ModelMap: an R Package for Model Creation and Map Production

Elizabeth A. Freeman, Tracey S. Frescino, Gretchen G. Moisen

April 21, 2010

Abstract

The **ModelMap** package (Freeman, 2009) for R (R Development Core Team, 2008) enables user-friendly modeling, validation, and mapping over large geographic areas though a single R function or GUI interface. It constructs predictive models of continuous or discrete responses using Random Forests or Stochastic Gradient Boosting. It validates these models with an independent test set, cross-validation, or (in the case of Random Forest Models) with Out OF Bag (OOB) predictions on the training data. It creates graphs and tables of the model validation diagnostics. It applies these models to GIS image files of predictors to create detailed prediction surfaces. It will handle large predictor files for map making, by reading in the GIS data in sections,thus keeping memory usage reasonable.

1 Introduction

Maps of tree species presence and silvicultural metrics like basal area are needed throughout the world for a wide variety of forest land management applications. Knowledge of the probable location of certain key species of interest as well as their spatial patterns and associations to other species are vital components to any realistic land management activity. Recently developed modeling techniques such as Random Forest (Breiman, 2001) and Stochastic Gradient Boosting (Friedman, 2001, 2002) offer great potential for improving models and increasing map accuracy (Evans and Cushman, 2009; Moisen et al., 2006).

The R software environment offers sophisticated new modeling techniques, but requires advanced programming skills to take full advantage of these capabilities. In addition, spatial data files can be too memory intensive to analyze easily with standard R code. The **ModelMap** package provides an interface between several existing R packages to automate and simplify the process of model building and map construction.

While spatial data is typically manipulated within a Geographic Information System (GIS), the **ModelMap** package facilitates modeling and mapping extensive spatial data in the R software environment. **ModelMap** has simple to use GUI prompts for non-programmers, but still has the flexibility to be run at the command line or in batch mode, and the power to take full advantage of sophisticated new modeling techniques. **ModelMap** uses the **rgdal** package to read and predict over GIS raster data. Large maps are read in sections, to keep memory usage reasonable.

The current implementation of **ModelMap** builds predictive models using Random Forests or Stochastic Gradient Boosting. Random Forest models are constructed using the **randomForest** package (Liaw and Wiener, 2002) and Stochastic Gradient Boosting models are constructed using the **gbm** package (Ridgeway, 2007). The **ModelMap** package models both continuous and binary response variables. For binary response, the **PresenceAbsence** package (Freeman, 2007) package is used for model diagnostics.

Both Random Forest and Stochastic Gradient Boosting models are built as an ensemble of classification or regression trees (Breiman et al., 1984). Classification and regression trees are intuitive methods, often described in graphical or biological terms. Typically shown growing upside down, a tree begins at its root. An observation passes down the tree through a series of splits, or nodes, at which a decision is made as to which direction to proceed based on the value of one of the explanatory variables. Ultimately, a terminal node or leaf is reached and predicted response is given.

Trees partition the explanatory variables into a series of boxes (the leaves) that contain the most homogeneous collection of outcomes possible. Creating splits is analogous to variable selection in regression. Trees are typically fit via binary recursive partitioning. The term binary refers to the fact that the parent node will always be split into exactly two child nodes. The term recursive is used to indicate that each child node will, in turn, become a parent node, unless it is a terminal node. To start with a single split is made using one explanatory variable. The variable and the location of the split are chosen to minimize the impurity of the node at that point. There are many ways to minimizing the impurity of each node. These are known as splitting rules. Each of the two regions that result from the initial split are then split themselves according to the same criteria, and the tree continues to grow until it is no longer possible to create additional splits or the process is stopped by some user-defined criteria. The tree may then be reduced in size using a process known as pruning. Overviews of classification and regression trees are provided by De'ath and Fabricius (2000), Vayssieres et al. (2000), and Moisen (2008).

While classification and regression trees are powerful methods in and of themselves, much work has been done in the data mining and machine learning fields to improve the predictive ability of these tools by combining separate tree models into what is often called a committee of experts, or ensemble. Random Forests and Stochastic Gradient Boosting are two of these newer techniques that use classification and regression trees as building blocks.

Random Forests — In a Random Forests model, a bootstrap sample of the training data is chosen. At the root node, a small random sample of explanatory variables is selected and the best split made using that limited set of variables. At each subsequent node, another small random sample of the explanatory variables is chosen, and the best split made. The tree continues to be grown in this fashion until it reaches the largest possible size, and is left un-pruned. The whole process, starting with a new bootstrap sample, is repeated a large number of times. As in committee models, the final prediction is a (weighted) plurality vote or average from prediction of all the trees in the collection.

Stochastic Gradient Boosting — Stochastic gradient boosting is another ensemble technique in which many small classification or regression trees are built sequentially from pseudo-residuals from the previous tree. At each iteration, a tree is built from a random sub-sample of the dataset (selected without replacement) producing an incremental improvement in the model. Ultimately, all the small trees are stacked together as a weighted sum of terms. The overall model accuracy gets progressively better with each additional term.

2 Package Overview

The **ModelMap** package for R enables user-friendly modeling, diagnostics, and mapping over large geographic areas though simple R function calls: `model.build()`, `model.diagnostics()`, and `model.mapmake()`. The function `model.build()` constructs predictive models of continuous or discrete responses using Random Forests or Stochastic Gradient Boosting. The function `model.diagnostics()` validates these models with an independent test set, cross-validation, or (in the case of Random Forest Models) with Out OF Bag (OOB) predictions on the training data. This function also creates graphs and tables of the model validation diagnostics. The function `model.mapmake()` applies the models to GIS image files of predictors to create detailed prediction surfaces. This function will handle large predictor files for map making, by reading in the GIS data in sections, thus keeping memory usage reasonable.

2.1 Interactive Model Creation

The **ModelMap** package can be run in a traditional R command line mode, where all arguments are specified in the function call. However, in a Windows environment, **ModelMap** can also be used in an interactive, pushbutton mode. If the functions `model.build()`, `model.diagnostics()`, and `model.mapmake()` are called without argument lists, pop up windows ask questions about the type of model, the file locations of the data, response variable, predictors, etc ...

To provide a record of the options chosen for a particular model and map, a text file is generated each time these functions are called, containing a list of the selected arguments.

This paper concentrates on the traditional command line function calls, but does contain some tips on using the GUI prompts.

2.2 File Names

File names in the argument lists for the functions can be provided either as the full path, or as the base name, with the path specified by the folder argument. However, file names in the Raster Look Up Table (the `rastLUTfn`, described in section 2.8) must include the full path.

2.3 Training Data

Training and test data can be supplied in two forms. The argument `qdata.trainfn` can be either an R data frame containing the training data, or the file name (full path or base name) of the comma separated values (CSV) training data file. If a filename is given, the file must be a comma-delimited text file with column headings. The data frame or CSV file should include columns for both response and predictor variables.

In a Windows environment, if `qdata.trainfn = NULL` (the default), a GUI interface prompts the user to browse to the training data file.

Note: If `response.type = "binary"`, any response with a value greater than `0` is treated as a presence. If there is a cutoff value where anything below that value is called trace, and treated as an absence, the response variable must be transformed before calling the functions.

2.4 Independent Test Set for Model Validation

The argument `qdata.testfn` is the file name (full path or base name) of the independent data set for testing (validating) the model's predictions, or alternatively, the R data frame containing the test data. The column headings must be the same as those in the training data (`qdatatrainfn`).

If no test set is desired (for example, cross-validation will be performed, or RF models with out-of-bag estimation), set `qdata.testfn = FALSE`.

In a Windows environment, if `qdata.testfn = NULL` (default), a prompt will ask the a test set is available, and if so asks the user to browse to the test data file. If no test set is available, the a prompt asks if a proportion of the data should be set aside as an independent test set. If this is desired, the user will be prompted to specify the proportion to set aside as test data, and two new data files will be generated in the output folder. The new file names will be the original data file name with `"_train"` and `"_test"` pasted on the end.

2.5 Missing predictor values

There are three circumstances that can lead to having missing predictor values. First, there are true missing values for predictors within the study area. Second, there are categorical predictors with categories that are present in the test or mapping data but not in the training data. And finally, portions of the mapping rectangle lie outside of the study area. Each of the three cases is handled slightly differently by **ModelMap**.

In the first instance, true `NODATA` values in the test set or within the study area for production mapping could be caused by data collection errors. These are data points or pixels for which you may still need be interested in a prediction based on the other remaining predictors. These missing values should be coded as `NA`. (Note: in Imagine image files, values of the specified `NODATA` value will be read into R as `NA`.) The argument `na.action` will determine how these `NA` pixels will be treated. There are 2 options: (1) `na.action = "na.omit"` (the default) where any data point or pixel with any `NA` predictors is returned as `-9999`; (2) `na.action = "na.roughfix"` where before making predictions, a missing categorical predictor is replaced with the most common category for that predictor, and a missing continuous predictor is replaced with the median for that predictor and a warning message is generated.

The second type of missing value occurs when using categorical predictors. There may be cases where a category is found in the validation test set or in the map region that was not present in the training data. This is a particularly common occurrence when using cross-validation on a small dataset. Again, the argument `na.action` will determine how these data points or pixels are treated. If `na.action = "na.omit"`, no prediction will be made for these locations. If `na.action = "na.roughfix"`, the most common category will be substituted for the unknown category. In either instance, a warning will be generated with a list of the categories that were missing from the training data. After examining these categories, you may decide that rather than ignoring these locations or substituting the most common category, a better option would be to collapse similar categories into larger groupings. In this case you would need to pre-process your data and run the predictions again.

The final type of missing predictor occurs when creating maps of non-rectangular study regions. There may be large portions of the rectangle where you have no predictors, and are uninterested in making predictions. The suggested value for the pixels outside the study area is `-9999`. These pixels will be ignored, thus saving computing time, and will be exported as `-9999`. Any value other than `-9999` will be treated as a legal data value and a prediction will be generated for each pixel.

Note: in Imagine image files, if the specified `NODATA` is set as `-9999`, any `-9999` pixels will be read into R as `NA`, and if `na.action = "na.roughfix"`, predictions will be attempted for these pixels. This will cause the computation time to increase, and these predictions will need to be masked out when the final map is imported back into a GIS.

2.6 The Model Object

The two available model types (Random Forest and Stochastic Gradient Boosting) are stochastic models. If a seed is not specified (with argument `seed`) each function call will result in a slightly different model. The function `model.build()` returns the model object. To keep this particular model for use in later R sessions, assign the function output to an R object, then use the functions `save()` and `load()`.

Random Forest is implemented through the **randomForest** package within R. Random Forest is more user friendly than Stochastic Gradient Boosting, as it has fewer parameters to be set by the user, and is less sensitive to tuning of these parameters. The number of predictors used to select the splits (the `mtry` argument) is the primary user specified parameter that can affect model performance, and the default for **ModelMap** is to automatically optimize this parameter with the `tuneRF()` function from the randomForest package. In most circumstance, Random Forest is less likely to over fit data. For an in depth discussion of the possible penalties of increasing the number of trees (the `ntree` argument) see Lin and Jeon (2002). The **randomForest** package provides two measures to evaluate variable importance. The first is the percent increase in Mean Standard Error (MSE) as each variable is randomly permuted. The second is the increase in node purity from all the splits in the forest based on a particular variable, as measured by the gini criterion

(Breiman, 2001). These importance measures should be used with caution when predictors vary in scale or number of categories (Strobl et al., 2007).

Stochastic gradient boosting is implemented through the **gbm** package within R. Like Random Forest, Stochastic gradient boosting also provides measures of variable importance. In this case, it is the relative influence as described in Friedman (2001). Stochastic Gradient Boosting is more challenging for a user, in that it requires a greater number of user specified parameters, and the SGB models tend to be quite sensitive to these parameters. Model fitting parameters available from the **gbm** package include shrinkage rate, interaction depth, bagging fraction, training fraction, and the minimum number of observations per terminal node. In the **ModelMap** package, values for these parameters can be set in the argument list when calling `model.build()`. Friedman (2001, 2002) and Ridgeway (2002) provide guidelines on appropriate settings for these model parameters. In addition, the supplementary materials in Elith et al. (2008) provide R tools to tune model parameters, and also to select the most relevant variables. Models created with the code from Elith et al. (2008) can be used with the **ModelMap** package functions `model.diagnostics()` and `model.mapmake()` to run additional diagnostics and predict over large raster grids.

2.7 Spatial Raster Layers

The **ModelMap** uses the **rgdal** package to read spatial rasters into R. The data for predictive mapping in **ModelMap** should be in the form of pixel-based raster layers representing the predictors in the model. The layers must also be in either ESRI Grid or ERDAS Imagine image (single or multi-band) raster data formats, having continuous or categorical data values. For effective model development and accuracy, if there is more than one raster layer, the layers must have the same extent, projection, and pixel size.

The function `model.mapmake()` outputs an ASCII grid file of map information suitable to be imported into a GIS. Small maps can also be imported back into R using the function `read.asciigrid()` from the **sp** package (Pebesma and Bivand, 2005).

