S.txt")
source("phase4.bmalogit.S.txt")
```

These commands read the S-Plus script files into S-Plus itself and make them available for use. You need do this only once for each S-Plus data directory that you use. However, if you change any of the script files, you will need to re-execute the `source()` command to get the revised script into S-Plus.

These six files contain definitions of many S-Plus functions. You will only use a handful of them directly, as most of them are called internally and are not used by you at the command line.

As mentioned above, you need to re-execute the `source()` command when a script file changes. The principal reason why this should happen is that you want to add a new variable description. The file phase4.core.S.txt contains the definition of an object named `mnmodel.var.names.and.labels`. This is a matrix of variable names and longer variable descriptions. If your dataset contains a new variable not in this list, you need to add a new variable name and description. To do this, search in the file phase4.core.S.txt to find the line beginning `mnmodel.var.names.and.labels <- matrix(c(`. Below this you will find name and description pairs like

```
"X",
    "Easting",
```

Simply insert the new name and description into the list, for example

```
"X",
    "Easting",
"newvar",
    "Nice long description",
```

As shown here, the variable name and description should be enclosed in quotes and separated by commas. It is not necessary to have the short name/description pairs entered in any particular order, although you may find it convenient to do so.

5.2 Read Data into S-Plus

Data are read into S-Plus using the function `mnmodel.readdata()`. The required argument for this function is a character string giving the abbreviation for a region, so a typical usage would be

```
mnmodel.readdata("BGWD")
```

("`BGWD`" for Big Woods). The function expects that there is a file in the current directory with the name REGIONABBR_DAT.TXT, where the region abbreviation replaces REGIONABBR in the name --- BGWD_DAT.TXT in the example.

This function also has two optional arguments: `rootvars` and `cmult`. If rootvars is NULL (the default), then a standard set of predictors will be transformed by taking square roots (more below). This can be very helpful for the logistic regression based methods `biclogit` and `bmalogit`, but is of no advantage for tree based methods such as `tree`, `bagging`, and `bumping`. Alternatively, you may specify `rootvars` as a vector of character strings giving the names of variables that you wish to have transformed, for example,

```
mnmodel.readdata("BGWD",rootvars=c("Ded.blk1","Ded.cors"))
```

In this example, only variables `Ded.blk1` and `Ded.cors` would be square rooted. If you wish to take square roots of no variables, set rootvars equal to some nonexistent variable name:

```
mnmodel.readdata("BGWD",rootvars="no.such.variable")
```

If possible, `mnmodel.readdata()` will form spatial clusters for use in spatial cross-validation. If there is a variable named `Clusters` in the data set, that variable will be used to indicate spatial cluster membership. Those clusters will be randomly divided into 10 groups for spatial cross-validation. If `Clusters` is not provided in the data, `mnmodel.readdata()` will construct spatial clusters if `X` and `Y` (the UTM easting and northing) are variables in the data set. The third

parameter `cmult` controls the number of clusters that will be created; there will be `10*cmult` clusters formed, which are then grouped into 10 groups for cross-validation. By default, `cmult` is 40.

The principal purpose of `mnmodel.readdata()` is to create three S-Plus objects (variables): `REGIONABBR.data.all`, `REGIONABBR.subsets`, and `REGIONABBR.transformed.vars`. The first of these is an S-Plus data frame that contains all of the data that we just read in, plus any new variables formed by transformation (see below). The second of these is also an S-Plus data frame, but this frame contains variables that indicate the membership of each location of the data in different data types (see below). The last variable is an S-Plus character vector giving the names of the variables that were transformed (or NULL if no variables were transformed).

In addition to creating the side-effect variables, this function also prints some summary information about the data. For example:

```
> mnmodel.readdata("BGWD4NEW")
Variables in BGWD4NEW.data.all are:
 [1] "Id"        "X"         "Y"         "Abl"       "Alluv"     "Blg"
 [7] "Ht90"      "Rdedblk1"  "Rdedcors"  "Rdedpriv"  "Rdisasbi"  "Rdiscon"
[13] "Rdishdw"   "Rdislkse"  "Rdismin"   "Rdismix"   "Rdisok"    "Rdispibf"
[19] "Rdispr"    "Rdisrb"    "Rdissug"   "Rel90a"    "Rgh90"     "Rlk1size"
[25] "Rlkinout"  "Rlkpinou"  "Rmajarea"  "Rplk1siz"  "Rwtpinou"  "Slp"
[31] "Soilcat"   "Terr"      "Site.type" "Phase3"    "Phase4"


Variables in BGWD4NEW.subsets are:
 [1] "p3.cent"       "p3.sec"        "p3.neg"        "p3.aux"
 [5] "p4.cent"       "p4.sec"        "p4.line"       "p4.poly"
 [9] "p4.surv"       "p4.aux"        "all.rand"      "no.rand"
[13] "all.locations" "all3.sites"    "all4.sites"    "all3.survey"
[17] "all4.survey"   "mod3.cent"     "mod3.site"     "mod3.surv"
[21] "mod4.cent"     "mod4.site"     "mod4.surv"     "CVsets"
[25] "SCVsets"


Variables in BGWD4NEW.transformed.vars are:
NULL


Total number of locations: 7626

    Number of Phase 4 centroid sites: 711
    Number of Phase 4 secondary sites: 111
    Number of Phase 4 line sites: 0
    Number of Phase 4 polygon sites: 40
Total number of Phase 4 sites: 862

    Number of Phase 3 negative surveys: 1273
    Number of Phase 4 DOT surveys: 3707
```

```
    Number of Phase 4 sites as surveys: 169
Total number of Phase 4 non-site survey locations: 5149

Total number of random: 1615

Total number for Phase 4 site-centroid models: 2326
Total number for Phase 4 site models: 2477
Total number for Phase 4 survey models: 7626
```

Subsetting details. The principal subsetting variables available are: `mod3.cent, mod3.site, mod3.surv, mod4.cent, mod4.site, mod4.surv`. These are logical (TRUE/FALSE) variables wherein TRUE indicates that the location is a member of the subset. The "mod" indicates subsets suitable for modeling (that is, they contain both the locations of interest and random locations). The "3" or "4" indicates Phase 3 or Phase 4 archaeological site locations. The suffixes "cent", "site", and "surv" indicate subsets for site centroids, centroids plus site secondary points, or all surveyed places.

Somewhat parallel to these are `all.rand, no.rand, all3.sites, all4.sites, all3.survey`, and `all4.survey`. These subsets indicate the random locations, the non-random locations, and the locations of interest (no random locations) for Phase 3 or 4, all site locations or all survey locations. Finally, `p3.cent, p3.sec, p3.neg, p3.aux, p4.cent, p4.sec, p4.line, p4.poly, p4.surv, and p4.aux` indicate specific site categories: Phase 3 site centroids, secondary points for Phase 3 sites, Phase 3 negative survey points, and Phase 3 sites used as surveys, and Phase 4 site centroids, Phase 4 site secondary points, linear Phase 4 sites, polygon Phase 4 sites, Phase 3 and DOT negative surveys, and Phase 4 sites used as surveys.

In addition to the logical subsetting variables, there are three grouping variables: `Cvsets, SCVsets` and `clusters`. The first two contain integers from 1 through 10 that indicate the groupings that will be used for cross-validation and spatial cross-validation respectively. The last contains integers from 1 through the number of clusters giving cluster membership used during spatial cross-validation. These are not ordinarily accessed by the user.

Transformation details. Some prediction schemes (in particular, those based on logistic regression) may work poorly when the predictor variables are skewed. Many of the landscape-based predictors used in MN/Model are skewed to the right, so `mnmodel.readdata()` automatically tries to make some variables less

skewed by taking their square roots and adding the square-rooted variables to the predictor set. By default, `mnmodel.readdata()` will take the square roots of variables named `D.dra30`, `Ded.blk1`, `Dedbwet1`, `Ded.cors`, `Ded.or30`, `Ded.priv`, `Ded.swm`, `Dint`, `Dis.asbi`, `Dis.br`, `Dis.bw`, `Dis.con`, `Dis.hdw`, `Dislksed`, `Dis.mix`, `Dis.ok`, `Dis.pibf`, `Dis.pr`, `Dis.rb`, `Lkinout`, `Lk1.size`, `Lkpinout`, `Plk1size`, `Riv.conf`, `Wtpinout`, `Dis.pap`, `Dis.sug`, `Dis.maj`, `Dis.min`, `Maj.area`, and `Min.area`.

All newly square-rooted variables are named `sqrt.originalname`, for example, `sqrt.Plk1size` or `sqrt.Dis.pap`.

Some predictors are directions in compass degrees. We assume that a name beginning with `Dir.` is a direction variable; for example, `Dir.ww` is the direction to nearest wetland or water. All direction variables are replaced by their sines and cosines, and the new variables are named as the old variable with `sin.` or `cos.` prepended; for example `sin.Dir.ww` and `cos.Dir.ww` . Some of the prediction techniques could adapt to compass degrees, but most will work better with these transformed variables.

5.3 Subset Choice Modeling is done either to produce a model for surveys or for sites. The standard choices for subsetting variables are `mod3.cent`, `mod3.site`, `mod3.surv`, `mod4.cent`, `mod4.site`, `mod4.surv`. The “3” or “4” indicates Phase 3 or Phase 4 archaeological site and/or survey location data. The suffixes “cent”, “site”, and “surv” indicate subsets for site centroids, all cells occupied by a site, or all surveyed places. Thus, when functions below call for a subset, the user will nearly always use one of these six variables.

Nearly always is not always, and users may from time to time wish to use a subset a bit off of the beaten track. For example, suppose that a user wishes to model DOT survey points (only) against random points. To do this, the user can combine subsetting variables described in the preceding section using the logical “or” operation in S-Plus, which is denoted by the operator `|` (the vertical bar). To form a subset consisting of locations that are either DOT survey points (`p4.surv`) or random points (`all.rand`), the subset can be chosen via `p4.surv | all.rand` . Of course, more than two variables can be OR-ed together to combine more than two groups. For example, `p4.cent | p4.surv | all.rand` would produce a subset consisting

of all locations that are either Phase 4 site centroids, or Phase 4 DOT survey points, or random points. For more complex combinations, S-Plus has a full set of Boolean operations including `!` for logical NOT (inverse) and `&` for logical AND.

5.4 Variable Summaries

After reading data into S-Plus, you may choose to make some simple summaries of the variables. The summarization process looks at the data separately for sites, negative surveys, and random points, so **it is important that you choose a survey-subset for this step**, because only the survey subsets have data from all three types. (Should the subset you choose not include locations of all three types, the function will stop and print an informative message.)

The command to produce these summaries is `mnmodel.var.summaries()`, and it takes three arguments. The first argument is a region abbreviation, such as `"BGWD"`. The second argument is a subsetting variable, such as `mod4.surv` . (A subsetting expression can be used, for example, by OR-ing two or more subsetting variables.) The last argument is a character string label used in constructing the name of the directory where the output should be stored. When using a single subsetting variable, it may be simplest just to label the results with the subsetting variable. Thus a typical usage is

```
mnmodel.var.summaries("BGWD",mod4.surv,"mod4.surv")
```

and a more unusual usage might be