The supplementary materials in Elith et al. (2008) also contain R code to predict to grids imported from a GIS program, including large grids that need to be imported in pieces. However this code requires pre-processing of the raster data in the GIS software to produce ASCII grids for each layer of data before they can be imported into R. **ModelMap** simplifies and automates this process, by reading Arcinfo grids and Imagine image files directly, (including multi band images). **ModelMap** also will verify that the extent of all rasters is identical (same cell size, same number of row and columns, corners line up) and will produce informative error messages if this is not true. **ModelMap** also simplifies working with masked values and missing predictors.

2.8 Raster Look Up Table

The Raster Look Up Table (`rastLUTfn`) provides the link between the spatial rasters for map production and the column names of the Training and Test datasets. The Raster Look Up Table can be given as an R data frame specified by the argument `rastLUTfn` or read in from a CSV file specified by `rastLUTfn`.

The `rastLUTfn` must include 3 columns: (1) the full path and base names of the raster file or files; (2) the column headers from the Training and Test datasets for each predictor; (3) the layer (band) number for each predictor. The names (column 2) must match not only the column headers in Training and Test data sets (`qdata.trainfn` and `qdata.testfn`), but also the predictor names in the arguments `predList` and `predFactor`, and the predictor names in `model.obj`.

In a windows environment, the function `build.rastLUT()` may be used to help build the look-up-table with the aid of GUI prompts.

Name	Type	Description
ELEV250	Continuous	90m NED elevation (ft) resampled to 250m, average of 49 points
NLCD01_250	Categorical	National Land Cover Dataset 2001 resampled to 250m - min. value of 49 points
EVI2005097	Continuous	MODIS Enhanced vegetation index
NDV2005097	Continuous	MODIS Normalized difference vegetation index
NIR2005097	Continuous	MODIS Band 2 (Near Infrared)
RED2005097	Continuous	MODIS Band 1 (Red)

Table 1: Predictor variable

3 Examples

These examples demonstrate some of the capabilities of the **ModelMap** package by building three types models: Random Forest with continuous response; Random Forest with binary response; and, Stochastic Gradient Boosting with binary response. The continuous response variables are percent cover for two species of interest: Pinyon and Sage. The binary response variables are Presence/Absence of these same species.

Next, model validation diagnostics are performed with three techniques: an independent test set; Out Of Bag estimation; and cross-validation. Independent test set validation and cross-validation work for both Random Forest and Stochastic Gradient Boosting models. Out Of Bag estimation is only available for Random Forest models. (Note: in an actual model comparison study, rather than a package demonstration, the models would be compared with the same validation technique, rather than mixing techniques.)

Finally, spatial maps are produced by applying these models to remote sensing raster layers.

3.1 Example dataset

The dataset is from a pilot study in Nevada launched in 2004 involving acquisition and photo-interpretation of large-scale aerial photography, the Nevada Photo-Based Inventory Pilot (NPIP) (Frescino et al., 2009).

The predictor data set consists of 6 predictor variables: 5 continuous variables, and 1 categorical variable (Table 1). The predictor layers are 250-meter resolution, pixel-based raster layers including Moderate Resolution Imaging Spectro-radiometer (MODIS) satellite imagery (Justice et al., 2002), a Landsat Thematic Mapper-based, thematic layer of predicted land cover, National Land Cover Dataset (NLCD) (Homer et al., 2004), and a topographic layer of elevation from the National Elevation Dataset (Gesch et al., 2002).

The MODIS data included 250-meter, 16-day, cloud-free, composites of MODIS imagery for April 6, 2005: visible-red (RED) and near-infrared (NIR) bands and 2 vegetation indices, normalized difference vegetation index (NDVI) and enhanced vegetation index (EVI) (Huete et al., 2002). The land cover and topographic layers were 30-meter products re-sampled to 250 meter using majority and mean summaries, respectively.

The rectangular subset of Nevada chosen for these maps contains a small mountain range surrounded by plains, and was deliberately selected to lie along the edge of the study region to illustrate how **ModelMap** handles unsampled regions of a rectangle (Figure 5).

3.2 Example 1 - Random Forest - Continuous Response

Example 1 builds Random Forest models for two continuous response variables: Percent Cover for Pinyon and Percent Cover for Sage. An independent test set is used for model validation.

3.2.1 Set up

Begin by defining some of the arguments for our model.

After installing the **ModelMap** package, download the sample datasets and save them in the working directory for R.

Start by loading the **ModelMap** package.

```
R> library("ModelMap")
```

Specify model type. The choices are "RF" for Random Forest models, and "SGB" for Stochastic Gradient boosting models.

```
R> model.type <- "RF"
```

Define training and test data file names. Note that the arguments `qdata.trainfn` and `qdata.testfn` will accept both character strings giving the file names of CSV files of data, or the data itself in the form of a data frame.

```
R> qdatafn <- "VModelMapData.csv"
R> qdata.trainfn <- "VModelMapData_TRAIN.csv"
R> qdata.testfn <- "VModelMapData_TEST.csv"
```

Define the output folder.

```
R> folder <- getwd()
```

Split the data into training and test sets. In example 1, an independent test set is used for model validation diagnostics. The function `get.test()` randomly divides the original data into training and test sets. This function writes the training and test sets to the folder specified by `folder`, under the file names specified by `qdata.trainfn` and `qdata.testfn`. If the arguments `qdata.trainfn` and `qdata.testfn` are not included, filenames will be generated by appending `"_train"` and `"_test"` to `qdatafn`.

```
R> get.test(proportion.test = 0.2, qdatafn = qdatafn, seed = 42,
+     folder = folder, qdata.trainfn = qdata.trainfn, qdata.testfn = qdata.testfn)
```

Define file names to store model output. This filename will be used for saving the model itself. In addition, since we are not defining other output filenames, the names for other output files will be generated based on `MODELfn`.

```
R> MODELfn.a <- "VModelMapEx1a"
R> MODELfn.b <- "VModelMapEx1b"
```

Define the predictors and define which predictors are categorical. Example 1 uses five continuous predictors: the four predictor layers from the MODIS imagery plus the topographic elevation layer. As none of the chosen predictors are categorical set `predFactor` to `FALSE`.

```
R> predList <- c("ELEV250", "EVI2005097", "NDV2005097", "NIR2005097",
+     "RED2005097")
R> predFactor <- FALSE
```

Define the response variable, and whether it is continuous or binary.

```
R> response.name.a <- "PINYON"
R> response.name.b <- "SAGE"
R> response.type <- "continuous"
```

Define the seeds for each model.

```
R> seed.a <- 38
R> seed.b <- 39
```

Define the column that contains unique identifiers for each data point. These identifiers will be used to label the output file of observed and predicted values when running model validation.

```
R> unique.rowname <- "ID"
```

3.2.2 Model creation

Now create the models. The `model.build()` function returns the model object itself. The function also saves a text file listing the values of the arguments from the function call. This file is particularly useful when using the GUI prompts, as otherwise there would be no record of the options used for each model.

```
R> model.obj.ex1a <- model.build(model.type = model.type, qdata.trainfn = qdata.trainfn,
+     folder = folder, MODELfn = MODELfn.a, predList = predList,
+     predFactor = predFactor, response.name = response.name.a,
+     response.type = response.type, seed = seed.a)
R> model.obj.ex1b <- model.build(model.type = model.type, qdata.trainfn = qdata.trainfn,
+     folder = folder, MODELfn = MODELfn.b, predList = predList,
+     predFactor = predFactor, response.name = response.name.b,
+     response.type = response.type, seed = seed.b)
```

3.2.3 Model Diagnostics

Next make model predictions on an independent test set and run the diagnostics on these predictions. Model predictions on an independent test set are not stochastic, it is not necessary to set the seed.

The `model.diagnostics()` function returns a data frame of observed and predicted values. This data frame is also saved as a CSV file. This function also runs model diagnostics, and creates graphs and tables of the results. The graphics are saved as files of the file type specified by `device.type`.

For a continuous response model, the model validation diagnostics graphs are the variable importance plot (Figure 1 and Figure 2), and a scatter plot of observed verses predicted values, labeled with the Pearson's and Spearman's correlation coefficients and the slope and intercept of the linear regression line (Figure 3 and Figure 4). In example 1, the diagnostic plots are saved as PDF files.

These diagnostics show that while the most important predictor variables are similar for both models, the correlation coefficients are considerably higher for the Pinyon percent cover model as compared to the Sage model.

```
R> model.pred.ex1a <- model.diagnostics(model.obj = model.obj.ex1a,
+     qdata.testfn = qdata.testfn, folder = folder, MODELfn = MODELfn.a,
+     unique.rowname = unique.rowname, prediction.type = "TEST",
+     device.type = c("pdf"), cex = 1.2)
R> model.pred.ex1b <- model.diagnostics(model.obj = model.obj.ex1b,
```

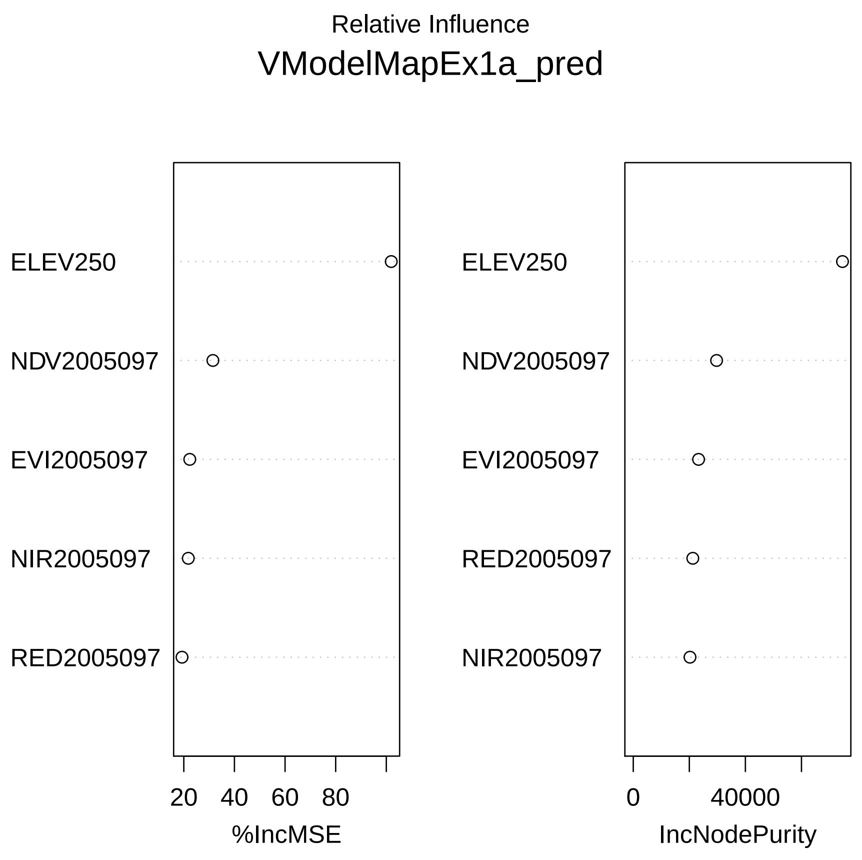

Figure 1: Example 1 - Variable importance graph for Pinyon percent cover (RF model).

```
+     qdata.testfn = qdata.testfn, folder = folder, MODELfn = MODELfn.b,
+     unique.rowname = unique.rowname, prediction.type = "TEST",
+     device.type = c("pdf"), cex = 1.2)
```

3.2.4 Map production

Before creating maps of the response variable, Examine the predictor variable for elevation (Figure 5).

```
R> elevfn <- paste(folder, "/VModelMapData_dem_ELEVM_250.img", sep = "")
R> mapgrid <- readGDAL(elevfn, band = 1)
R> mapgrid <- as.image.SpatialGridDataFrame(mapgrid)

R> opar <- par(mar = c(4, 4, 3, 6), xpd = NA, mgp = c(3, 2, 0.3))
R> col.ramp <- terrain.colors(101)
R> zlim <- c(1500, max(mapgrid$z, na.rm = TRUE))
R> legend.label <- rev(pretty(zlim, n = 5))
R> legend.colors <- col.ramp[trunc((legend.label/max(legend.label)) *
+     100) + 1]
R> legend.label <- paste(legend.label, "m", sep = "")
R> legend.label <- paste((7:3) * 500, "m")
```

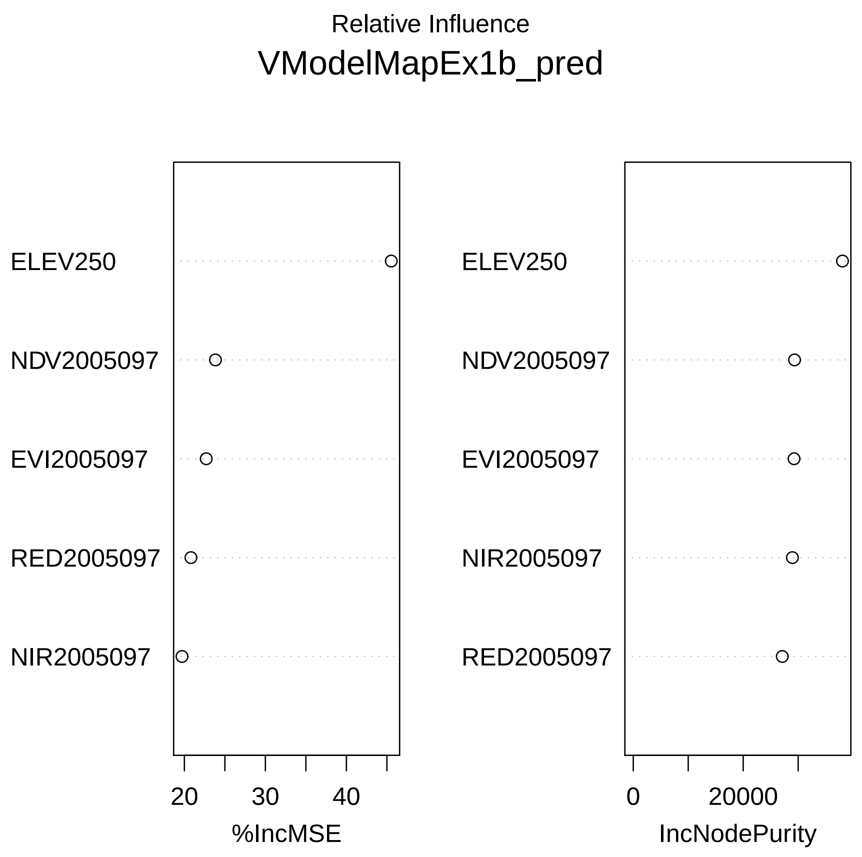

Figure 2: Example 1 - Variable importance graph for Sage percent cover (RF model).

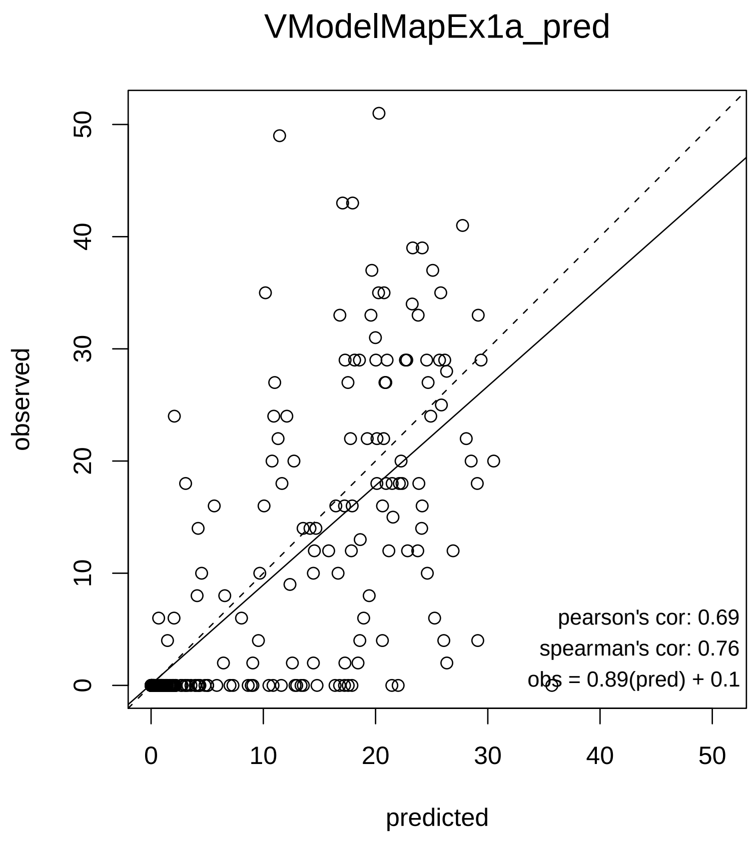

Figure 3: Example 1 - Observed verses predicted values for Pinyon percent cover (RF model).

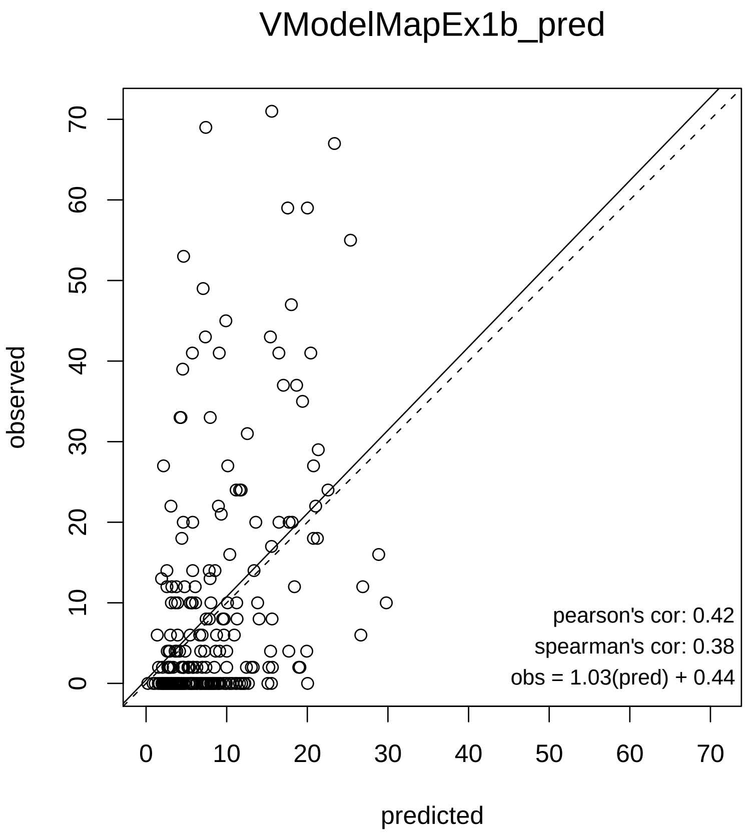

Figure 4: Example 1 - Observed verses predicted values for Sage percent cover (RF model).

```
R> legend.colors <- col.ramp[c(100, 75, 50, 25, 1)]
R> image(mapgrid, col = col.ramp, zlim = zlim, asp = 1, bty = "n")
R> legend(x = max(mapgrid$x), y = max(mapgrid$y), legend = legend.label,
+     fill = legend.colors, bty = "n", cex = 1.2)
R> mtext("Elevation of Study Region", side = 3, line = 1, cex = 1.5)
R> par(opar)
```

Run the function `model.mapmake()` to map the response variable over the study area.

The `model.mapmake()` function can extract information about the model from the `model.obj`, so it is not necessary to re-supply the arguments that define the model, such as the type of model, the predictors, etc . . . (Note: If model was created outside of **ModelMap**, it may be necessary to supply the `response.name` argument) Also, unlike model creation, map production is not stochastic, so it is not necessary to set the seed.

The `model.mapmake()` uses a look up table to associate the predictor variables with the rasters. The function argument `rastLUTfn` will accept either a file name of the CSV file containing the table, or the data frame itself.

Although in typical user applications the raster look up table must include the full path for predictor rasters, the table provided for the examples will be incomplete when initially downloaded, as the working directory of the user is unknown and will be different on every computer. This needs to be corrected by pasting the full paths to the user's working directory to the first column, using the value from `folder` defined above.

```
R> rastLUTfn <- "VModelMapData_LUT.csv"
R> rastLUTfn <- read.table(rastLUTfn, header = FALSE, sep = ",",
+     stringsAsFactors = FALSE)
R> rastLUTfn[, 1] <- paste(folder, rastLUTfn[, 1], sep = "/")
```

To produce a map from a raster larger than the memory limits of R, predictions are made on subsets of the grid. The number of rows to read in at one time is defined by the `numrows` argument. The higher the value of `numrows`, the faster the map will be produced, but the higher the memory required. The maximum value of `numrows` depends on the width of the raster. If `model.mapmake()` crashes with the warning, `"unable to assign..."` try setting `numrows` to a lower number or use the `memory.limit()` function to increase the memory limits in R.

Since this is a Random Forest model of a continuous response, the prediction at each pixel is the mean of all the trees. Therefore these individual tree predictions can also be used to map measures of uncertainty such as standard deviation and coefficient of variation for each pixel. To do so, set `map.sd = "TRUE"`. To calculate these pixel uncertainty measures, `model.map()` must keep all the individual trees in memory, so `map.sd = "TRUE"` is much more memory intensive, and the `numrows` argument may have to be set to an even lower value.

```
R> numrows = 500

R> model.mapmake(model.obj = model.obj.ex1a, folder = folder, MODELfn = MODELfn.a,
+     rastLUTfn = rastLUTfn, numrows = numrows, map.sd = TRUE)
R> model.mapmake(model.obj = model.obj.ex1b, folder = folder, MODELfn = MODELfn.b,
+     rastLUTfn = rastLUTfn, numrows = numrows, map.sd = TRUE)
```

The function `model.mapmake()` creates an ASCII grid file of map information suitable to be imported into a GIS. As this sample dataset is relatively small, we can also import it into R for display.

We need to define a color ramp. For this response variable, zero values will display as white, shading to dark green for high values.

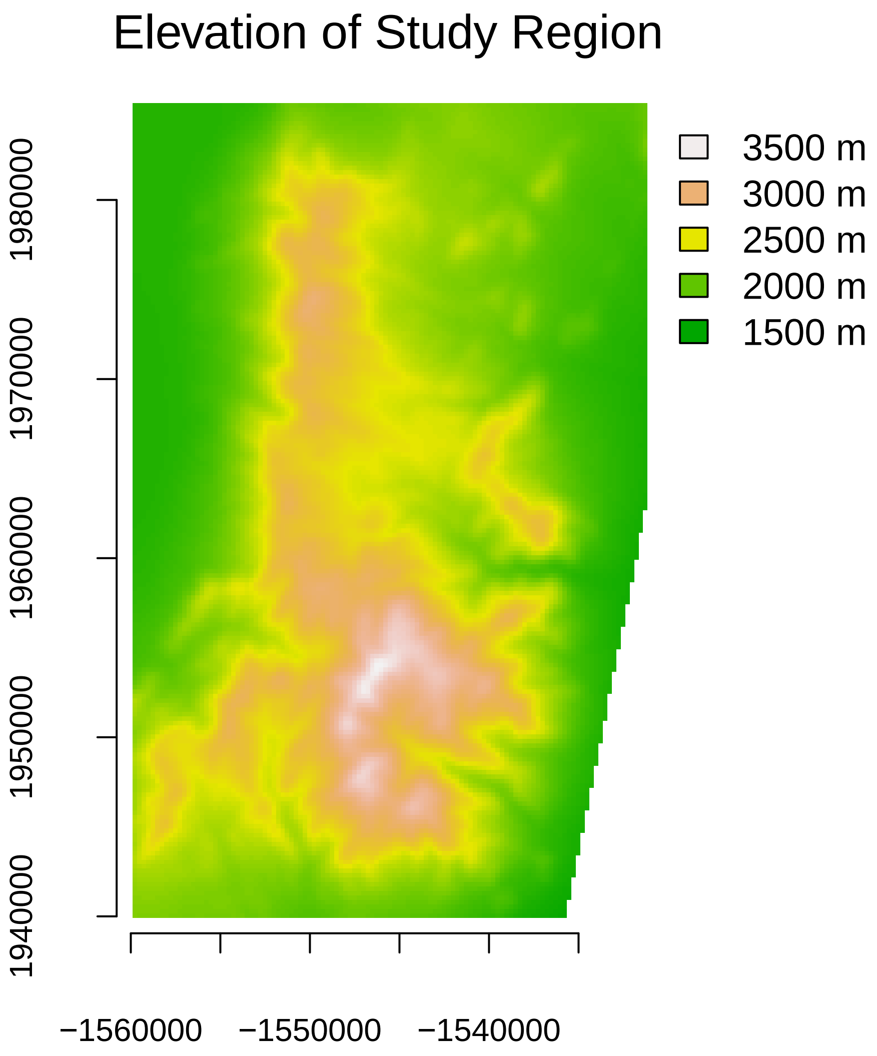

Figure 5: Elevation of study region. Projection: Universal Transverse Mercator (UTM) Zone 11, Datum: NAD83

```
R> l <- seq(100, 0, length.out = 101)
R> c <- seq(0, 100, length.out = 101)
R> col.ramp <- hcl(h = 120, c = c, l = l)
```

Next, we import the data and create the map (Figure 6). From the map, we can see that Pinyon percent cover is higher in the mountains, while Sage percent cover is higher in the foothills at the edges of the mountains.

Note that the sample map data was taken from the South Eastern edge of our study region, to illustrate how **ModelMap** deals with portions of the rectangle that fall outside of the study region. The empty wedge at lower right in the maps is the portion outside the study area. **ModelMap** uses `-9999` for unsampled data. When viewing maps in a GIS, a mask file can be used to hide unsampled regions, or other commands can be used to set the color for `-9999` values.