```
mnmodel.var.summaries("BGWD",p4.cent | p4.surv | all.rand,
"centroids.and.DOTneg")
```

The output from this command is stored in a directory named REGIONABBR.label.summaries, where the region argument replaces REGIONABBR and the label argument replaces label. In our examples, we get directories names BGWD.mod4.surv.summaries and BGWD.centroids.and.DOTneg.summaries. If this directory does not exist, the S-Plus function creates it.

`mnmodel.var.summaries()` produces summaries for all data in the region and subset, except that it first eliminates any previously

transformed variables and any variables with names X, Y, Easting, Northing, Site.type, Phase3, Phase4, Id, and Centroid (along with various capitalizations of those names).

The first summary is descriptive statistics. All descriptive statistics are placed in a file named variable.summary.txt in the (previously created) output directory. The output to this file begins with means and standard deviations of the variables, reported separately for sites, negative survey points, and random points. For example, output for the first two variables is:

```
Means and standard deviations of variables:

, , Abl
         Sites Negative Surveys Random Points
mean 943.43155        927.56963     987.37895
  sd  91.55414         98.68696      76.14381

, , Alluv
          Sites Negative Surveys Random Points
mean 0.07656613       0.09263935     0.0247678
  sd 0.26605614       0.28995453     0.1554649
```

After means and variances come two-sample Wilcoxon rank-sum tests for each variable comparing site data to random location data, and negative survey data to random location data. The two-sample Wilcoxon test is a nonparametric test of location (center or median). That is, it asks whether there is evidence that two groups (that are otherwise comparable) are centered at different values, and the test does not rely on the shape of the underlying group distributions. Small p-values indicate that the groups have different centers. For both comparisons, the Wilcoxon statistic and the p-value are reported. For example, output from the first two variables is:

```
Wilcoxon tests of variables:

, , Abl
         Sites vs Random Surveys vs Random
Wilcoxon   -1.201355e+01     -2.267643e+01
 p-value    3.016131e-33     7.655441e-114

, , Alluv
         Sites vs Random Surveys vs Random
Wilcoxon    6.066053e+00      8.956191e+00
 p-value    1.310920e-09      3.360875e-19
```

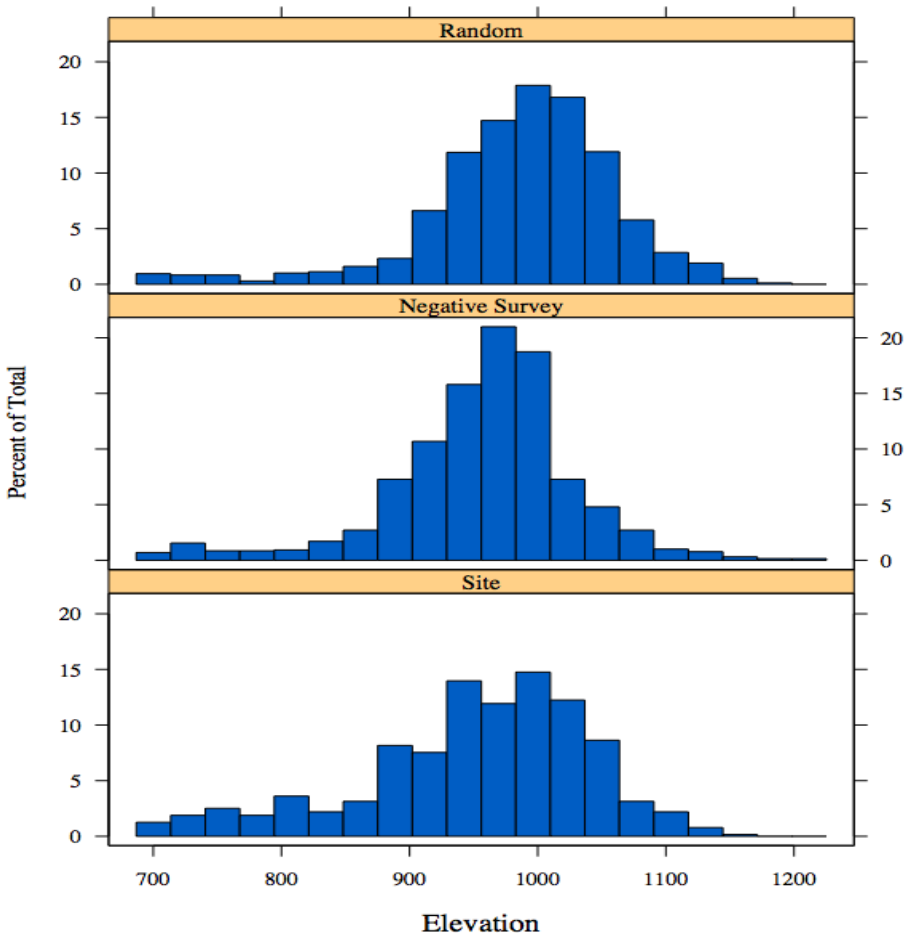
Finally, the Spearman rank correlation matrix of the variables is printed. The Spearman rank correlation coefficient indicates how two variables vary together: positive values indicate that they vary

directly; negative values indicate that they vary inversely. The further the coefficient is from zero, the stronger the relationship. The rank correlation coefficient is bounded between one and negative one. For example, output for five variables is (most data sets will have more than five variables, so there will be more than five rows and columns):

```
Spearman rank correlations:
            Abl  Alluv    Blg   Ht90 Rdedblk1
     Abl  1.000 -0.440  0.005  0.017   -0.369
   Alluv -0.440  1.000  0.070 -0.016    0.356
     Blg  0.005  0.070  1.000 -0.022    0.027
    Ht90  0.017 -0.016 -0.022  1.000   -0.033
Rdedblk1 -0.369  0.356  0.027 -0.033    1.000
```

After summary statistics, `mnmodel.var.summaries()` produces a PDF graphics file for each variable that is a stack of three histograms showing the distribution of the variable separately for sites, negative survey points, and random locations. The file is named var.1.pdf, where “var” is the variable name. For example, Figure 1 shows Abl.1.pdf for one region (elevation, 20 histogram classes).

Figure 1. Sample histograms for elevation separately for sites, negative surveys, and random locations.



5.5 Fitting a Model

Now we get to the meat of the modeling exercise, and the user has several potential directions to go. The function that does model fitting is `mnmodel.fit()`. This function has eight arguments, but we usually only use four of them and let the others take their default values. We begin with the four standard arguments and discuss the uncommon arguments later.

The first argument is a region abbreviation, such as `"BGWD"`. The second argument is a subsetting variable, such as `mod4.surv` . (A subsetting expression can be used, for example, by OR-ing two or more subsetting variables.) The third argument is a character string label used in constructing the name of the S-Plus variable where the results should be stored. These first three arguments are exactly analogous to the first three arguments of `mnmodel.var.summaries()`.

The fourth argument is a character string indicating which modeling technique to use. The available choices are `"biclogit"`, `"bmalogit"`, `"naive"`, `"tree"`, `"bumping"`, `"bagging"`, and `"double"`.

We recommend using `"bagging"`.

The method used in Phase 3 of Mn/Model is essentially `"biclogit"`. Also, `"biclogit"` and `"bmalogit"` are equivalent during the fitting stage, so you only need to do one of them. Details of how these methods work are given in Chapter 3 of "Statistical Methods for MN/Model Phase 4".

A typical use of `mnmodel.fit()` is thus

```
mnmodel.fit("BGWD",mod4.site,"mod4.site","bagging")
```

which would use the bagging method on the Big Woods data to fit a model to all Phase 4 site data. Remember: you must have read the region's data into S-Plus before attempting to fit a model.

The `mnmodel.fit()` function can be quite slow, depending on the method (`"biclogit"` and `"bmalogit"` are the slowest; `"tree"` is the fastest). However, it is doing a lot of work during that time. First,

`mnmodel.fit()` prints out some notification about the variables used and the numbers of cases being used, for example,

```
> mnmodel.fit("BGWD",mod4.site,"mod4.site","tree")
Constructing BGWD.mod4.site.results

Available predictors:
 [1] "Id"        "X"         "Y"         "Abl"       "Alluv"     "Blg"
 [7] "Ht90"      "Rdedblk1"  "Rdedcors"  "Rdedpriv"  "Rdisasbi"  "Rdiscon"
[13] "Rdishdw"   "Rdislkse"  "Rdismin"   "Rdismix"   "Rdisok"    "Rdispibf"
[19] "Rdispr"    "Rdisrb"    "Rdissug"   "Rel90a"    "Rgh90"     "Rlk1size"
[25] "Rlkinout"  "Rlkpinou"  "Rmajarea"  "Rplk1siz"  "Rwtpinou"  "Slp"
[31] "Soilcat"   "Terr"      "Site.type" "Phase3"    "Phase4"


Total number of locations: 7626
Total number of sites: 862
Total number of survey points: 0
Total number of non-random: 862
Total number of random: 1615
Eliminating 'Id', 'Easting', 'Northing', 'Phase3', and 'Phase4' from predictors

Eliminating previously transformed variables from predictors:
NULL



Eliminating collinear and/or nearly constant variables from predictors:
[1] "Rdismix"



Fitting with these variables:
 [1] "Abl"      "Alluv"    "Blg"      "Ht90"     "Rdedblk1" "Rdedcors"
 [7] "Rdedpriv" "Rdisasbi" "Rdiscon"  "Rdishdw"  "Rdislkse" "Rdismin"
[13] "Rdisok"   "Rdispibf" "Rdispr"   "Rdisrb"   "Rdissug"  "Rel90a"
[19] "Rgh90"    "Rlk1size" "Rlkinout" "Rlkpinou" "Rmajarea" "Rplk1siz"
[25] "Rwtpinou" "Slp"      "Soilcat"  "Terr"
```

After the notifications, it gets down to the business of fitting. First, it fits the model to the full data set; this is the “base” model. Then it does 10-fold cross-validation. In this process, the data are randomly divided into 10 subsets (this was done when the subsetting variable `CVsets` was formed for this region). Then we fit the model using only 9 of the 10 groups, and use the model fit from 90% of the data to predict on the one group held back. Next we cycle through the remaining 9 groups, hold each one out, fitting the model based on the other 9 groups, and predicting to the held out group. When finished, we have constructed 10 different prediction models beyond the base model and a set of predictions that were made without using the data being predicted.

If the variable SCVsets is available in the subsets for this region, we then do spatial cross-validation. This is similar to cross-validation except that the 10 groups consist of small spatial clusters instead of individual locations.

While all this fitting is going on, mnmodel.fit() will print out some progress information. For example, in the following output, a tree model has been fit, cross-validation has been completed, and we are now working on the sixth spatial cross-validation fit

```
Doing tree ... done.
Cross-validating, please be patient: 1  2  3  4  5  6  7  8  9  10   done.
Spatially cross-validating, please be patient: 1  2  3  4  5  6
```

When the fitting is done, mnmodel.fit() creates an S-Plus object (variable) named REGIONABBR.label.method.results. In the example above, this would be BGWD.mod4.site.tree.results. The S-Plus variable is an S-Plus list with seven named members. The first member is always a method-specific member containing the fitting results (variables, coefficients, etc) for the fitting method that was used. Methods "biclogit" and "bmalogit" both use a member named bmaresults; methods "tree" and "bumping" both use a member named treeresults; methods "bagging" and "naive" use members named baggingresults and naiveresults, respectively. Details of these members are rather arcane and will not be detailed here. As we are recommending "bagging", we simply state that baggingresults is an S-Plus list object (usually with 11 elements), with each element of the list being an S-Plus tree object. Details of the tree object are documented within S-Plus.

The other six members of the list are named preds, cvpreds, scvpreds, cvsets, scvsets, nonrandzero, and groups. The members cvsets and scvsets correspond to the variables CVsets and SCVsets in the subsetting data frame. The groups member is 0/1 with 1 indicating a target location (typically a site or survey location), and 0 indicating a non-target location (usually a random location); this is the response in the fitting problem. The preds member is the vector of predicted values using the full model. The cvpreds member is a matrix with 11 columns. The first 10 columns are the predictions (for all locations in the subset) using the 10 models derived during cross-validation. The last column is the cross-validated predictions combining the "out of sample" predictions in the first 10 columns. The

scvpreds member is analogous but is for spatial cross-validation. The nonrandzero member is a logical vector indicating nonrandom locations that are used as non-target locations.

The predictions (and cross-validated and spatial cross-validated predictions) are numbers, not categories, with larger values corresponding to locations less likely to be random points. The user must choose a threshold, and then declare locations with predictions above that threshold to be sites. Guidance on choosing the threshold is given below in the discussion of the mnmodel.fit.summaries() function.

Additional arguments. There are four additional arguments to mnmodel.fit(): site.prob, use.sampling.wts, verbose, and response. When you fit without specifying site.prob, you get predicted values that are larger when the location is more likely to be a site and smaller otherwise, but the actual predicted value cannot be interpreted. If you know the *a priori* probability that a location is a site, you may specify that probability using site.prob, for example, by including site.prob=.01 in the argument list for mnmodel.fit(). Specifying the *a priori* probability reweights the points during fitting so that the total fraction of weight given to sites is equal to the *a priori* probability. In that case, the model predictions can be interpreted as the probability of a location being a site. By default, the proportions of sites and non-sites in the sample are assumed to be the same as those in the population.

Using the argument verbose=FALSE will suppress the printed output.

The default model-fitting behavior is to assume that the random points in the subset are the non-target points and all other points in the subset are the target points. This is usually what we want to do, but may not always be what we want to do. For example, we might wish to fit a model with all Phase 4 sites as the targets, but use both the random points and the Phase 4 DOT survey points as the non-targets. Under the default behavior, the Phase 4 DOT points, being nonrandom, would be used as targets. To change this behavior, we must explicitly specify the response by setting the response argument. The response should be a logical vector where TRUE indicates a target location and FALSE indicates a non-target location. Continuing the example above, we would choose the subset of data to include in our

model via `subset=(mod4.site | p4.surv)` (all Phase 4 site points, all random points, and all Phase 4 DOT survey points), and we would set the response via `response=mod4.site` (only archaeological site points used as targets).

The last argument is `sampling.wts`, which permits us to re-weight the non-random points during fitting. Classical sampling theory says that points should be weighted by the reciprocal of their probability of inclusion into the sample when computing summary statistics. The survey models show us that not all locations are equally likely to be surveyed for archaeological artifacts, so it might be worthwhile to re-weight the sites we do have by the reciprocals of their probabilities of being surveyed.

When `sampling.wts` is NULL, then all data points receive the same weight during model fitting. You may optionally set different weights for the non-random locations by setting the `sampling.wts` argument equal to your chosen weights. (Random locations all get weight 1. They were selected by a different mechanism than the survey and site locations and are all equally likely regardless of the mechanism that produced the survey and site locations. You may adjust the relative weight of the random and non-random locations by setting `site.prob`.) For example, suppose that we have previously fit a survey model to Big Woods data and that we now wish to use the sampling probabilities from that model when fitting a site model for the Big Woods. The region abbreviation is `"bgwd4"`, the survey model subset is `"mod4.surv"` and the site model subset is `"mod4.site"`. The regional data are in `bgwd4.data.all`, and the subsetting variables are in `bgwd4.subsets`. We can access a subsetting variable, say `mod4.site`, via `bgwd4.subsets$mod4.site`. The first thing we must do is extract the data for the non-random locations included in the site modeling subset and store it in a new data frame, say `bgwd4.site.data`.

```
> bgwd4.site.data <- bgwd4.data.all[bgwd4.subsets$mod4.site &
bgwd4.subsets$no.rand,]
```

In this command, we have selected the rows of `bgwd4.data.all` (`[rows,cols]` indicates element selection) that are used in the site model (`mod4.site` is true) and are nonrandom (`no.rand` is true).

We next use the `mnmodel.predict()` function (see Section 5.8 below) to apply the survey model to the selected data to estimate the survey probabilities for these site locations and then store the probabilities in `bgwd4.sampling.prob`. The output of `mnmodel.predict()` is a two column matrix, with the first column being the estimated probabilities of survey and the second column our actual prediction of surveyed or not. We only need the first column, so we select that using `[,1]` .

```
> bgwd4.sampling.prob <- mnmodel.predict("bgwd4","mod4.surv","bagging",
      bgwd4.site.data)[,1]
```

Next we compute the reciprocals and store them in `bgwd4.sampling.wts`

```
> bgwd4.sampling.wts <- 1/bgwd4.sampling.prob
```

Finally, we fit the site model using these sampling weights.