Since we know that percent cover can not be negative, we will set `zlim` to range from zero to the maximum value found in our map.

```
R> opar <- par(mfrow = c(1, 2), mar = c(3, 3, 2, 1), oma = c(0,
+     0, 3, 4), xpd = NA)
R> mapgrid.a <- read.asciigrid(paste(MODELfn.a, "_map.txt", sep = ""),
+     as.image = TRUE)
R> mapgrid.b <- read.asciigrid(paste(MODELfn.b, "_map.txt", sep = ""),
+     as.image = TRUE)
R> zlim <- c(0, max(mapgrid.a$z, mapgrid.b$z, na.rm = TRUE))
R> legend.label <- rev(pretty(zlim, n = 5))
R> legend.colors <- col.ramp[trunc((legend.label/max(legend.label)) *
+     100) + 1]
R> legend.label <- paste(legend.label, "%", sep = "")
R> image(mapgrid.a, col = col.ramp, zlim = zlim, asp = 1, bty = "n",
+     xaxt = "n", yaxt = "n")
R> mtext(response.name.a, side = 3, line = 1, cex = 1.2)
R> image(mapgrid.b, col = col.ramp, zlim = zlim, asp = 1, bty = "n",
+     xaxt = "n", yaxt = "n")
R> mtext(response.name.b, side = 3, line = 1, cex = 1.2)
R> legend(x = max(mapgrid.b$x), y = max(mapgrid.b$y), legend = legend.label,
+     fill = legend.colors, bty = "n", cex = 1.2)
R> mtext("Percent Cover", side = 3, line = 1, cex = 1.5, outer = T)
R> par(opar)
```

Next, we will define color ramps for the standard deviation and the coefficient of variation, and map these uncertainty measures. Often, as the mean increases, so does the standard deviation (Zar, 1996), therefore, a map of the standard deviation of the pixels (Figure 7) will look to the naked eye much like the map of the mean. However, mapping the coefficient of variation (dividing the standard deviation of each pixel by the mean of the pixel), can provide a better visualization of spatial regions of higher uncertainty (Figure 8). In this case, for Pinyon the coefficient of variation is interesting as it is higher in the plains on the upper left portion of the map, where percent cover of Pinyon is lower.

```
R> stdev.ramp <- hcl(h = 15, c = c, l = l)
```

```
R> opar <- par(mfrow = c(1, 2), mar = c(3, 3, 2, 1), oma = c(0,
+     0, 3, 4), xpd = NA)
R> mapgrid.a <- read.asciigrid(paste(MODELfn.a, "_stdev.txt", sep = ""),
+     as.image = TRUE)
R> mapgrid.b <- read.asciigrid(paste(MODELfn.b, "_stdev.txt", sep = ""),
```

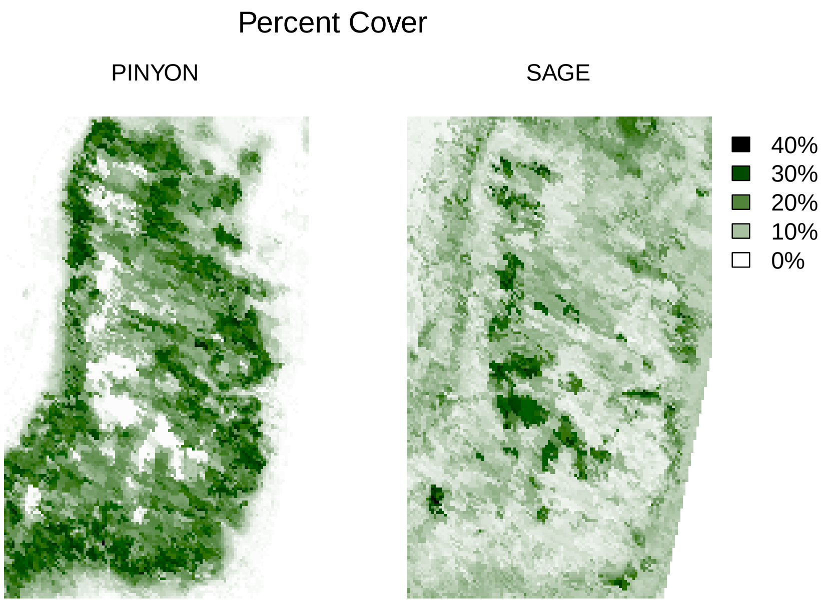

Figure 6: Example 1 - Maps of percent cover for Pinyon and Sage (RF models).

```
+     as.image = TRUE)
R> zlim <- c(0, max(mapgrid.a$z, mapgrid.b$z, na.rm = TRUE))
R> legend.label <- rev(pretty(zlim, n = 5))
R> legend.colors <- stdev.ramp[trunc((legend.label/max(legend.label)) *
+     100) + 1]
R> legend.label <- paste(legend.label, "%", sep = "")
R> image(mapgrid.a, col = stdev.ramp, zlim = zlim, asp = 1, bty = "n",
+     xaxt = "n", yaxt = "n")
R> mtext(response.name.a, side = 3, line = 1, cex = 1.2)
R> image(mapgrid.b, col = stdev.ramp, zlim = zlim, asp = 1, bty = "n",
+     xaxt = "n", yaxt = "n")
R> mtext(response.name.b, side = 3, line = 1, cex = 1.2)
R> legend(x = max(mapgrid.b$x), y = max(mapgrid.b$y), legend = legend.label,
+     fill = legend.colors, bty = "n", cex = 1.2)
R> mtext("Standard Deviation of Percent Cover", side = 3, line = 1,
+     cex = 1.5, outer = T)
R> par(opar)
```

```
R> coefv.ramp <- hcl(h = 70, c = c, l = l)
```

```
R> opar <- par(mfrow = c(1, 2), mar = c(3, 3, 2, 1), oma = c(0,
+     0, 3, 4), xpd = NA)
R> mapgrid.a <- read.asciigrid(paste(MODELfn.a, "_coefv.txt", sep = ""),
+     as.image = TRUE)
R> mapgrid.b <- read.asciigrid(paste(MODELfn.b, "_coefv.txt", sep = ""),
+     as.image = TRUE)
R> zlim <- c(0, max(mapgrid.a$z, mapgrid.b$z, na.rm = TRUE))
```

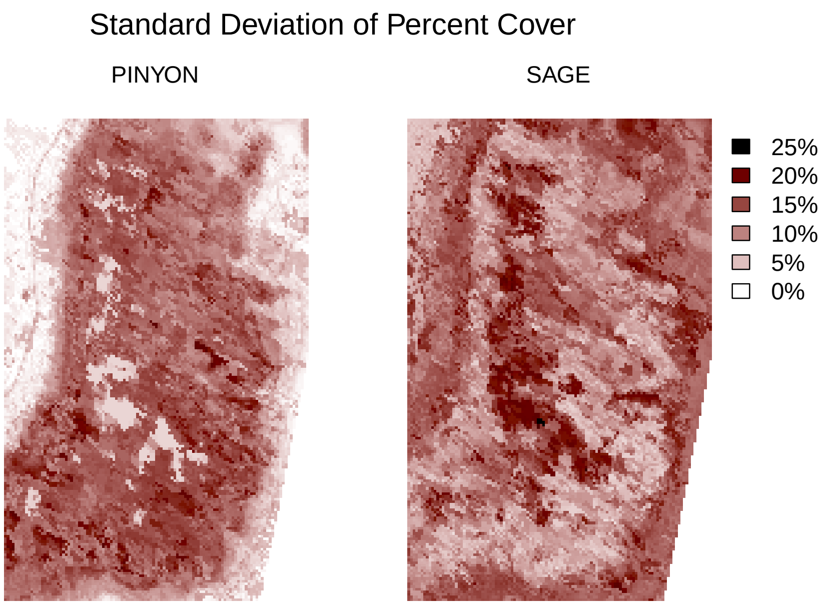

Figure 7: Example 1 - Map of standard deviation of Random Forest trees at each pixel for Pinyon and Sage (RF models).

```
R> legend.label <- rev(pretty(zlim, n = 5))
R> legend.colors <- coefv.ramp[trunc((legend.label/max(legend.label)) *
+     100) + 1]
R> image(mapgrid.a, col = coefv.ramp, zlim = zlim, asp = 1, bty = "n",
+     xaxt = "n", yaxt = "n")
R> mtext(response.name.a, side = 3, line = 1, cex = 1.2)
R> image(mapgrid.b, col = coefv.ramp, zlim = zlim, asp = 1, bty = "n",
+     xaxt = "n", yaxt = "n")
R> mtext(response.name.b, side = 3, line = 1, cex = 1.2)
R> legend(x = max(mapgrid.b$x), y = max(mapgrid.b$y), legend = legend.label,
+     fill = legend.colors, bty = "n", cex = 1.2)
R> mtext("Coefficient of Variation of Percent Cover", side = 3,
+     line = 1, cex = 1.5, outer = T)
R> par(opar)
```

3.3 Example 2 - Random Forest - Binary Response

Example 2 builds a binary response model for presence of Pinyon and Sage. A catagorical predictor is added to the model. Out-of-bag estimates are used for model validation.

3.3.1 Set up

Define model type.

```
R> model.type <- "RF"
```

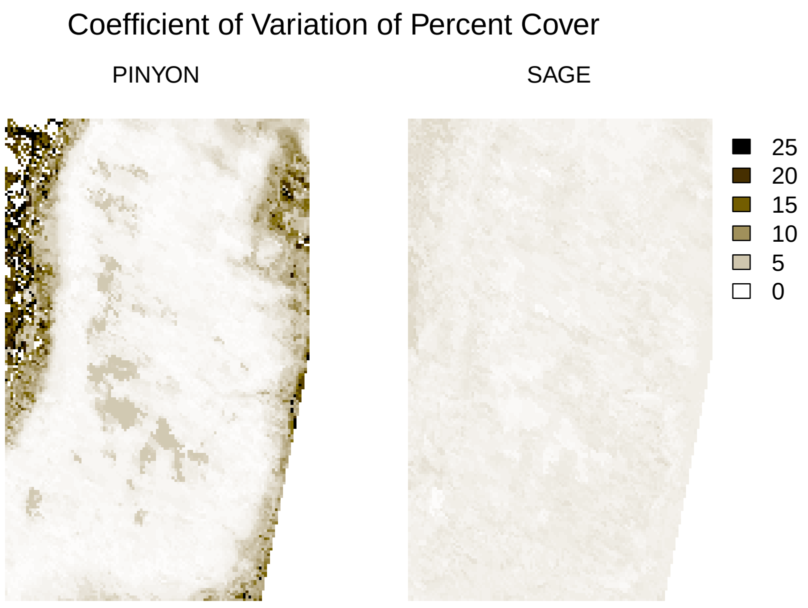

Figure 8: Example 1 - Map of coefficient of variation of Random Forest trees at each pixel for Pinyon and Sage (RF models).

Define data.

```
R> qdatafn <- "VModelMapData.csv"
```

Define `folder`.

```
R> folder <- getwd()
```

Define model filenames.

```
R> MODELfn.a <- "VModelMapEx2a"
R> MODELfn.b <- "VModelMapEx2b"
```

Define the predictors. These are the five continuous predictors from the first example, plus one categorical predictor layer, the thematic layer of predicted land cover classes from the National Land Cover Dataset. The argument `predFactor` is used to specify the categorical predictor.

```
R> predList <- c("ELEV250", "NLCD01_250", "EVI2005097", "NDV2005097",
+     "NIR2005097", "RED2005097")
R> predFactor <- c("NLCD01_250")
```

Define the data column to use as the response, and if it is continuous or binary. Since `response.type = "binary"` this variable will be automatically translated so that zeros are treated as Absent and any value greater than zero is treated as Present.

```
R> response.name.a <- "PINYON"
R> response.name.b <- "SAGE"
R> response.type <- "binary"
```

Define the seeds for each model.

```
R> seed.a <- 40
R> seed.b <- 41
```

Define the column that contains unique identifiers.

```
R> unique.rowname <- "ID"
```

Define numrows.

```
R> numrows = 500
```

Define raster look up table.

```
R> rastLUTfn <- "VModelMapData_LUT.csv"
R> rastLUTfn <- read.table(rastLUTfn, header = FALSE, sep = ",",
+     stringsAsFactors = FALSE)
R> rastLUTfn[, 1] <- paste(folder, rastLUTfn[, 1], sep = "/")
```

3.3.2 Model creation

Create the model. Because Out-Of-Bag predictions will be used for model diagnostics, the full dataset can be used as training data. To do this, set `qdata.trainfn <- qdatafn`, `qdata.testfn <- FALSE` and `v.fold = FALSE`.

```
R> model.obj.ex2a <- model.build(model.type = model.type, qdata.trainfn = qdatafn,
+     folder = folder, MODELfn = MODELfn.a, predList = predList,
+     predFactor = predFactor, response.name = response.name.a,
+     response.type = response.type, seed = seed.a)
R> model.obj.ex2b <- model.build(model.type = model.type, qdata.trainfn = qdatafn,
+     folder = folder, MODELfn = MODELfn.b, predList = predList,
+     predFactor = predFactor, response.name = response.name.b,
+     response.type = response.type, seed = seed.b)
```

3.3.3 Model Diagnostics

Make Out-Of-Bag model predictions on the training data and run the diagnostics on these predictions. This time, save JPEG, PDF, and PS versions of the diagnostic plots.

Out of Bag model predictions for a Random Forest model are not stochastic, so it is not necessary to set the seed.

Since this is a binary response model model diagnostics include ROC plots and other threshold selection plots generated by **PresenceAbsence** (Freeman, 2007; Freeman and Moisen, 2008a) (Figure 9 and Figure 10) in addition to the variable importance graph (Figure 11 and Figure 12).