```
> mnmodel.fit("bgwd4",mod4.site,"mod4.site.wtd",method="bagging",
      sampling.wts=bgwd4.sampling.wts)
```

Warning: The usefulness of this approach depends on getting good weights. If we use reciprocal sampling probabilities, then we need good sampling probabilities. Getting those good probabilities can be a problem. Our models will mostly be used to classify locations into low, medium, and high probabilities of having a site or having a survey. When we are interested in this kind of classification, then only the order of our estimated probabilities matters, not their actual magnitudes. However, when we are going to use these probabilities to form weights, then we need to get the magnitudes correct as well. This is more difficult, and in particular requires an accurate value for `site.prob` when the survey model is fit. (When fitting a survey model, `site.prob` indicates the proportion of the study area that has been surveyed.)

5.6 Fit Summaries

After fitting a model, you may use the `mnmodel.fit.summaries()` function to obtain several evaluations of fit quality, guidance on selecting the cutoff for producing classifications, and information on how the actual fit is done (variables, coefficients, and so on). The arguments to `mnmodel.fit.summaries()` are a region abbreviation, a label, a method, and an optional *a priori* probability for sites. The region, label, and method are character strings used to identify the

results to summarize. The *a priori* probability is used only to prepare the gain curves (as described in the chapter on model evaluation). This probability is set separately and independently from the *a priori* probability used to fit the model, which is not carried forward to the summary stage. If you want to use the same *a priori* probability at both the model fitting and model evaluation stages, you must set them both to your desired value; you may set them to differing values if you desire. (The *a priori* probability specified during model fitting adjusts the overall level of the estimated probabilities up and down; the *a priori* probability specified here affects the gain curves only.) Thus, a typical usage might be `mnmodel.fit.summaries("BGWD","mod4.site","bagging",.001)` . This summary would be computed based on information stored in the S-Plus variable `REGIONABBR.label.method.results,` or in our example, `BGWD.mod4.site.bagging.results`.

Output from `mnmodel.fit.summaries()` goes into the REGIONABBR.label.summaries directory, which in our example is BGWD.mod4.site.summaries. The output consists of a text file named method.summary.txt (bagging.summary.txt in the example) and either 8 or 13 summary graphics in pdf format, the number depending on whether spatially cross-validated predictions are available in the results variable (`REGIONABBR.label.method.results`).

Let us begin with a discussion of the graphs, which are produced for the predictions, cross-validated predictions, and, if available, spatially cross-validated predictions. For these kinds of predictions we produce cumulative plots, ROC curves, and gain curves. (See Chapter 2 of "Statistical Methods for Mn/Model Phase 4" on model evaluation for an explanation of these plots, their construction, and their interpretation.) For cross-validated data, we also produce graphs with multiple cumulative curves and ROC curves, one for each cross-validation subset. The graphs are stored in the files with the names:

1. appcumpred.method.pdf, appgain.method.pdf, approc.method.pdf.

2. cvcumpred.method.pdf, cvgain.method.pdf, cvroc.method.pdf, cvcumpred.multi.method.pdf, cvroc.multi.method.pdf.

3. scvcumpred.method.pdf, scvgain.method.pdf, scvroc.method.pdf, scvcumpred.multi.method.pdf, scvroc.multi.method.pdf.

The “app” prefix stands for “apparent” and indicates results for predictions constructed using all the data. The “cv” and “scv” prefixes indicate results for cross-validated and spatially cross-validated predictions. The “multi” indicates separate curves for each cross-validation subset. In all cases, the “method” is replaced with the method of interest, for example, approc.bagging.pdf.

The cumulative predicted plots allow us to see the actual values predicted for different locations and to compare the distributions of predictions for sites and non-sites. An example cumulative predicted plot is shown in Figure 2.

We see two bands of points, the lower band for random points and the upper band for sites. The curves show the cumulative distributions, that is, fractions of points with values less than or equal to the current value, for random points (dashed) and sites (solid). For a good prediction, the random points should cluster to the left and their cumulative should rise steeply and then flattening near 1; and the sites should cluster to the right, with their cumulative staying low and then rising sharply on the right. The greater the area between the two curves, the better the prediction method is doing.

The “multi” form of the plot shows the same points, but the cumulative curves are plotted separately for the 10 cross-validation subsets. This illustrates the variability in the quality of the prediction. This is illustrated in Figure 3. In this example, the curves are fairly stable across the subsets.

Figure 2. Cumulative predicted plot showing separate curves for sites (solid) and non-sites (dotted).

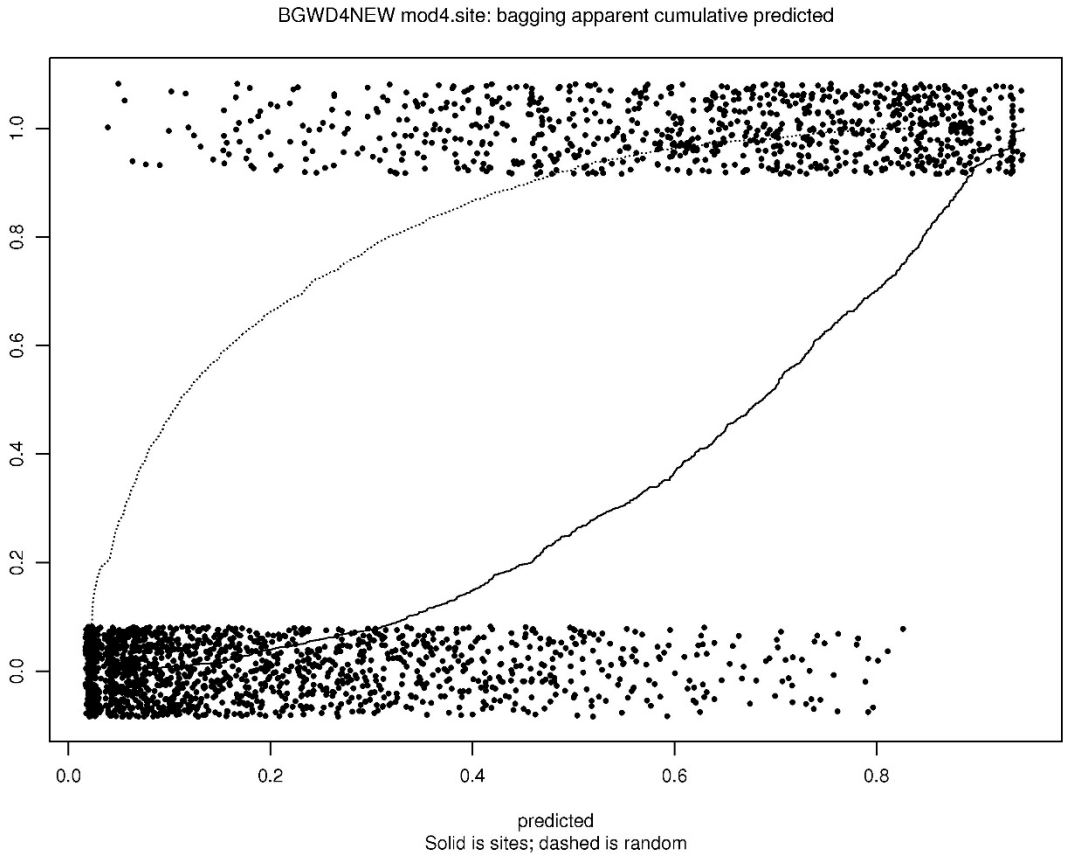
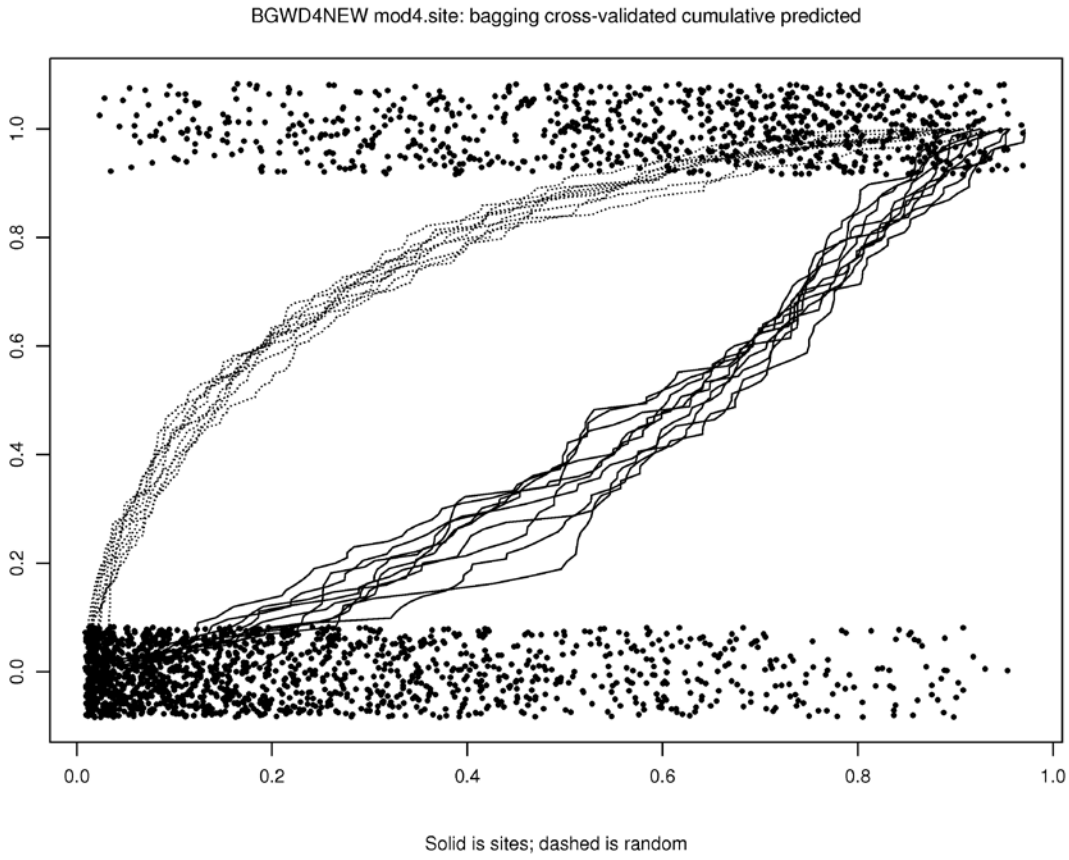


Figure 3. Multiple form of cumulative predicted plots, showing separate curves for each cross-validation subset.



The ROC curve plots the true positive rate (the rate for sites, vertical axis) against the false positive rate (rate for random locations, horizontal axis) for a large number of potential thresholds. The curve starts in the lower left hand corner and moves to the upper right hand corner. Ideally, the curve should move up to the top very quickly and then move along the upper boundary. This gives us maximum true positive rate with minimum false positive rate. The diagonal line corresponds to randomly guessing site versus random. The false positive rates for .7 and .85 true positive rates are highlighted. While our principal figure of merit is the false positive rate at a .85 true positive rate, the area under the ROC curve gives an overall summary of the quality of the prediction. Figure 4 shows a sample ROC curve based on cross-validated predictions.

The "multi" form of the graph plots the ROC curve separately for each cross-validation subset. Figure 5 shows an example of these multiple ROC curves for the same data shown in Figure 4. Again we see that the ROC curves are fairly stable.

The final graph is the gain curves. These curves are already somewhat complex, so we only plot them for the full data set, and not separately by cross-validation subsets. The gain curve for the same data used above and assuming an *a priori* probability of .01 is shown in Figure 6.

Figure 4. ROC curve for cross-validated predictions.

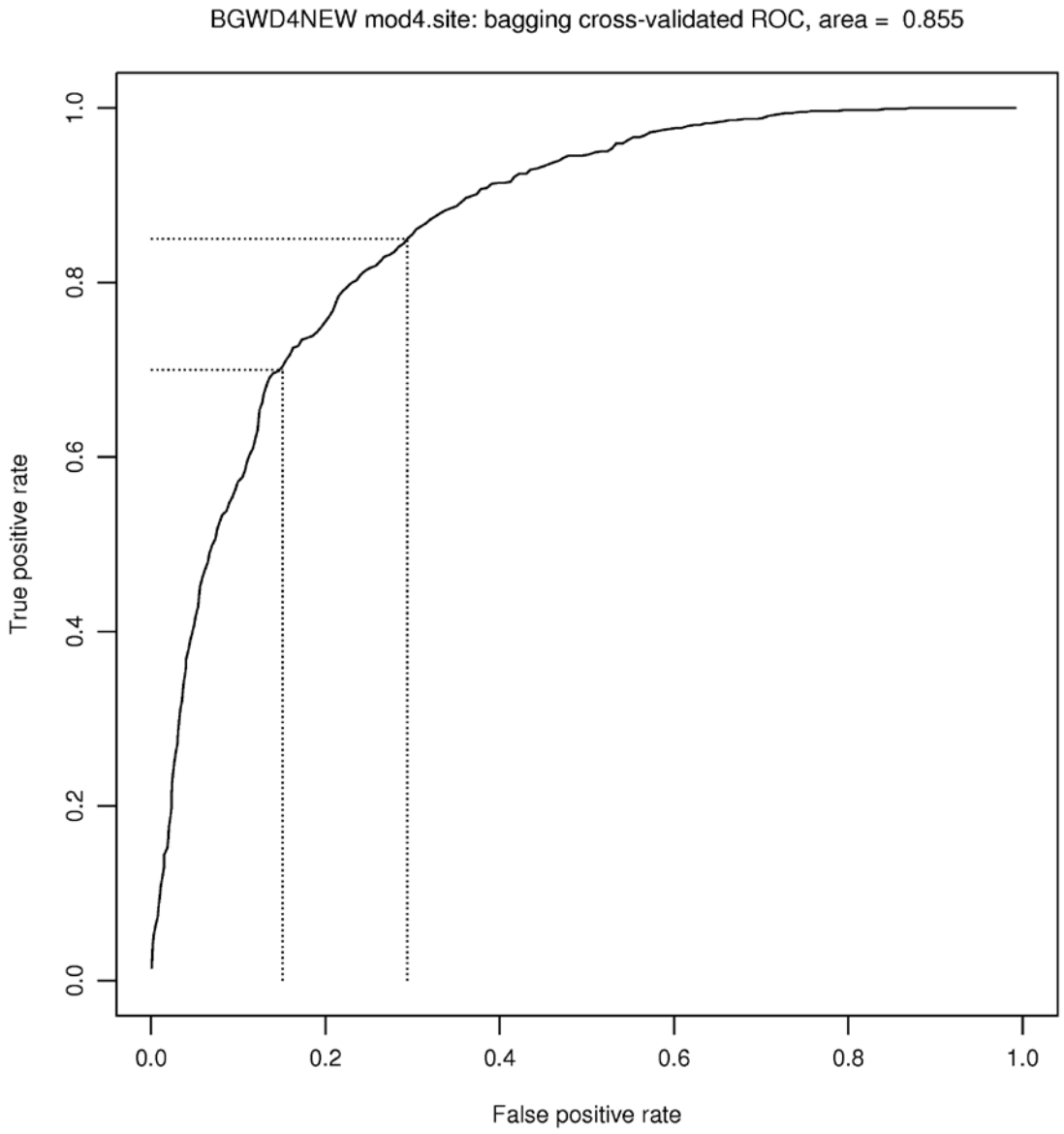


Figure 5. ROC curves separately for each cross-validation subset.

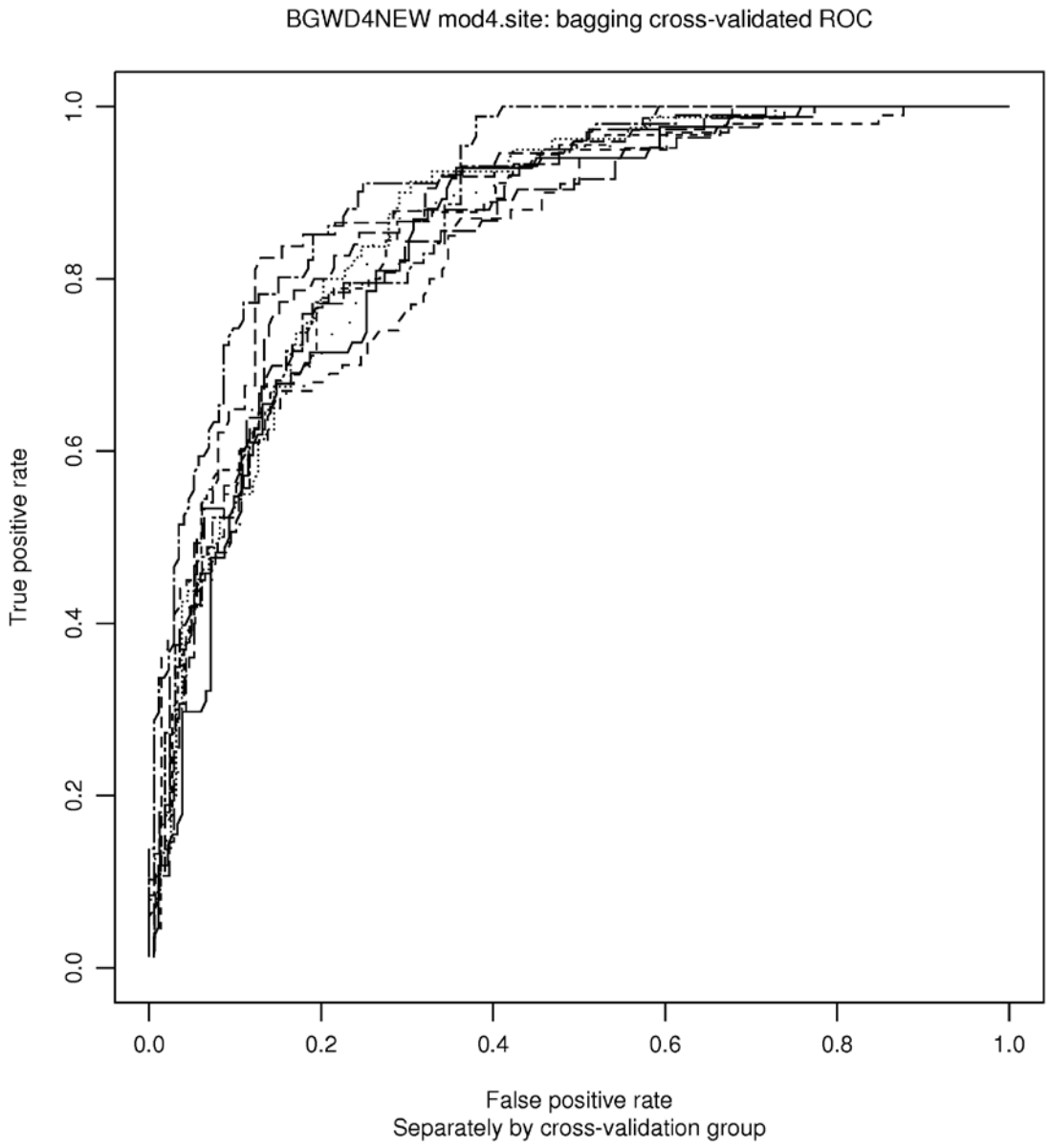
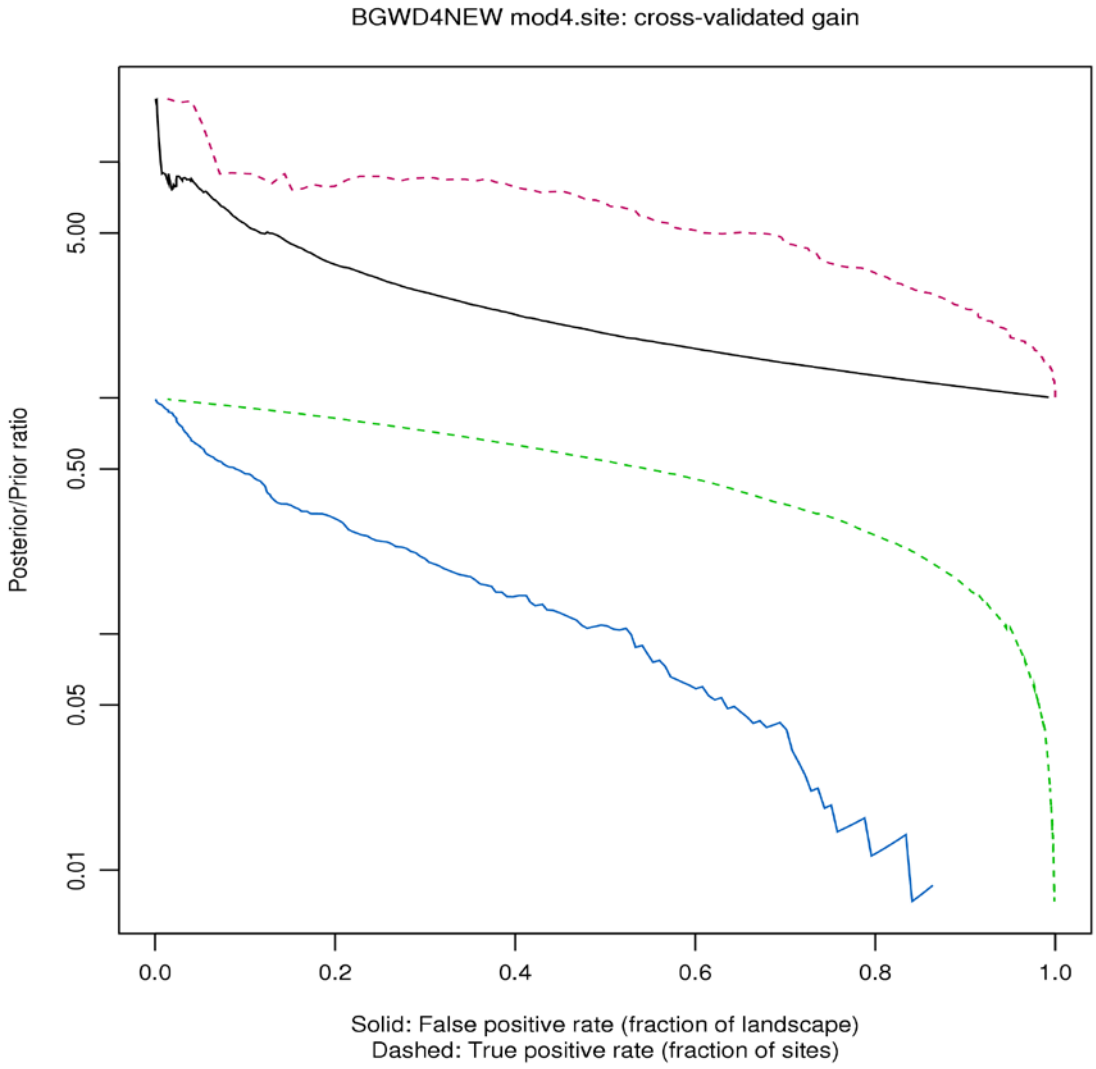


Figure 6. Gain curves for sites and non-sites.



In addition to graphical summaries, we also have text/numeric summaries of the results in a text file named method.summary.txt (bagging.summary.txt in the example). These files consist of a method-specific part and a generic part. We begin with an example of the generic part and describe the contents.

```
Apparent false positive rates and cutoffs
at .70 and .85 true positive rates
          Cutoff  TPR  FPR
  HIGH 0.5425123 0.70 0.06
MEDIUM 0.4015590 0.85 0.13
```

The output begins with apparent false positive rates, which are based on predictions from the full data set. Here is how to interpret the output. For this prediction (bagging in this example), if we use a cutoff of .5424 and declare locations with predictions greater than the cutoff to be sites, then we capture 70% of the sites (true positive rate of 70%) but only 6% of the random points (false positive rate of 6%). Similarly, if we choose a cutoff of .4016, then we capture 85% of the sites and 13% of the random points.

If there are non-random locations that are also non-target locations, then the FPR will also be computed separately for the random non-target locations and the non-random, non-target locations.

These rates are called apparent rates, because it appears that those are the rates that we will obtain. However, these rates are based on predictions applied to data used to build the predictions, and that leads to overly optimistic estimates of the quality of prediction.

To get around that, we use cross-validation. Our 10-fold cross-validation fits the model using 90% of the data and then uses that model to predict the 10% of the data held out. We cycle through 10 different subsets of hold out data until all locations were predicted using models fit excluding the points of interest. Cross-validated results are summarized next.

```
Cross-validated true and false positive rates and
cutoffs at nominal .70 and .85 true positive rates

, , HIGH
            Cutoff   TPR   FPR
       1 0.5631354 0.620 0.120
       2 0.5608065 0.660 0.150
       3 0.5445611 0.610 0.110
       4 0.5719469 0.650 0.090
       5 0.5702637 0.520 0.100
```

```
       6 0.5490074 0.650 0.080
       7 0.5730262 0.530 0.100
       8 0.5436854 0.520 0.110
       9 0.5745496 0.560 0.100
      10 0.5516697 0.600 0.110
 Average 0.5602652 0.592 0.107
Combined 0.4685972 0.700 0.150

, , MEDIUM
            Cutoff   TPR   FPR
       1 0.4419980 0.700 0.180
       2 0.4047221 0.750 0.190
       3 0.3844049 0.710 0.190
       4 0.4217128 0.850 0.190
       5 0.4093691 0.710 0.170
       6 0.4227110 0.800 0.160
       7 0.4164569 0.720 0.170
       8 0.3916342 0.740 0.230
       9 0.4150209 0.690 0.210
      10 0.4208373 0.770 0.160
 Average 0.4128867 0.744 0.185
Combined 0.2901404 0.850 0.290
```

Begin with the “High” results, which are supposed to capture 70% of the sites, and consider group 1. Here we fit a model to the other 90% of the data and find the cutoff that gives us 70% true positive rate in the data used to fit the model. Here the cutoff is .5631. We now apply this model and cutoff to the 10% of the data held back. When we do this, we find that we actually obtain a 62% true positive rate (instead of the nominal 70%) and a 12% false positive rate. Similarly, when we hold back group 10, we select a cutoff of .5517 and obtain a TPR of 60% and a FPR of 11%. Averaging across the 10 groups (the line labeled “Average”), the cutoff is .5603, the TPR is 59%, and the FPR is 11%. The average cutoff of .5603 is pretty close to the apparent cutoff from above (.5425), so that is reassuring, but the TPR that we actually attain is well below the 70% that we wanted. To get 70% TPR, we must reduce the cutoff to .4686 (the line labeled “Combined”), which gives us a FPR of 15%. Thus the apparent cutoffs are too optimistic, and we really should use a lower cutoff to attain 70% TPR when predicting to new data.

The “Medium” results tell a similar story. They should have a TPR of 85%, but when cross-validated the TPR only averages about 74%. We need to lower the threshold to .2901 to capture 85% of the sites when predicting to new data.

Results from spatial cross-validation are even more pessimistic, but that is because they simulate a much more challenging modeling

situation. In cross-validation, we predict using models fit without benefit of the data we are trying to predict. However, since the 10 subsets are chosen randomly, there is a good chance that landscapes similar to where we are trying to predict are included in the modeling subset. In spatial cross-validation, we exclude data in small spatial clusters rather than one point at a time. Spatial cross-validation simulates predicting into landscapes where we've never had any data before! We have to expect that our predictions will work more poorly, and, unfortunately, our expectations are met.

Spatial cross-validation results are reported just like the cross-validation results. For example, when trying to capture 85% of the sites, our average threshold is .412 and we achieve an average TPR of 67% with a FPR of 20%. To get our desired TPR of 85%, we must lower the threshold to .235, which also gives us a FPR of 36%. These results should be compared with .290 and FPR of 29% for simple cross-validation.

```
Spatial cross-validated true and false positive rates and
cutoffs at nominal .70 and .85 true positive rates

, , HIGH
            Cutoff  TPR   FPR
       1 0.5596008 0.44 0.080
       2 0.5643144 0.49 0.130
       3 0.5588316 0.59 0.140
       4 0.5548809 0.55 0.100
       5 0.5436384 0.44 0.070
       6 0.5487280 0.62 0.170
       7 0.5603275 0.33 0.080
       8 0.5615844 0.53 0.100
       9 0.5505675 0.50 0.040
      10 0.5500290 0.51 0.130
 Average 0.5552503 0.50 0.104
Combined 0.3914553 0.70 0.210

, , MEDIUM
            Cutoff   TPR   FPR
       1 0.4405121 0.480 0.160
       2 0.4155323 0.660 0.270
       3 0.4074808 0.760 0.240
       4 0.4073247 0.760 0.210
       5 0.4140435 0.590 0.130
       6 0.4046139 0.790 0.250
       7 0.3948178 0.440 0.180
       8 0.4083685 0.740 0.200
       9 0.4052451 0.760 0.140
      10 0.4205485 0.700 0.200
 Average 0.4118487 0.668 0.198
Combined 0.2351554 0.850 0.360
```

Overall, we recommend using the cross-validated cutoffs.

If you know that you are predicting into a landscape unlike the rest of your data set, you could consider the more pessimistic spatial cross-validated threshold.

Method-specific output. There are three basic types of output reflecting the three basic types of predictions: logistic regression, tree structured regression, and naïve Bayes classification.

Summaries for "bmalogit" and "biclogit" are nearly identical. Internally, they search for different subsets of predictor variables that will form good models. Usually, there are several plausible models, with some models more likely and others less likely. The "biclogit" technique chooses the single most likely model, and the "bmalogit" technique takes a weighted average of the coefficients of the most likely models (weighted by the probability of the model).

The summary output begins with information about the most likely models: their probability, the number of predictors, and their BIC. BIC is the Bayesian Information Criterion, and models with lower BIC are generally preferred.