For binary response models, there are also CSV files of presence-absence thresholds optimized by 12 possible criteria, along with their associated error statistics. For more details on these 12 optimization criteria see Freeman and Moisen (2008a). Some of these criteria are dependent on user selected parameters. In this example, two of these parameters are specified: required sensitivity (`req.sens`) and required specificity (`req.spec`). Other user defined parameters, such as False Positive Cost (`FPC`) and False Negative Cost (`FNC`) are left at the default values. When default values are used for these parameters, `model.diagnostics()` will give a warning. In this case:

```
1: In error.threshold.plot(PRED, opt.methods = optimal.thresholds(),  ... :
  costs assumed to be equal
```

The variable importance graphs show NLCD was a very important predictor for Pinyon presence, but not an important variable when predicting Sage presence.

```
R> model.pred.ex2a <- model.diagnostics(model.obj = model.obj.ex2a,
+     qdata.trainfn = qdatafn, folder = folder, MODELfn = MODELfn.a,
+     unique.rowname = unique.rowname, prediction.type = "OOB",
+     device.type = c("jpeg", "pdf", "postscript"), cex = 1.2)
R> model.pred.ex2b <- model.diagnostics(model.obj = model.obj.ex2b,
+     qdata.trainfn = qdatafn, folder = folder, MODELfn = MODELfn.b,
+     unique.rowname = unique.rowname, prediction.type = "OOB",
+     device.type = c("jpeg", "pdf", "postscript"), cex = 1.2)
```

Take a closer look at the text file of thresholds optimized by multiple criteria. These thresholds are used later to display the mapped predictions, so read this file into R now.

```
R> opt.thresh.a <- read.table(paste(MODELfn.a, "_pred_optthresholds.csv",
+     sep = ""), header = TRUE, sep = ",", stringsAsFactors = FALSE)
R> opt.thresh.a[, -1] <- signif(opt.thresh.a[, -1], 2)
R> opt.thresh.b <- read.table(paste(MODELfn.b, "_pred_optthresholds.csv",
+     sep = ""), header = TRUE, sep = ",", stringsAsFactors = FALSE)
R> opt.thresh.b[, -1] <- signif(opt.thresh.b[, -1], 2)
R> pred.prev.a <- read.table(paste(MODELfn.a, "_pred_prevalence.csv",
+     sep = ""), header = TRUE, sep = ",", stringsAsFactors = FALSE)
R> pred.prev.a[, -1] <- signif(pred.prev.a[, -1], 2)
R> pred.prev.b <- read.table(paste(MODELfn.b, "_pred_prevalence.csv",
+     sep = ""), header = TRUE, sep = ",", stringsAsFactors = FALSE)
R> pred.prev.b[, -1] <- signif(pred.prev.b[, -1], 2)
```

Optimized thresholds for Pinyon:

```
R> opt.thresh.a

    opt.methods threshold  PCC sensitivity specificity Kappa
1       Default      0.50 0.92        0.92        0.92  0.83
2     Sens=Spec      0.51 0.92        0.92        0.92  0.83
3  MaxSens+Spec      0.53 0.92        0.92        0.92  0.84
4      MaxKappa      0.53 0.92        0.92        0.92  0.84
5        MaxPCC      0.61 0.92        0.91        0.93  0.84
6  PredPrev=Obs      0.58 0.92        0.91        0.93  0.83
7       ObsPrev      0.46 0.91        0.92        0.91  0.83
8      MeanProb      0.47 0.91        0.92        0.91  0.83
9    MinROCdist      0.53 0.92        0.92        0.92  0.84
10      ReqSens      0.75 0.90        0.85        0.95  0.81
11      ReqSpec      0.23 0.90        0.96        0.85  0.81
12         Cost      0.59 0.92        0.91        0.93  0.83
```

And for Sage:

```
R> opt.thresh.b
```

```
    opt.methods threshold  PCC sensitivity specificity Kappa
1       Default      0.50 0.66        0.75        0.54  0.30
2     Sens=Spec      0.57 0.64        0.64        0.64  0.28
3  MaxSens+Spec      0.49 0.67        0.78        0.53  0.31
4      MaxKappa      0.49 0.67        0.78        0.53  0.31
5        MaxPCC      0.46 0.67        0.81        0.48  0.30
6  PredPrev=Obs      0.54 0.65        0.69        0.60  0.29
7       ObsPrev      0.56 0.64        0.66        0.62  0.28
8      MeanProb      0.56 0.64        0.66        0.62  0.28
9     MinROCdist     0.53 0.66        0.71        0.59  0.30
10      ReqSens      0.41 0.67        0.85        0.43  0.30
11      ReqSpec      0.76 0.55        0.32        0.85  0.16
12         Cost      0.48 0.67        0.79        0.51  0.31
```

Observed and predicted prevalence for Pinyon:

```
R> pred.prev.a
```

```
   opt.thresh.opt.methods threshold Obs.Prevalence pred
1                 Default      0.50           0.46 0.47
2               Sens=Spec      0.51           0.46 0.47
3            MaxSens+Spec      0.53           0.46 0.47
4                MaxKappa      0.53           0.46 0.47
5                  MaxPCC      0.61           0.46 0.46
6            PredPrev=Obs      0.58           0.46 0.46
7                 ObsPrev      0.46           0.46 0.47
8                MeanProb      0.47           0.46 0.47
9              MinROCdist      0.53           0.46 0.47
10                ReqSens      0.75           0.46 0.42
11                ReqSpec      0.23           0.46 0.52
12                   Cost      0.59           0.46 0.46
```

And for Sage:

```
R> pred.prev.b
```

```
   opt.thresh.opt.methods threshold Obs.Prevalence pred
1                 Default      0.50           0.56 0.63
2               Sens=Spec      0.57           0.56 0.52
3            MaxSens+Spec      0.49           0.56 0.64
4                MaxKappa      0.49           0.56 0.64
5                  MaxPCC      0.46           0.56 0.68
6            PredPrev=Obs      0.54           0.56 0.56
7                 ObsPrev      0.56           0.56 0.54
8                MeanProb      0.56           0.56 0.53
9              MinROCdist      0.53           0.56 0.58
10                ReqSens      0.41           0.56 0.73
11                ReqSpec      0.76           0.56 0.24
12                   Cost      0.48           0.56 0.66
```

The model quality graphs show that the model of Pinyon presence is much higher quality than the Sage model. This is illustrated with four plots: a histogram plot, a calibration plot, a ROC plot with it's associated Area Under the Curve (AUC), and an error rate verses threshold plot

Pinyon has a double humped histogram plot, with most of the observed presences and absences neatly divided into the two humps. Therefor the optimized threshold values fall between the two humps and neatly divide the data into absences and presences. For Sage, on the other hand, the observed presences and absences are scattered throughout the range of predicted probabilities, and so there is no single threshold that will neatly divide the data into present and absent groups. In this case, the different optimization criteria tend to be widely separated, each representing a different compromise between the error statistics (Freeman and Moisen, 2008b).

Calibration plots provide a goodness-of-fit plot for presence-absence models, as described by Pearce and Ferrier (2000), Vaughan and Ormerod (2005), and Reineking and Schröder (2006). In a Calibration plot the predicted values are divided into bins, and the observed proportion of each bin is plotted against the predicted value of the bin. For Pinyon, the standard errors for the bins overlap the diagonal, and the bins do not show a bias. For Sage, however, the error bars for the highest and lowest bins do not overlap the diagonal, and there is a bias where low probabilities tend to be over predicted, and high probabilities tend to be under predicted.

The ROC plot from a good model will rise steeply to the upper left corner then level off quickly, resulting in an AUC near 1.0. A poor model (i.e. a model that is no better than random assignment) will have a ROC plot lying along the diagonal, with an AUC near 0.5. The Area Under the Curve (AUC) is equivalent to the chance that a randomly chosen plot with an observed value of present will have a predicted probability higher than that of a randomly chosen plot with an observed value of absent. The **PresenceAbsence** package used to create the model quality graphs for binary response models uses the method from DeLong et al. (1988) to calculate Area Under the Curve (AUC). For these two models, the Area Under the Curve (AUC) for Pinyon is 0.97 and the ROC plot rises steeply, while the AUC for Sage is only 0.70, and the ROC plot is much closer to the diagonal.

In the Error Rate verses Threshold plot sensitivity, specificity and Kappa are plotted against all possible values of the threshold (Fielding and Bell, 1997). In the graph of Pinyon error rates, sensitivity and specificity cross at a higher value, and also, the error statistics show good values across a broader range of thresholds. The Kappa curve is high and flat topped, indicating that for this model, Kappa will be high across a wide range of thresholds. For Sage, sensitivity and specificity cross at a lower value, and the Kappa curve is so low that it is nearly hidden behind the graph legend. For this model even the most optimal threshold selection will still result in a relatively low Kappa value.

3.3.4 Map production

The function `model.mapmake()` creates ascii text files of map predictions.

```
R> model.mapmake(model.obj = model.obj.ex2a, folder = folder, MODELfn = MODELfn.a,
+     rastLUTfn = rastLUTfn, numrows = numrows, na.action = "na.omit")
R> model.mapmake(model.obj = model.obj.ex2b, folder = folder, MODELfn = MODELfn.b,
+     rastLUTfn = rastLUTfn, numrows = numrows, na.action = "na.omit")
```

When working with categorical predictors, sometimes there are categories in the prediction data (either the test set, or the map data) not found in the training data. In this case, there were three classes for the predictor `NLCD01_250` that were not present in the training data. With the default `na.action = "na.omit"` the `model.mapmake()` function generated the following warnings, and these pixels will show up as blank pixels in the maps.

```
2: In production.prediction(model.obj = model.obj, rastLUTfn = rastLUTfn,  :
  categorical factored predictor NLCD01_250 contains levels 41, 43, 20 not
  found in training data
3: In production.prediction(model.obj = model.obj, rastLUTfn = rastLUTfn,  :
  Returning -9999 for data points with levels not found in the training
  data
```

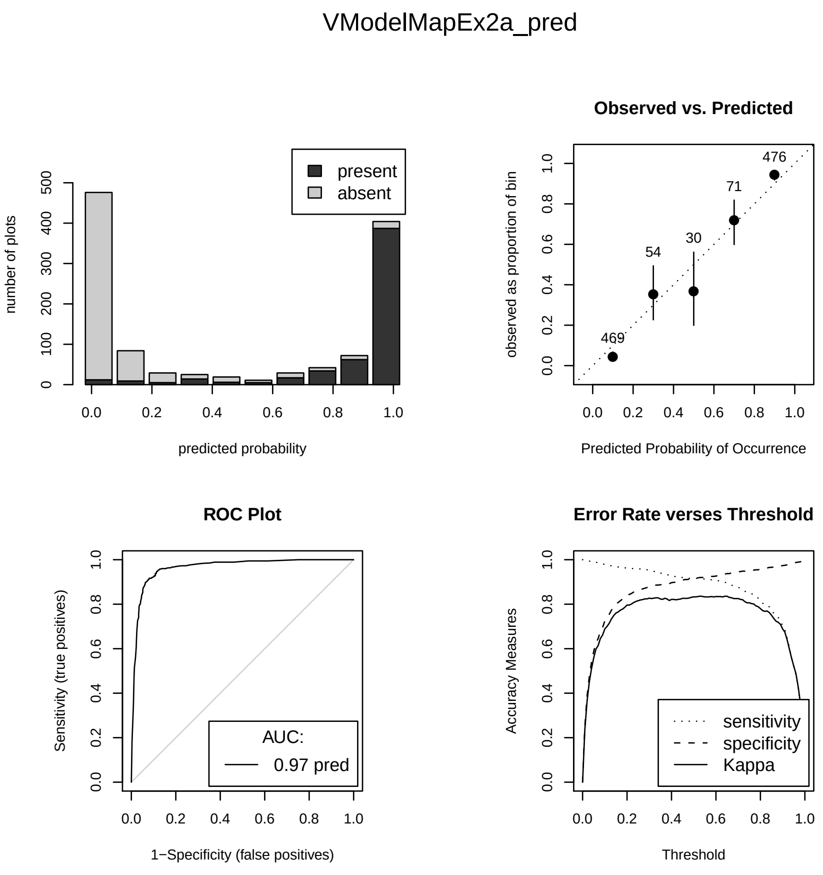

Figure 9: Example 2 - Model quality and threshold selection graphs for Pinyon presence (RF model).