```
Basic model comparison:
   Post. Prob.       BIC Size of model
 1  0.26493293 -16912.09            12
 2  0.16364680 -16911.13            13
 3  0.08981361 -16909.93            13
 4  0.05780320 -16909.04            13
 5  0.04630060 -16908.60            14
 6  0.04212704 -16908.41            13
 7  0.03932683 -16908.27            13
 8  0.03921897 -16908.27            14
 9  0.03542374 -16908.06            13
10  0.03316238 -16907.93            14
11  0.02868477 -16907.64            14
12  0.02226626 -16907.14            14
13  0.01834238 -16906.75            13
14  0.01804442 -16906.72            14
15  0.01800146 -16906.71            14
16  0.01755544 -16906.66            11
17  0.01725541 -16906.63            14
18  0.01665255 -16906.56            14
19  0.01609317 -16906.49            15
20  0.01534805 -16906.39            13
```

The next summary list information about which variables appear in which models. For each variable, we get the probability that its coefficient is non-zero and an indication of in which of the top three models that the variable appears. (Variables may appear in later

models. For example, below we see that Blg has a 14.2% probability of being nonzero, but it does not appear in the top three models.)

```
Variables present in top 3 models:
         probne0   1   2   3
     Abl 100       X   X   X
   Alluv 1.8
     Blg 14.2
    Ht90 100       X   X   X
Rdedblk1 100       X   X   X
Rdedcors 0
Rdedpriv 100       X   X   X
Rdisasbi 100       X   X   X
 Rdiscon 100       X   X   X
 Rdishdw 0
Rdislkse 7.4
 Rdismin 9.3
  Rdisok 98.2      X   X   X
Rdispibf 94.4      X   X   X
  Rdispr 0
  Rdisrb 0
 Rdissug 9.2
  Rel90a 0
   Rgh90 0
Rlk1size 21.6              X
Rlkinout 100       X   X   X
Rlkpinou 100       X   X   X
Rmajarea 0
Rplk1siz 100       X   X   X
Rwtpinou 39.5          X
     Slp 1.5
 Soilcat 100       X   X   X
    Terr 0
```

Next comes the coefficients in the top three models, and the posterior mean coefficients (the weighted average mentioned above). The "bmalogit" method uses the posterior mean coefficients, and the "biclogit" method uses the coefficients from model 1.

```
Coefficients in top 3 models:
            Post. mean            1             2             3
Intercept  1.854609e+00  1.422326670  1.2141522745  1.5496547151
      Abl -7.560788e-03 -0.008040118 -0.0068289419 -0.0080729536
    Alluv -8.763240e-03  0.000000000  0.0000000000  0.0000000000
      Blg -3.048708e-04  0.000000000  0.0000000000  0.0000000000
     Ht90  5.918020e-02  0.059978366  0.0584924367  0.0601699716
 Rdedblk1 -3.220591e-02 -0.032390598 -0.0322334545 -0.0329433940
 Rdedcors  0.000000e+00  0.000000000  0.0000000000  0.0000000000
 Rdedpriv -1.575067e-02 -0.015570510 -0.0166737554 -0.0143074235
 Rdisasbi -5.371819e-02 -0.051261278 -0.0616858985 -0.0511421582
  Rdiscon  3.720780e-02  0.030264146  0.0468975321  0.0304027313
  Rdishdw  0.000000e+00  0.000000000  0.0000000000  0.0000000000
 Rdislkse -3.765354e-04  0.000000000  0.0000000000  0.0000000000
  Rdismin  8.668204e-04  0.000000000  0.0000000000  0.0000000000
   Rdisok -6.923538e-03 -0.006627322 -0.0078798367 -0.0065628494
 Rdispibf  3.393696e-02  0.037717899  0.0346270936  0.0374464515
   Rdispr  0.000000e+00  0.000000000  0.0000000000  0.0000000000
```

```
   Rdisrb  0.000000e+00  0.000000000  0.0000000000  0.0000000000
  Rdissug -6.909364e-04  0.000000000  0.0000000000  0.0000000000
   Rel90a  0.000000e+00  0.000000000  0.0000000000  0.0000000000
    Rgh90  0.000000e+00  0.000000000  0.0000000000  0.0000000000
 Rlk1size -9.036624e-05  0.000000000  0.0000000000 -0.0004105098
 Rlkinout -2.201561e-02 -0.021014656 -0.0237648855 -0.0207090667
 Rlkpinou  1.333016e-02  0.012958104  0.0146946673  0.0114264030
 Rmajarea  0.000000e+00  0.000000000  0.0000000000  0.0000000000
 Rplk1siz  4.439313e-04  0.000364858  0.0003730413  0.0007049974
 Rwtpinou -2.252879e-03  0.000000000 -0.0050202120  0.0000000000
      Slp  6.081871e-04  0.000000000  0.0000000000  0.0000000000
  Soilcat  9.387926e-02  0.091343506  0.0954787101  0.0939202058
     Terr  0.000000e+00  0.000000000  0.0000000000  0.0000000000
```

The final summary differs between the "bmalogit" and "biclogit" methods. This summary lists the rank correlation between the predictors and the linear combination of predictors (the logit or linear predictor) used in the apparent model, the cross-validated model, and the spatial cross-validated model. The logits differ for the two modeling approaches, so these correlations will differ between "bmalogit" and "biclogit".

```
Rank correlations between data and logits
               logits    cvlogits   scvlogits
   Alluv  0.230266867  0.22981774  0.23084153
     Blg -0.074374290 -0.07711332 -0.07386792
    Ht90  0.452984945  0.44949740  0.45874668
Rdedblk1 -0.501243748 -0.50312317 -0.50093067
Rdedcors -0.152337766 -0.15124266 -0.15897507
Rdedpriv -0.320652511 -0.31537619 -0.31508663
Rdisasbi -0.172859240 -0.17439194 -0.17912158
 Rdiscon -0.184700500 -0.18545665 -0.18766181
 Rdishdw -0.048516413 -0.05139217 -0.05244087
Rdislkse -0.290043226 -0.29156415 -0.28080221
 Rdismin  0.114881869  0.11460543  0.11858553
  Rdisok -0.069104809 -0.05987480 -0.05352862
Rdispibf -0.102472790 -0.10359924 -0.10793041
  Rdispr -0.006630704  0.00102292  0.01403406
  Rdisrb -0.265350004 -0.26699279 -0.25956227
 Rdissug -0.129842966 -0.13469836 -0.14501984
  Rel90a  0.472220221  0.46907741  0.48163937
   Rgh90 -0.058003506 -0.06046620 -0.05061741
Rlk1size  0.348888164  0.34916660  0.34219420
Rlkinout -0.489080227 -0.48932536 -0.48921647
Rlkpinou -0.371260503 -0.37728736 -0.39687056
Rmajarea -0.056736081 -0.05762397 -0.05563201
Rplk1siz  0.365974589  0.36701180  0.35511252
Rwtpinou -0.165211381 -0.16761171 -0.17133908
     Slp  0.363403454  0.36099889  0.37230756
 Soilcat  0.139554376  0.13648177  0.13144280
    Terr  0.075990737  0.07374489  0.07888598
```

Both "bumping" and "tree" use a single S-Plus tree object to do prediction, and this tree object is summarized in the method-specific output file. We first show the output and then explain it.

```
Summaries for region: BGWD4NEW  subset:  mod4.site  method:  tree

Regression tree:
snip.tree(tree = out, nodes = c(59., 233., 10., 198., 98., 199., 56., 23.,
        13., 931., 22., 117., 30., 9.))
Variables actually used in tree construction:
 [1] "Rdedblk1" "Ht90"     "Rdiscon"  "Rdislkse" "Rdisasbi" "Abl"
 [7] "Rdedpriv" "Rplk1siz" "Soilcat"  "Rwtpinou" "Rdispibf" "Slp"
Number of terminal nodes:  24
Residual mean deviance:  0.1354 = 332.1 / 2453
Distribution of residuals:
       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
 -8.465e-01 -2.108e-01 -4.687e-02 -7.122e-17  1.667e-01  9.615e-01
```

The output begins with some overall summary information. The most useful bits of information are which variables are actually used in the making splits in the tree (in this example, only 12 of the 28 available variables were actually used), and the number of terminal nodes (24 here). One way of thinking of the terminal nodes is that each terminal node is one element of a partition of the predictor space. In this example, there are 24 distinct but exhaustive regions of predictor space, each of these regions gets its own predicted value, and every location within those regions is predicted with the same value. Each split adds one terminal node, so there must be 23 splits in this tree.

The next part of the output is the tree itself. The tree is labeled node by node, beginning with the root node (all data). Each node is described by a node number, the split criterion that formed the node from its parent node, the number of data cases in the node, the deviance (here the sum of squares of the 0/1 data around the mean value of the node), and the mean value of the response for the data in the node. When a node is a terminal node, it is marked by an asterisk, and any case that would fall into that node is predicted by the mean value for the node. In the example below, node 8 includes 24 cases and has a predicted value of .8333. The locations that fall into this node have `Rdedblk1 < 17.5` and `Ht90 < 5.5` and `Rdiscon < 234.5`.

The general structure of the tree is that node 1 is the root node. It is split into nodes 2 and 3. Node 2 is split into nodes 4 and 5, while node 3 is split into nodes 6 and 7. Node 4 is split into 8 and 9, 5 is split into 10 and 11, 6 is split into 12 and 13, and 7 is splint into 14 and 15. This continues until the nodes are too small to split (minimum of 40 locations in order to split a node). However, only some of the splits actually improve the predictive ability of the tree, and the unneeded

splits are pruned off. The tree that remains contains only the useful splits and nodes.

The split criterion tells how to split the parent node. For example, the root node 1 splits into nodes 2 and 3 with criteria `Rdedblk1 < 17.5` and `Rdedblk1 >= 17.5`. So the first node is split based just on the `Rdedblk1` variable and whether or not it meets a threshold split value. Node 2 splits into nodes 4 and 5, which have criteria `Ht90 < 5.5` and `Ht90 >= 5.5`. So all the cases in node 2 (which are those with `Rdedblk1 < 17.5`) are split into two groups according to the value of the variable `Ht90` and the threshold split value 5.5.

```
node), split, n, deviance, yval
      * denotes terminal node

   1) root 2477 562.000 0.34800
     2) Rdedblk1<17.5 501 113.000 0.65670
       4) Ht90<5.5 152  37.050 0.42110
         8) Rdiscon<234.5 24   3.333 0.83330 *
         9) Rdiscon>234.5 128  28.870 0.34380 *
       5) Ht90>5.5 349  63.780 0.75930
        10) Rdislkse<205.5 215  27.930 0.84650 *
        11) Rdislkse>205.5 134  31.590 0.61940
          22) Rdisasbi<289 54  13.200 0.42590 *
          23) Rdisasbi>289 80  15.000 0.75000 *
     3) Rdedblk1>17.5 1976 389.200 0.26970
       6) Abl<924 449 110.600 0.56120
        12) Rdedpriv<31.5 356  81.400 0.64610
          24) Rdedblk1<95.5 318  68.920 0.68240
            48) Rdiscon<236 57  14.140 0.45610
              96) Ht90<12 26   4.038 0.19230 *
              97) Ht90>12 31   6.774 0.67740 *
            49) Rdiscon>236 261  51.230 0.73180
              98) Rdiscon<255.5 132  18.330 0.83330 *
              99) Rdiscon>255.5 129  30.140 0.62790
               198) Rdedblk1<80 63  15.560 0.44440 *
               199) Rdedblk1>80 66  10.440 0.80300 *
          25) Rdedblk1>95.5 38   8.553 0.34210 *
        13) Rdedpriv>31.5 93  16.800 0.23660 *
       7) Abl>924 1527 229.300 0.18400
        14) Ht90<18.5 1283 152.600 0.13800
          28) Rdislkse<98.5 110  26.760 0.41820
            56) Rplk1siz<838.5 76  15.200 0.27630 *
            57) Rplk1siz>838.5 34   6.618 0.73530 *
          29) Rdislkse>98.5 1173 116.400 0.11170
            58) Rdisasbi<329.5 597  85.880 0.17420
             116) Soilcat<3 550  69.070 0.14730
               232) Rwtpinou<206.5 262  46.800 0.23280
                 464) Rdispibf<353.5 52   1.923 0.03846 *
                 465) Rdispibf>353.5 210  42.420 0.28100
                   930) Rdisasbi<276.5 44  10.910 0.54550
                    1860) Slp<1.5 20   3.750 0.25000 *
                    1861) Slp>1.5 24   3.958 0.79170 *
                   931) Rdisasbi>276.5 166  27.620 0.21080 *
               233) Rwtpinou>206.5 288  18.610 0.06944 *
```

```
      117) Soilcat>3 47  11.740 0.48940 *
     59) Rdisasbi>329.5 576  25.730 0.04687 *
  15) Ht90>18.5 244  59.670 0.42620
    30) Abl<1015.5 155  38.480 0.54190 *
    31) Abl>1015.5 89  15.510 0.22470
      62) Rdedblk1<25.5 20   4.200 0.70000 *
      63) Rdedblk1>25.5 69   5.478 0.08696 *
```

The "bagging" method works by producing multiple trees (11 by default in our S-Plus functions) and then averaging the predictions from these multiple trees. The method-specific output for "bagging" is simply the descriptions of the multiple trees, in the same format we just saw for "tree" and "bumping". The "double" method does bagging twice, so its output consists of two bagging structures.

The "naive" method works by estimating a completely arbitrary function of each variable, and then summing the function values seen at the observed data across all the variables. For any given variable value x, we are trying to estimate the log likelihood ratio for sites to non-sites at that x value. Very roughly speaking, the log likelihood ratio is the (natural) logarithm of the ratio of the probabilities that sites and non-sites take the value x on the variable of interest. The output specific to the "naive" method is an attempt to represent this arbitrary function. For each variable, there is a 50x2 matrix, with the first column a list of potential x values, and the second column the estimated values of the log likelihood ratio (labeled the `log den ratio`) at those x values. For example,

```
, , abl
          x log den ratio
 1  695.0000  -0.653172475
 2  704.7959  -0.803247266
 3  714.5918  -0.853943074
 4  724.3878  -0.792476422
 5  734.1837  -0.617606304
 6  743.9796  -0.352090085
 7  753.7755  -0.050855577
 8  763.5714   0.214850070
 9  773.3673   0.409039253
10  783.1633   0.536368787
11  792.9592   0.569937575
12  802.7551   0.475112115
13  812.5510   0.294739469
14  822.3469   0.120170351
15  832.1429   0.026262449
16  841.9388   0.061139381
17  851.7347   0.241157875
18  861.5306   0.510851606
19  871.3265   0.733966386
20  881.1224   0.792048500
21  890.9184   0.674939495
```

```
22  900.7143   0.451860794
23  910.5102   0.211938867
24  920.3061   0.028683358
25  930.1020  -0.062972114
26  939.8980  -0.070104545
27  949.6939  -0.029318601
28  959.4898   0.009082630
29  969.2857   0.003393434
30  979.0816  -0.050176201
31  988.8776  -0.114837043
32  998.6735  -0.151914837
33 1008.4694  -0.149773450
34 1018.2653  -0.119922528
35 1028.0612  -0.082204208
36 1037.8571  -0.050041824
37 1047.6531  -0.018681516
38 1057.4490   0.032289248
39 1067.2449   0.118647107
40 1077.0408   0.228888720
41 1086.8367   0.319928695
42 1096.6327   0.350747887
43 1106.4286   0.321169952
44 1116.2245   0.253784266
45 1126.0204   0.153283364
46 1135.8163   0.012746823
47 1145.6122  -0.162386400
          x log den ratio
48 1155.408    -0.3627156
49 1165.204    -0.5940992
50 1175.000    -0.8785598
```

In these data, the log likelihood ratio is negative for an `abl` value of 695, indicating that 695 is more likely to arise from a non-site. On the other hand, the log likelihood ratio is positive when `abl` is 871, indicating that 871 is more likely to arise from a site.

5.7 Export to GIS

Mn/Model predictions are done on a production scale through GIS. ArcGIS can now do scripting through Python, and the S-Plus functions have a limited ability to prepare Python scripts that may be of use. The S-Plus function is `mnmodel.make.python()`. This function can produce Python implementations for the `"tree"` and `"bagging"` methods. The required arguments to `mnmodel.make.python()` are a region abbreviation, a label, and a method. So, for example,

```
mnmodel.make.python("BGWD","mod4.site","bagging")
```

This will create a file method.py (here, bagging.py) in the directory region.label.summaries (here BGWD.mod4.site.summaries). In this

file are a sequence of Python commands that should implement the prediction method.

By default, `mnmodel.make.python()` will make predictions with cross-validated true positive rate .85. You may select another cross-validated true positive rate with the `tpr` argument by adding, for example, `tpr=.70` to the arguments of the function, as in

```
mnmodel.make.python("BGWD","mod4.site","bagging",tpr=.70)
```

Alternatively, you may directly set the threshold for declaring a location to be a site by using the threshold argument, as in

```
mnmodel.make.python("BGWD4NEW","mod4.site","tree",
threshold=.21)
```

The python produced by this command would be in the file tree.py in the directory BGWD4NEW.mod4.site.summaries. One possible python script is shown here:

```
if Rdedblk1<17.5:
  if Ht90<5.5:
    if Rdiscon<234.5:
      thisout = 0.833333333333333
     else:
      thisout = 0.34375
   else:
    if Rdislkse<205.5:
      thisout = 0.846511627906977
     else:
      if Rdisasbi<289:
        thisout = 0.425925925925926
       else:
        thisout = 0.75
 else:
  if Abl<924:
    if Rdedpriv<31.5:
      if Rdedblk1<95.5:
        if Rdiscon<236:
          if Ht90<12:
            thisout = 0.192307692307692
           else:
            thisout = 0.67741935483871
         else:
          if Rdiscon<255.5:
            thisout = 0.833333333333333
           else:
            if Rdedblk1<80:
              thisout = 0.444444444444444
             else:
              thisout = 0.803030303030303
       else:
        thisout = 0.342105263157895
```

```
     else:
      thisout = 0.236559139784946
   else:
    if Ht90<18.5:
      if Rdislkse<98.5:
        if Rplk1siz<838.5:
          thisout = 0.276315789473684
         else:
          thisout = 0.735294117647059
       else:
        if Rdisasbi<329.5:
          if Soilcat<3:
            if Rwtpinou<206.5:
              if Rdispibf<353.5:
                thisout = 0.0384615384615385
               else:
                if Rdisasbi<276.5:
                  if Slp<1.5:
                    thisout = 0.25
                   else:
                    thisout = 0.791666666666667
                 else:
                  thisout = 0.210843373493976
             else:
              thisout = 0.0694444444444445
           else:
            thisout = 0.48936170212766
         else:
          thisout = 0.046875
     else:
      if Abl<1015.5:
        thisout = 0.541935483870968
       else:
        if Rdedblk1<25.5:
          thisout = 0.7
         else:
          thisout = 0.0869565217391305
prediction = 0
if thisout >= 0.21:
  prediction = 1
```

The structure of this output is a set of nested if, then, else constructs representing the tree, which results in setting the variable `thisout`. After `thisout` is set, it is compared with the `threshold` (here .21). Locations above the threshold are set to 1 (a site), while other locations are set to 0 (a non-site).

If you specify both a `threshold` and a `tpr`, the `threshold` will overrule the `tpr`.

5.8 Making Predictions

Once you have fit a model, you can make predictions using the fitted model via the `mnmodel.predict()` function. This function has four required arguments: a region, a label, a method, and new data in an S-Plus data frame.

```
mnmodel.predict("BGWD","mod4.site","bagging",new.data)
```

The variables in the new data frame must include all of the variables originally used to fit the model. The output is a data frame with two variables: `score`, which is the numerical prediction, and `prediction`, which is either `"Site"` or `"Non-site"`. By default, a threshold is chosen to provide a true positive rate of .85. You may select your own threshold or your own true positive rate by adding, for example, `tpr=.70` or `threshold=.21` to the argument list. Again, if you specify both, the `threshold` will overrule the `tpr`.

Just for amusement, let's apply the Anoka site model to the Mille Lacs data. Then we'll look at a few of the results.

```
> tmp <- mnmodel.predict("anok4ex","mod4.site","bagging",mlac4ex.data.all)
> tmp[11:20,]
        score prediction
11 0.01837041   Non-site
12 0.41278223       Site
13 0.35958376   Non-site
14 0.22432380   Non-site
15 0.04041375   Non-site
16 0.35825936   Non-site
17 0.10856103   Non-site
18 0.57498622       Site
19 0.05165805   Non-site
20 0.46228055       Site
```

For these ten locations (all random locations), two were predicted to be sites and the other eight were predicted to be non-sites.

6. Appendix

In this appendix, we describe again in more compact form the standard functions that users call. We also describe some of the internal structure of those functions and some additional functions that they call internally. In the discussion below, we will see lines of the form

```
function.name <- function(arg1, arg2=defval, ...)
```

This is the the first line of the formal S-Plus definition of a function. We have the function `function.name()`. It has a first argument `arg1`, and a second argument `arg2` with the default value of `defval`. That is, if `arg2` is not specified when the function is called, `defval` will be used. Default values are specified via the `arg=value` construction. The `"..."` allows additional arguments. These arguments are not used explicitly in this function, but they can be passed on to other functions that are called internally.

6.1 Reading Data

The `mnmodel.readdata()` function reads data from an external file into S-Plus and constructs several S-Plus data objects.

```
mnmodel.readdata <- function(region,rootvars=NULL,cmult=40)
```

`region` is a character scalar giving a region name.

`rootvars` determines which variables are transformed to square roots. If it is `NULL`, then a default list of variables is transformed; otherwise, the variables given in `rootvars` will be transformed. (Order of variable names in `rootvars` does not matter.) To transform no variables, use something like `rootvars="no.such.variable"`.

`cmult` may determine the number of spatial clusters. If the data set contains a variable named `Clusters`, then that variable will determine the spatial clusters. If `Clusters` is not present in the data (but `X` and `Y` are), the function will generate `10*cmult` spatial clusters and randomly group them into 10 groups for spatial cross-validation.

`mnmodel.readdata()` and several other functions internally call the function `mnmodel.find.var()`, which is used for extracting a named variable from an S-Plus data frame.

```
mnmodel.find.var <- function(varname,thisframe)
```

`varname` is a character string giving the variable name, and `thisframe` is an S-Plus data frame.

`mnmodel.find.var()` first tries the variable name given in `varname`. If that is not found, it then tries tries `varname` converted to all lower case. If that is not found, it tries removing periods `'.'`, underscores `'_'`, and spaces `' '` from `varname`.

If at any stage it gets a match, it returns that variable from the data frame. If there is no match, it returns a `NULL`.

6.2 Summarizing Variables

After reading in data you can summarize the variables in a data set using the `mnmodel.var.summaries()` function. This function computes several numerical summaries (means and standard deviations, rank correlations, Wilcoxon rank sum tests between random locations and sites or negative surveys) for each variable. It also plots histograms comparing the sites, negative survey, and random points.