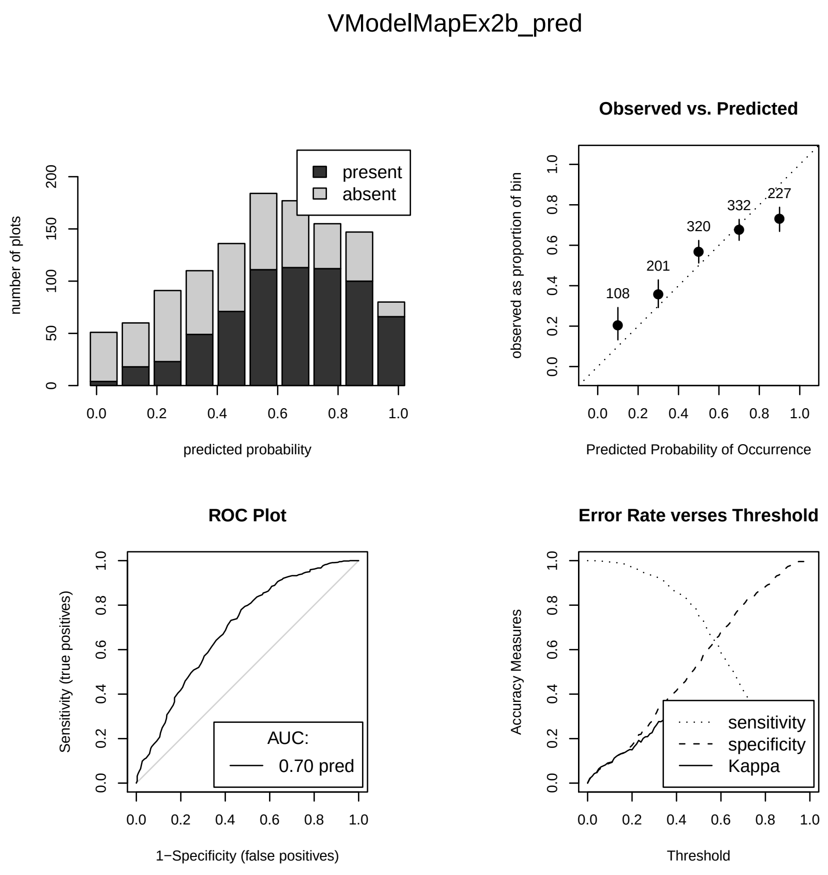

Figure 10: Example 2 - Model quality and threshold selection graphs for Sage presence (RF model).

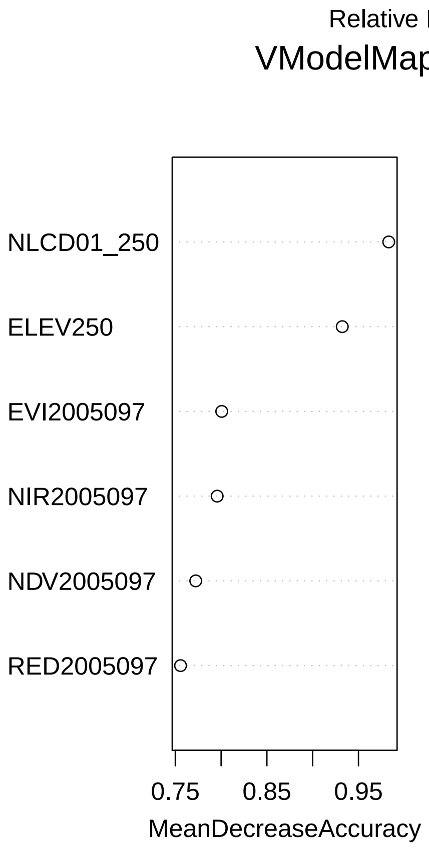
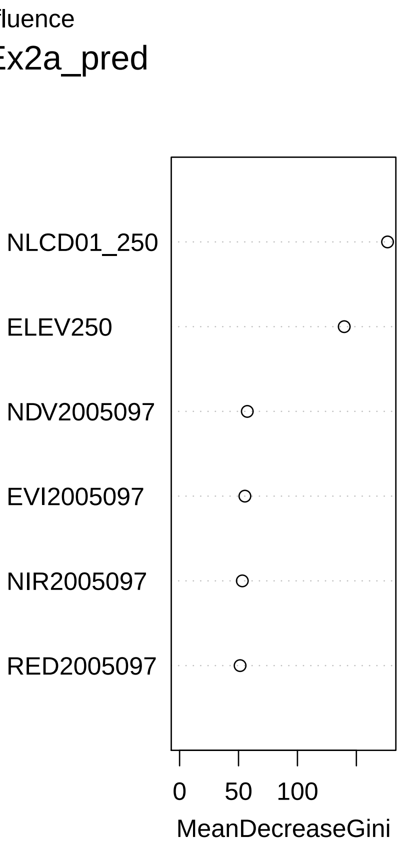

Figure 11: Example 2 - Variable importance graph for Pinyon presence (RF model).

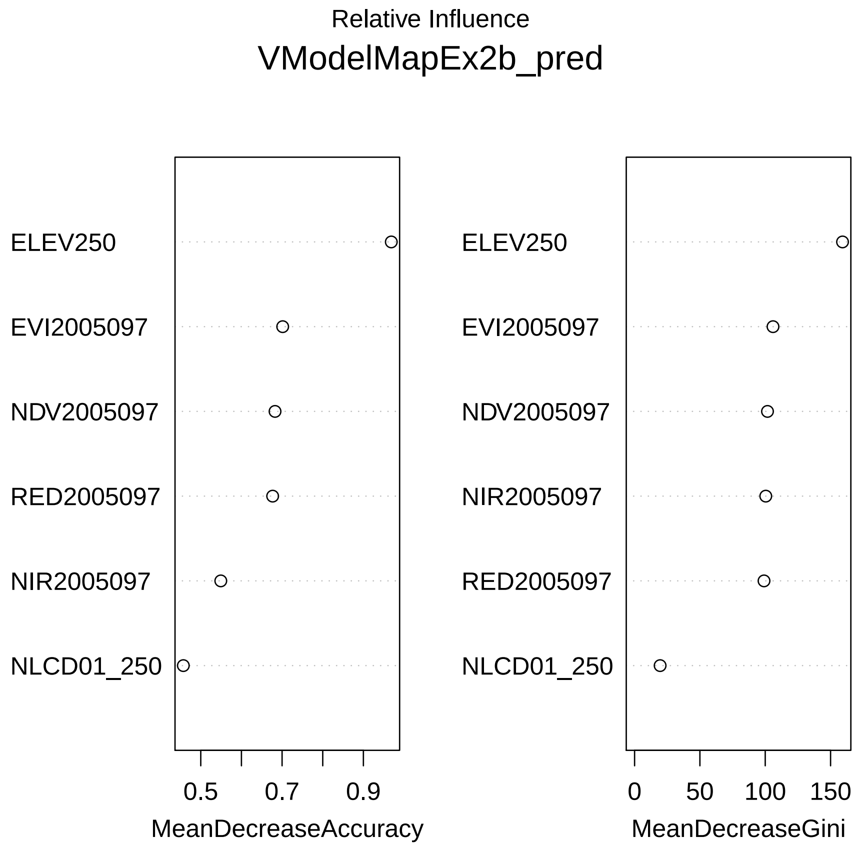

Figure 12: Example 2 - Variable importance graph for Sage presence (RF model).

Begin by mapping the probability surface, in other words, the probability that the species is present at each grid point (Figure 13).

First Define a color ramp. For this map, pixels with a high probability of presence will display as green, low probability will display as brown, and model uncertainty (probabilities near 50%) will display as yellow. Notice that the map for Pinyon, is mostly dark green and dark brown, with a thin dividing line of yellow. With a high quality model, most of the pixels are assigned high or low probabilities. The map for Sage, however, is mostly yellow, with only occasional areas of green and brown. With poor quality models, many of the pixels are inderminate, and assigned probabilities near 50%.

```
R> h = c(seq(10, 30, length.out = 10), seq(31, 40, length.out = 10),
+     seq(41, 90, length.out = 60), seq(91, 100, length.out = 10),
+     seq(101, 110, length.out = 10))
R> l = c(seq(25, 40, length.out = 10), seq(40, 90, length.out = 35),
+     seq(90, 90, length.out = 10), seq(90, 40, length.out = 35),
+     seq(40, 10, length.out = 10))
R> probpres.ramp <- hcl(h = h, c = 80, l = l)
```

Import the data and create the map. Since we know that probability of presence can range from zero to one, we will use those values for `zlim`.

```
R> opar <- par(mfrow = c(1, 2), mar = c(3, 3, 2, 1), oma = c(0,
+     0, 3, 4), xpd = NA)
R> mapgrid.a <- read.asciigrid(paste(MODELfn.a, "_map.txt", sep = ""),
+     as.image = TRUE)
R> mapgrid.b <- read.asciigrid(paste(MODELfn.b, "_map.txt", sep = ""),
+     as.image = TRUE)
R> legend.subset <- c(100, 80, 60, 40, 20, 1)
R> legend.colors <- probpres.ramp[legend.subset]
R> legend.label <- c("100%", " 80%", " 60%", " 40%", " 20%", "  0%")
R> image(mapgrid.a, col = probpres.ramp, zlim = c(0, 1), asp = 1,
+     bty = "n", xaxt = "n", yaxt = "n")
R> mtext(response.name.a, side = 3, line = 1, cex = 1.2)
R> image(mapgrid.b, col = probpres.ramp, zlim = c(0, 1), asp = 1,
+     bty = "n", xaxt = "n", yaxt = "n")
R> mtext(response.name.b, side = 3, line = 1, cex = 1.2)
R> legend(x = max(mapgrid.b$x), y = max(mapgrid.b$y), legend = legend.label,
+     fill = legend.colors, bty = "n", cex = 1.2)
R> mtext("Probability of Presence", side = 3, line = 1, cex = 1.5,
+     outer = T)
R> par(opar)
```

To translate the probability surface into a Presence-Absence map it is necessary to select a cutoff threshold. Probabilities below the selected threshold are mapped as absent while probabilities above the threshold are mapped as present. Many criteria that can be used for threshold selection, ranging from the traditional default of 50 percent, to thresholds optimized to maximize Kappa, to thresholds picked to meet certain management criteria. The choice of threshold criteria can have a dramatic effect on the final map. For further discussion on this topic see Freeman and Moisen (2008b).

Here are examples of Presence-Absence maps for Pinyon and Sage produced by four different threshold optimization criteria (Figures 14 and 15). For a high quality model, such as Pinyon, the various threshold optimization criteria tend to result in similar thresholds, and the models tend to be less sensitive to threshold choice, therefore the Presence Absence maps from the four criteria

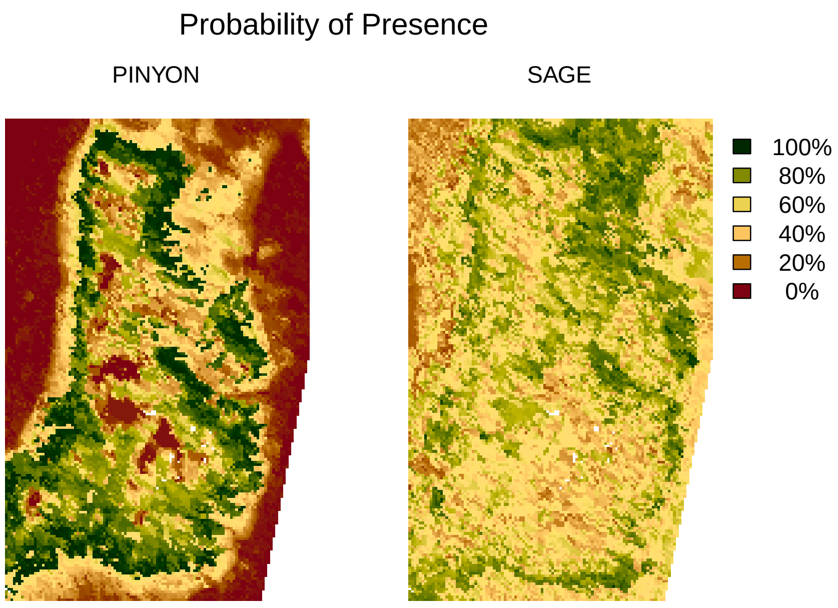

Figure 13: Example 2 - Probability surface map for presence of Pinyon and Sage (RF models).

are very similar. Poor quality models, such as this model for Sage, tend to have no single good threshold, as each criteria is represents a different compromise between errors of omission and errors of commission. It is therefore particularly important to carefully match threshold criteria to the intended use of the map.

```
R> opar <- par(mfrow = c(2, 2), mar = c(2.5, 3, 4, 1), oma = c(0,
+     0, 4, 6), xpd = NA)
R> mapgrid <- read.asciigrid(paste(MODELfn.a, "_map.txt", sep = ""),
+     as.image = TRUE)
R> criteria <- c("Default", "MaxKappa", "ReqSens", "ReqSpec")
R> criteria.labels <- c("Default", "MaxKappa", "ReqSens = 0.9",
+     "ReqSpec = 0.9")
R> for (i in 1:4) {
+     thresh <- opt.thresh.a$threshold[opt.thresh.a$opt.methods ==
+         criteria[i]]
+     presencegrid <- mapgrid
+     presencegrid$z <- ifelse(presencegrid$z > thresh, 1, 0)
+     image(presencegrid, col = c("white", "forestgreen"), zlim = c(0,
+         1), asp = 1, bty = "n", xaxt = "n", yaxt = "n")
+     if (i == 2) {
+         legend(x = max(mapgrid$x), y = max(mapgrid$y), legend = c("Present",
+             "Absent"), fill = c("forestgreen", "white"), bty = "n",
+             cex = 1.2)
+     }
+     mtext(criteria.labels[i], side = 3, line = 2, cex = 1.2)
+     mtext(paste("threshold =", thresh), side = 3, line = 0.5,
+         cex = 1)
+ }
```

```
R> mtext(MODELfn.a, side = 3, line = 0, cex = 1.2, outer = TRUE)
R> mtext(response.name.a, side = 3, line = 2, cex = 1.5, outer = TRUE)
R> par(opar)
```

```
R> opar <- par(mfrow = c(2, 2), mar = c(2.5, 3, 4, 1), oma = c(0,
+     0, 4, 6), xpd = NA)
R> mapgrid <- read.asciigrid(paste(MODELfn.b, "_map.txt", sep = ""),
+     as.image = TRUE)
R> criteria <- c("Default", "MaxKappa", "ReqSens", "ReqSpec")
R> criteria.labels <- c("Default", "MaxKappa", "ReqSens = 0.9",
+     "ReqSpec = 0.9")
R> for (i in 1:4) {
+     thresh <- opt.thresh.b$threshold[opt.thresh.b$opt.methods ==
+         criteria[i]]
+     presencegrid <- mapgrid
+     presencegrid$z <- ifelse(presencegrid$z > thresh, 1, 0)
+     image(presencegrid, col = c("white", "forestgreen"), zlim = c(0,
+         1), asp = 1, bty = "n", xaxt = "n", yaxt = "n")
+     if (i == 2) {
+         legend(x = max(mapgrid$x), y = max(mapgrid$y), legend = c("Present",
+             "Absent"), fill = c("forestgreen", "white"), bty = "n",
+             cex = 1.2)
+     }
+     mtext(criteria.labels[i], side = 3, line = 2, cex = 1.2)
+     mtext(paste("threshold =", thresh), side = 3, line = 0.5,
+         cex = 1)
+ }
R> mtext(MODELfn.b, side = 3, line = 0, cex = 1.2, outer = TRUE)
R> mtext(response.name.b, side = 3, line = 2, cex = 1.5, outer = TRUE)
R> par(opar)
```

3.4 Example 3 - Stochastic Gradient Boosting - Binary Response

Example 3 models the same data as previous examples, but this time with Stochastic Gradient Boosting. Stochastic Gradient Boosting does not have the option of out-of-bag estimates for model validation. To use all of the data for model building and avoid setting aside an independent test set use cross-validation for model validation.