```
mnmodel.var.summaries <- function(region,subset=mod4.surv,label="mod4.surv")
```

`region` is a character scalar giving the name of a region.

`subset` is an expression using variables in the subsetting frame that selects the data to be used. Note, subset should normally be a survey subset, so that there are data included from sites, negative surveys, and random locations.

`label` is a character scalar used to label the output directory and name the analysis.

When plotting, histograms are labeled with "long labels" based on the short variable names in the data set. The function `mnmodel.get.var.label()` attempts to find a long label for each variable name. The variable name and long label pairs are stored in the matrix `mnmodel.var.names.and.labels`, which you may need to update with new names and labels.

```
mnmodel.get.var.label <- function(shortlab,underscore=FALSE,exact=FALSE)
```

`shortlab` is a short label variable name, for example, "Blg". The function tries to find its long name: "Prevailing orientation".

`underscore` is a logical flag. If TRUE, spaces in the long name are replaced by underscores (for example, "Prevailing_orientation")

`exact` is a logical flag. If exact is TRUE, we try to match the short label exactly. Otherwise, we also try the short label converted to all lower case, then the lower case short label with spaces, periods, and underscores removed.

If the function cannot match `shortlab`, then `shortlab` will be returned as the value.

6.3 Fitting Models

The meat of the exercise is fitting a prediction model. This is done in `mnmodel.fit()`. The basic structure of this function is that it uses the `method` argument to determine a method-specific fitting function, which it then calls, passing all of its arguments to the method-specific function.

```
mnmodel.fit <- function(region,subset,label,method="bagging",
      site.prob=NULL, sampling.wts=NULL,verbose=TRUE,response=no.rand,...)
```

`region` is a character scalar giving the name of a region.

`subset` is an expression using variables in the subsetting frame to select the subset of data for analysis.

`label` is a character string giving a descriptive label for the results of this analysis.

`method` is a character string giving the method to be used. Current choices include `'bmalogit'`, `'biclogit'`, `'tree'`, `'naive'`, `'bagging'`, `'double'`, and `'bumping'`. We recommend 'bagging'.

`site.prob` is `NULL`, or a numeric value strictly between 0 and 1 that represents the *a priori* probability of a site. If numeric, weights for non-site locations are rescaled so that the total weight for sites is `site.prob` fraction of total weight for all locations.

`sampling.wts` is `NULL` or a numeric vector of positive numbers (with length equal to the number of non-random locations in the analysis). If `NULL`, equal sampling weights are used. If non-`NULL`, `sampling.wts` should contain sampling weights for the non-random locations in the selected subset. These should be reciprocals of the sampling probability for non-random locations. This only makes sense for site models.

`verbose` is a logical flag. If `TRUE`, informative output is printed.

`response` is an expression using variables in the subsetting frame for selecting the non-zero (target or site) responses. All variables in the subsetting frame can be used. The default is `no.rand`, so anything non-random is a 1 (a site) and anything random is a 0 (a non-site).

The method-specific functions take the same arguments (except for `method`). The functions for bagging, double, and bumping take additional arguments `bagK` or `bumpK`. When bagging or bumping, we fit with the original data and with several additional bootstrap samples. `bagK` and `bumpK` determine the number of additional bootstrap samples.

```
mnmodel.bagging.fit <- function(region,subset,label,site.prob=NULL,
       sampling.wts=NULL,verbose=TRUE,response=no.rand,bagK=10,...)

mnmodel.double.fit <- function(region,subset,label,site.prob=NULL,
       sampling.wts=NULL,verbose=TRUE,response=no.rand,bagK=10,...)

mnmodel.bumping.fit <- function(region,subset,label,site.prob=NULL,
       sampling.wts=NULL,verbose=TRUE,response=no.rand,bumpK=10,...)

mnmodel.tree.fit <- function(region,subset,label,site.prob=NULL,
       sampling.wts=NULL,verbose=TRUE,response=no.rand,...)

mnmodel.biclogit.fit <- function(region,subset,label,site.prob=NULL,
       sampling.wts=NULL,verbose=TRUE,response=no.rand,...)

mnmodel.bmalogit.fit <- function(region,subset,label,site.prob=NULL,
       sampling.wts=NULL,verbose=TRUE,response=no.rand,...)

mnmodel.naive.fit <- function(region,subset,label,site.prob=NULL,
       sampling.wts=NULL,verbose=TRUE,response=no.rand,...)
```

The method-specific fitting functions all do the same basic thing. Each of them calls `mnmodel.prepare.for.fit()` first. This checks arguments, prints some information, and selects the appropriate subset of data. After selecting the data, the method-specific functions

call the actual fitting functions to do model fitting and prediction (including cross-validation).

```
mnmodel.prepare.for.fit <- function(region,subset,label,site.prob,
       sampling.wts=NULL,verbose=TRUE, eliminate.linear=TRUE,response=no.rand)
```

The arguments to `mnmodel.prepare.for.fit()` are mostly the same as those of the method-specific functions. The only new one is `eliminate.linear`, which is a logical flag saying whether variables that are linearly dependent or nearly constant should be removed prior to fitting. This is done by default.

The functions that actually do the fitting (mostly) have a simple interface. In these functions, `x` is a matrix of predictors for the subset of interest, `y` is a 0/1 vector of responses for the subset, and `wts` is a vector of case weights for fitting. Some functions have an additional argument to declare the number of bootstrap samples to use.

```
mnmodel.bagging <- function(x,y,wts,bagK)
mnmodel.double <- function(x,y,wts,bagK)
mnmodel.bumping <- function(x,y,wts,bumpK)
mnmodel.tree <- function(x,y,wts)
mnmodel.naive <- function(x,y,wts)
```

The one exception is that `bic.glm()` is used for biclogit and bmalogit.

```
bic.glm <- function (x, y, glm.family, wt=rep(1,nrow(x)),strict=F,
       prior.param=c(rep(0.5,ncol(x))),OR=20, OR.fix=2, nbest=150,
       dispersion=NULL, factor.type=T, factor.prior.adjust=F)
```

This function was written by Chris Volinsky at the University of Washington and is licensed for free distribution and use for non-commercial purposes. Copyright 1996, 1997 by Chris T. Volinsky.

6.4 Summarizing Model Fits

After making a fit, we summarize the fit with the `mnmodel.fit.summaries()` function. This function prints details of the fitting method (coefficients, and so on), prints thresholds and false positive rates at 70% and 85% true positive rates for the complete data set, prints thresholds and attained true and false positive rates for nominal 70% and 85% true positive rates for all cross-validation subsets as well as cross-validated thresholds for 70% and 85% true positive rates. After the numerical summaries, it generates a number

of diagnostic plots, including cumulative prediction, ROC curves, and gain curves.

```
mnmodel.fit.summaries<-function(region,label,method="bagging",site.prob=.01)
```

`region`, `label`, and `method` have their usual meanings.

`site.prob` is the *a priori* probability of a location being a site, used in computing gain curves.

`mnmodel.fit.summaries()` calls method-specific functions to print the method-specific prediction information. For each of these functions, `results` is an internal S-Plus variable with an arcane format. These functions print that information out in a somewhat readable form. For bumping and tree, the functions also make a plot of the tree which is put in a file tree.pdf in the directory named `dirname`. `dirname` is otherwise ignored.

```
mnmodel.bagging.summaries <- function(results,dirname)
mnmodel.double.summaries <- function(results,dirname)
mnmodel.bumping.summaries <- function(results,dirname)
mnmodel.tree.summaries <- function(results,dirname)
mnmodel.bmalogit.summaries <- function(results,dirname)
mnmodel.biclogit.summaries <- function(results,dirname)
mnmodel.naive.summaries <- function(results,dirname)
```

`mnmodel.fit.summaries()` uses the function `mnmodel.getrates()`to compute apparent, cross-validated, and spatially cross-validated false positive rates.

```
mnmodel.getrates <- function(preds,groups,subsets=NULL,nonrandzero)
```

`preds` contains the predictions. For apparent rates, `preds` is a vector of predictions. For cross-validated data, `preds` is a matrix with 11 columns. The first 10 columns are complete-sample predictions based on the 10 cross-validation models; the last column collects the out-of-sample predictions.

`groups` is the 0/1 vector of responses (1 is a site).

If `subsets` is non-`NULL`, it is a vector containing elements 1 through 10 indicating the cross-validation subsets.

nonrandomzero is a vector of logical values. A TRUE represents a non-random location that has a 0 response (zeros are usually random locations). If there are any non-random zeros, the false positive rates are also computed separately for the random and non-random zeros.

After computing numerical summaries, mnmodel.fit.summaries() calls two functions repeatedly to generate cumulative predicted, ROC, and gain plots for apparent, cross-validated, and spatially cross-validated data.

```
mnmodel.eval.plots <- function(predicted,groups,cumfilename,cumtitle,
            rocfilename,roctitle,gainfilename,gaintitle,site.prob=.01)
```

predicted is the vector of "predicted" values from some method (this could actually be cross validated). Larger values should go with sites.

groups is the 0/1 response variable with 1 for sites and 0 for non-sites.

cumfilename, rocfilename, and gainfilename are names of files to contain the pdf output.

cumtitle, roctitle, and gaintitle are character strings for labeling the plots.

site.prob is the *a priori* probability of a site. This is used to compute the gain plots.

There is also a "multi" form of this function that plots curves for each cross-validation subset. The arguments are analogous, with the additional argument subsets, which is a vector containing 1 through 10 indicating the cross-validation subsets.

```
mnmodel.eval.plots.multi <- function(predicted,groups,subsets,
            cumfilename,cumtitle,rocfilename,roctitle)
```

6.5 Making Predictions

While our goal is to do prediction from within GIS, we can also do prediction from within S-Plus. Indeed, we must be able to do this to do cross-validation. The function mnmodel.predict() applies a fitted

model to new data to do prediction. It has the now familiar form of a function that calls method-specific functions internally.

```
mnmodel.predict <- function(region,label,method="bagging",
       newdata,threshold=NULL,tpr=.85)
```

`region`, `label`, and `method` have their usual meanings.

`newdata` is an S-Plus data frame containing the data values for which we wish to make predictions. `newdata` should contain all of the variables that were used to fit the original model (including any transformed variables), although order of the variables does not matter.

`tpr` is a desired true positive rate.

`threshold` is `NULL` or a cutoff used for splitting predictions into sites (above the cutoff) and non-sites (others).

If a `threshold` is given, it will override any `tpr` value. If `threshold` is `NULL`, then a threshold will be computed to obtain the desired true positive rate based on cross-validated data.

The method-specific functions called internally are:

```
mnmodel.bagging.predict <- function(baggingresults,newx)
mnmodel.double.predict <- function(doubleresults,newx)
mnmodel.bmalogit.predict <- function(bmaresults,thisdata)
mnmodel.biclogit.predict <- function(bicresults,thisdata)
mnmodel.naive.predict <- function(naiveresults,newx)
predict.tree(tree,newx)
```

`predict.tree()` is a built-in S-Plus function.

The output of `mnmodel.predict()` is a data frame with two variables: `score`, which is the numerical prediction, and `prediction`, which is either "`Site`" or "`Non-site`" based on the selected `threshold` or `tpr`. For example,

```
        score prediction
 1 0.01837041   Non-site
 2 0.41278223       Site
 3 0.35958376   Non-site
 4 0.22432380   Non-site
 5 0.04041375   Non-site
 6 0.35825936   Non-site
 7 0.10856103   Non-site
```

```
 8 0.57498622       Site
 9 0.05165805   Non-site
10 0.46228055       Site
```

shows three locations predicted as sites (scores above .4) and seven predicted as non-sites (scores below .4).