(Note: in the **gbm** package, the function `gbm.perf()` offers the optional argument `method = "OOB"`, however, this argument specifies the technique to be used to estimated the best number of trees (`n.trees`), and is not a method for SGB model prediction.)

3.4.1 Set up

Define model type.

```
R> model.type <- "SGB"
```

Define data.

```
R> qdatafn <- "VModelMapData.csv"
```

Define `folder`.

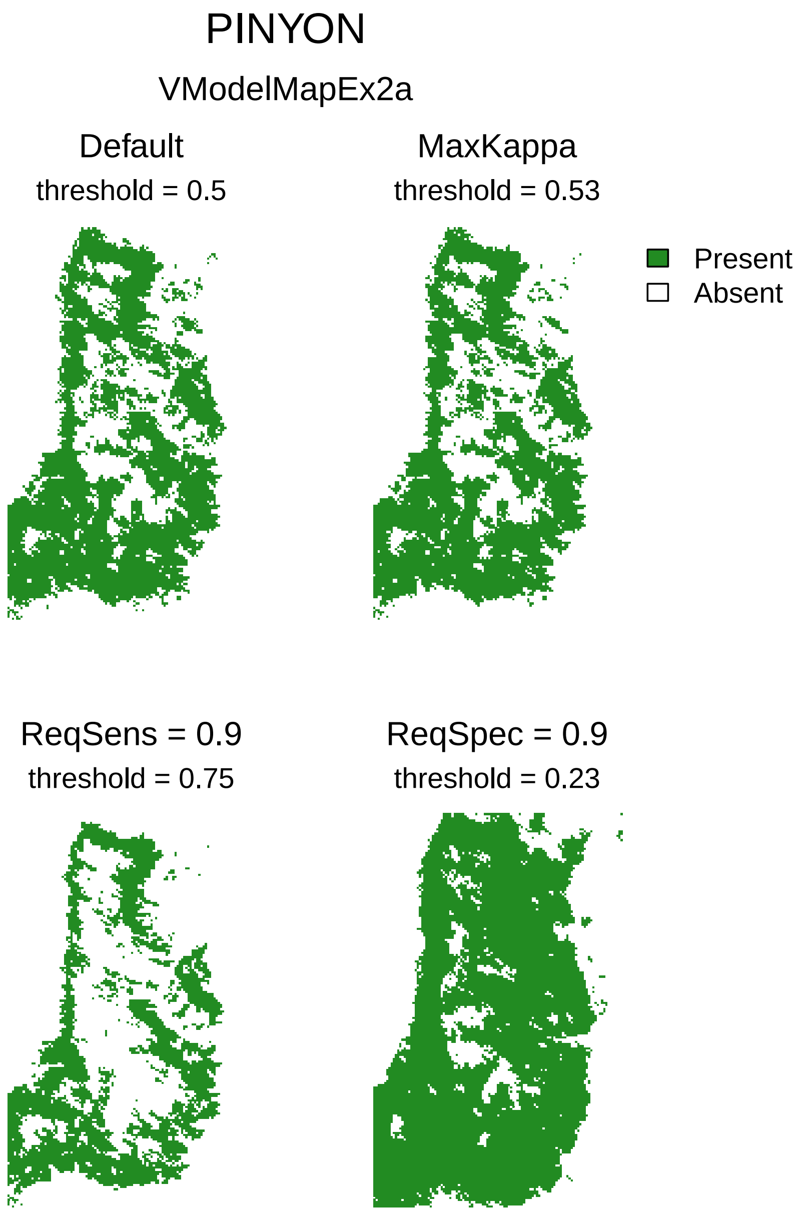

Figure 14: Example 2 - Presence-Absence maps by four different threshold selection criteria for Pinyon (RF model).

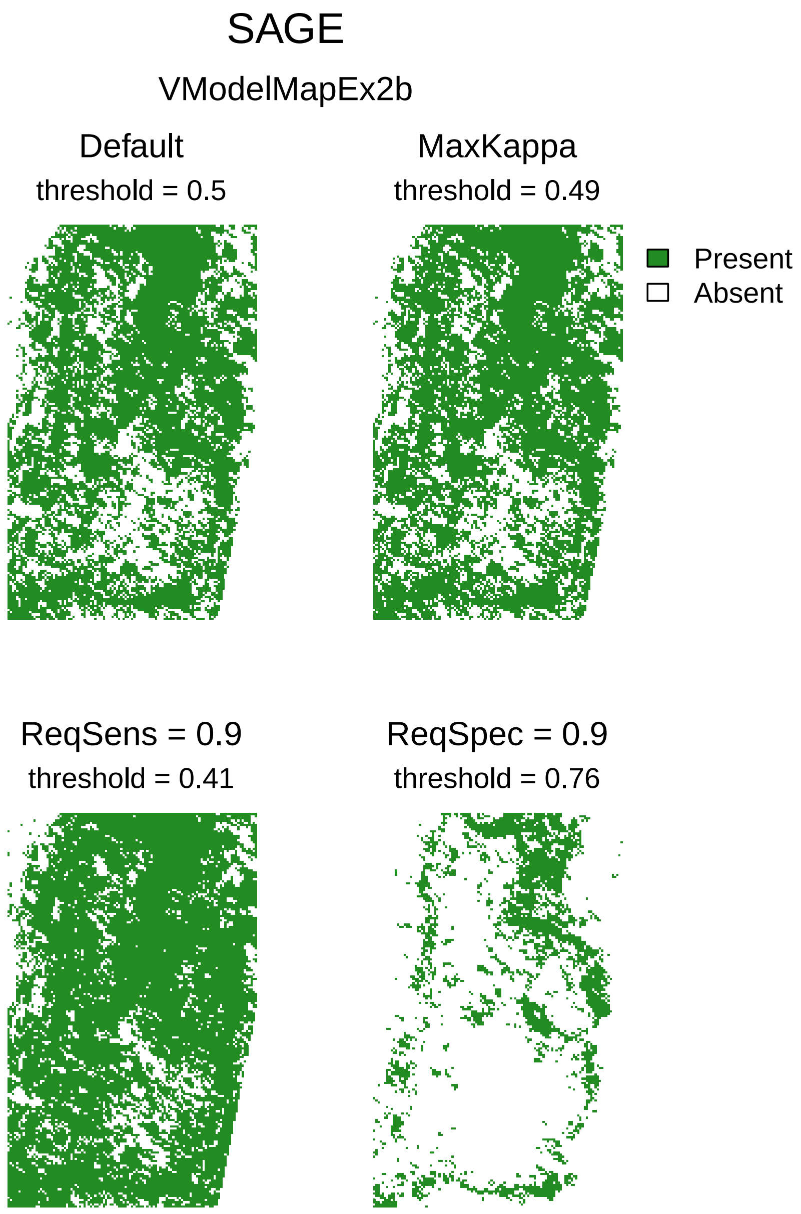

Figure 15: Example 2 - Presence-Absence maps by four different threshold selection criteria for Sage (RF model).

```
R> folder <- getwd()
```

Define model filenames.

```
R> MODELfn.a <- "VModelMapEx3a"
R> MODELfn.b <- "VModelMapEx3b"
```

Example 3 uses the same predictors as example 2.

```
R> predList <- c("ELEV250", "NLCD01_250", "EVI2005097", "NDV2005097",
+     "NIR2005097", "RED2005097")
R> predFactor <- c("NLCD01_250")
```

Define the response variable, and whether it is continuous or binary.

```
R> response.name.a <- "PINYON"
R> response.name.b <- "SAGE"
R> response.type <- "binary"
```

Define the seeds for each model.

```
R> seed.a <- 42
R> seed.b <- 43
```

Define the column that contains unique identifiers.

```
R> unique.rowname <- "ID"
```

Define numrows.

```
R> numrows = 500
```

Define raster look up table.

```
R> rastLUTfn <- "VModelMapData_LUT.csv"
R> rastLUTfn <- read.table(rastLUTfn, header = FALSE, sep = ",",
+     stringsAsFactors = FALSE)
R> rastLUTfn[, 1] <- paste(folder, rastLUTfn[, 1], sep = "/")
```

3.4.2 Model creation

Create the model and run the model validation diagnostics, this time saving JPEG, PDF, and PS versions of the diagnostic plots.

```
R> model.obj.ex3a <- model.build(model.type = model.type, qdata.trainfn = qdatafn,
+     folder = folder, MODELfn = MODELfn.a, predList = predList,
+     predFactor = predFactor, response.name = response.name.a,
+     response.type = response.type, seed = seed.a)
R> model.obj.ex3b <- model.build(model.type = model.type, qdata.trainfn = qdatafn,
+     folder = folder, MODELfn = MODELfn.b, predList = predList,
+     predFactor = predFactor, response.name = response.name.b,
+     response.type = response.type, seed = seed.b)
```

3.4.3 Model Diagnostics

Make cross validation model predictions on the training data and run the diagnostics on these predictions.

Model predictions using cross validation are stochastic, so it is necessary to set the seed.

This time, set `na.action = "na.roughfix"`. With this option, if the categorical predictor `NLCD01_250` has categories present in the validation data that were not present in the training data, the most common category from the training data will be substituted for the new, unknown category. When running cross-validation, this can be a common occurrence, especially if it is a small dataset with many categories. When this does occur, `model.map()` will generate warnings:

```
9: In production.prediction(model.obj = model.obj, rastLUTfn = rastLUTfn,  :
  categorical factored predictor NLCD01_250 contains levels 41, 43, 20 not
  found in training data
10: In production.prediction(model.obj = model.obj, rastLUTfn = rastLUTfn,  :
  Replacing categorical factored predictor levels not found in training data,
  with most common category that is found in training
```

Again, the `model.diagnostics()` function creates diagnostic graphs and saves a file of observed and predicted values. In the case of Cross Validation predictions, there is an additional column listing the assigned fold for each data point. Variable importance was almost identical for the SGB model and the RF model in Example 2. The AUC for Pinyon also very similar for the two models (0.95 for SGB verses 0.97 for RF). The SGB model for Sage, however, had a stronger AUC than the RF model (0.80 for SGB verse 0.70 for RF).

```
R> model.pred.ex3a <- model.diagnostics(model.obj = model.obj.ex3a,
+     qdata.trainfn = qdatafn, folder = folder, MODELfn = MODELfn.a,
+     unique.rowname = unique.rowname, seed = 44, prediction.type = "CV",
+     device.type = c("jpeg", "pdf", "postscript"), cex = 1.2,
+     na.action = "na.roughfix")
R> model.pred.ex3b <- model.diagnostics(model.obj = model.obj.ex3b,
+     qdata.trainfn = qdatafn, folder = folder, MODELfn = MODELfn.b,
+     unique.rowname = unique.rowname, seed = 45, prediction.type = "CV",
+     device.type = c("jpeg", "pdf", "postscript"), cex = 1.2,
+     na.action = "na.roughfix")
```

```
R> opt.thresh.a <- read.table(paste(MODELfn.a, "_pred_optthresholds.csv",
+     sep = ""), header = TRUE, sep = ",", stringsAsFactors = FALSE)
R> opt.thresh.a[, -1] <- signif(opt.thresh.a[, -1], 2)
R> opt.thresh.b <- read.table(paste(MODELfn.b, "_pred_optthresholds.csv",
+     sep = ""), header = TRUE, sep = ",", stringsAsFactors = FALSE)
R> opt.thresh.b[, -1] <- signif(opt.thresh.b[, -1], 2)
```

Here are the optimized thresholds for Pinyon:

```
R> opt.thresh.a

    opt.methods threshold  PCC sensitivity specificity Kappa
1       Default      0.50 0.92        0.92        0.92  0.83
2     Sens=Spec      0.49 0.92        0.92        0.92  0.83
3  MaxSens+Spec      0.54 0.92        0.91        0.92  0.84
4      MaxKappa      0.54 0.92        0.91        0.92  0.84
```

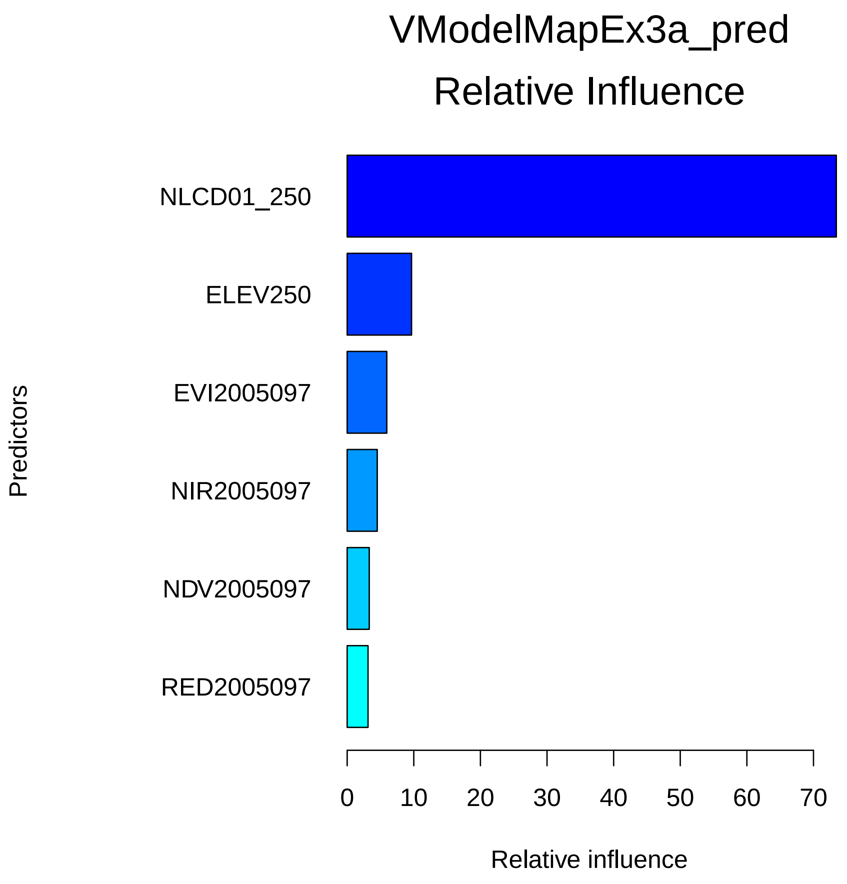

Figure 16: Example 3 - Variable importance graph for Pinyon presence (SGB model).

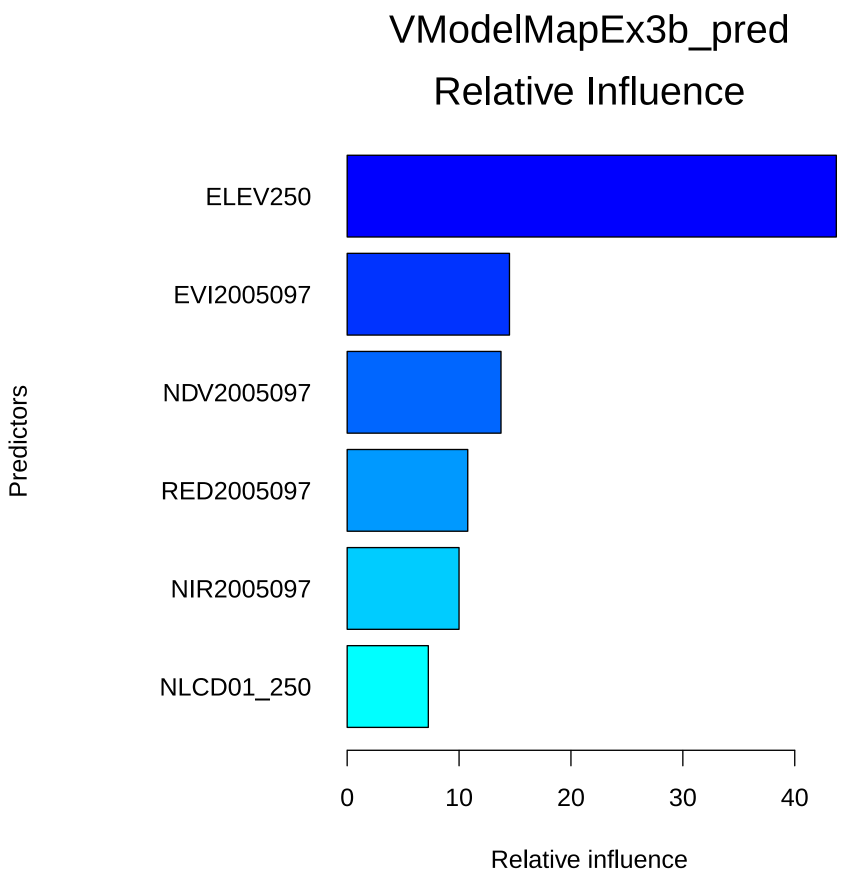

Figure 17: Example 3 - Variable importance graph for Sage presence (SGB model).

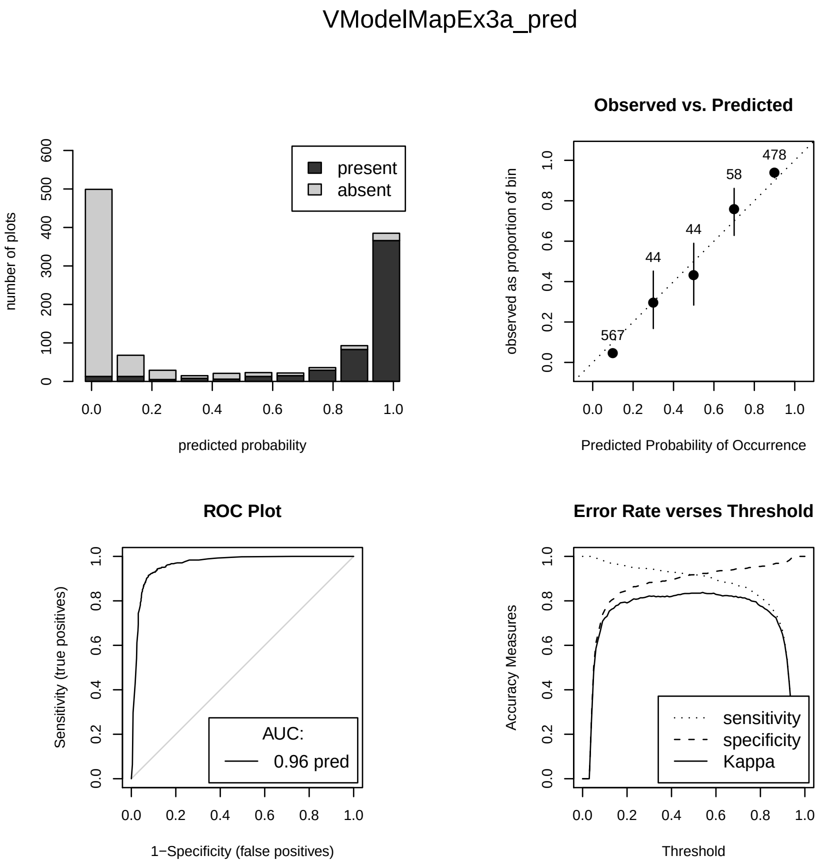

Figure 18: Example 3 - Model quality and threshold selection graphs for Pinyon presence (SGB model).

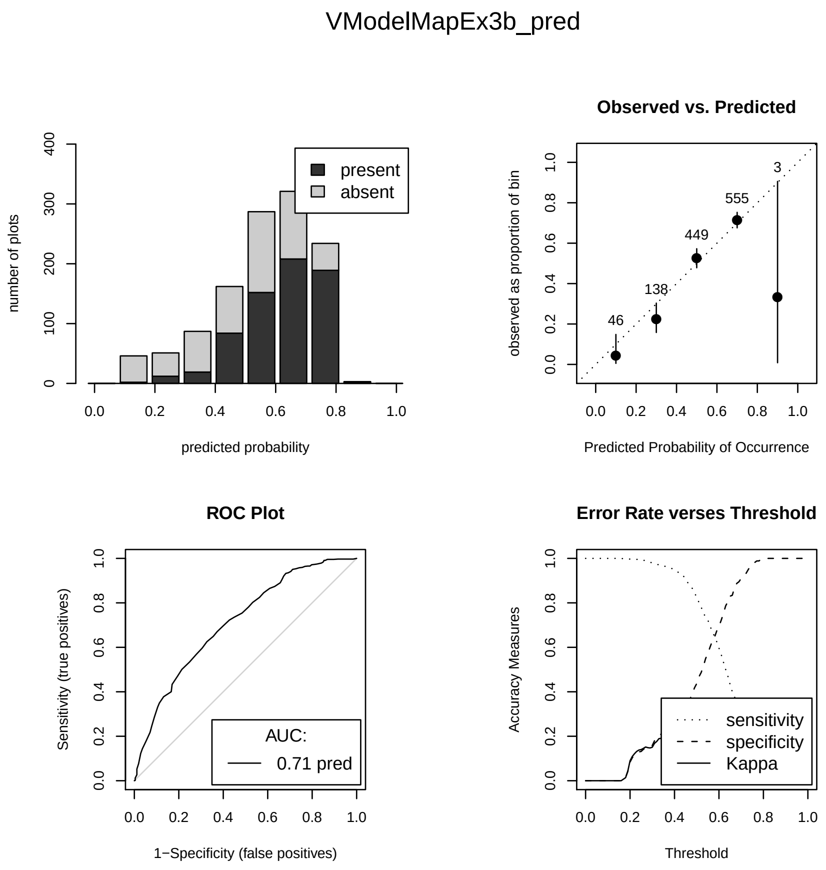

Figure 19: Example 3 - Model quality and threshold selection graphs for Sage presence (SGB model).

```
5         MaxPCC      0.54 0.92        0.91        0.92  0.84
6   PredPrev=Obs      0.55 0.92        0.91        0.92  0.83
7        ObsPrev      0.46 0.92        0.92        0.91  0.83
8       MeanProb      0.46 0.92        0.92        0.91  0.83
9     MinROCdist      0.54 0.92        0.91        0.92  0.84
10       ReqSens      0.75 0.90        0.85        0.95  0.81
11       ReqSpec      0.21 0.90        0.95        0.85  0.80
12          Cost      0.54 0.92        0.91        0.92  0.84
```

And for Sage:

```
R> opt.thresh.b

    opt.methods threshold  PCC sensitivity specificity Kappa
1       Default      0.50 0.65        0.82        0.44  0.27
2     Sens=Spec      0.58 0.65        0.65        0.65  0.29
3  MaxSens+Spec      0.59 0.65        0.63        0.67  0.29
4      MaxKappa      0.55 0.66        0.72        0.57  0.30
5        MaxPCC      0.43 0.66        0.93        0.32  0.27
6  PredPrev=Obs      0.56 0.65        0.70        0.60  0.30
7       ObsPrev      0.56 0.65        0.70        0.60  0.30
8      MeanProb      0.56 0.65        0.69        0.60  0.29
9    MinROCdist      0.59 0.65        0.63        0.67  0.29
10      ReqSens      0.48 0.66        0.87        0.39  0.27
11      ReqSpec      0.67 0.59        0.38        0.87  0.23
12         Cost      0.43 0.66        0.93        0.32  0.27
```

3.4.4 Map production

Run `model.mapmake()` to create the maps.

```
R> model.mapmake(model.obj = model.obj.ex3a, folder = folder, MODELfn = MODELfn.a,
+     rastLUTfn = rastLUTfn, map = TRUE, numrows = numrows, na.action = "na.roughfix")
R> model.mapmake(model.obj = model.obj.ex3b, folder = folder, MODELfn = MODELfn.b,
+     rastLUTfn = rastLUTfn, map = TRUE, numrows = numrows, na.action = "na.roughfix")
```

Map the probability surface (the probability that the species is present at each grid point) (Figure 20), using the color ramp defined in example 2.

```
R> h = c(seq(10, 30, length.out = 10), seq(31, 40, length.out = 10),
+     seq(41, 90, length.out = 60), seq(91, 100, length.out = 10),
+     seq(101, 110, length.out = 10))
R> l = c(seq(25, 40, length.out = 10), seq(40, 90, length.out = 35),
+     seq(90, 90, length.out = 10), seq(90, 40, length.out = 35),
+     seq(40, 10, length.out = 10))
R> probpres.ramp <- hcl(h = h, c = 80, l = l)

R> opar <- par(mfrow = c(1, 2), mar = c(3, 3, 2, 1), oma = c(0,
+     0, 3, 4), xpd = NA)
R> mapgrid.a <- read.asciigrid(paste(MODELfn.a, "_map.txt", sep = ""),
+     as.image = TRUE)
R> mapgrid.b <- read.asciigrid(paste(MODELfn.b, "_map.txt", sep = ""),
+     as.image = TRUE)
```

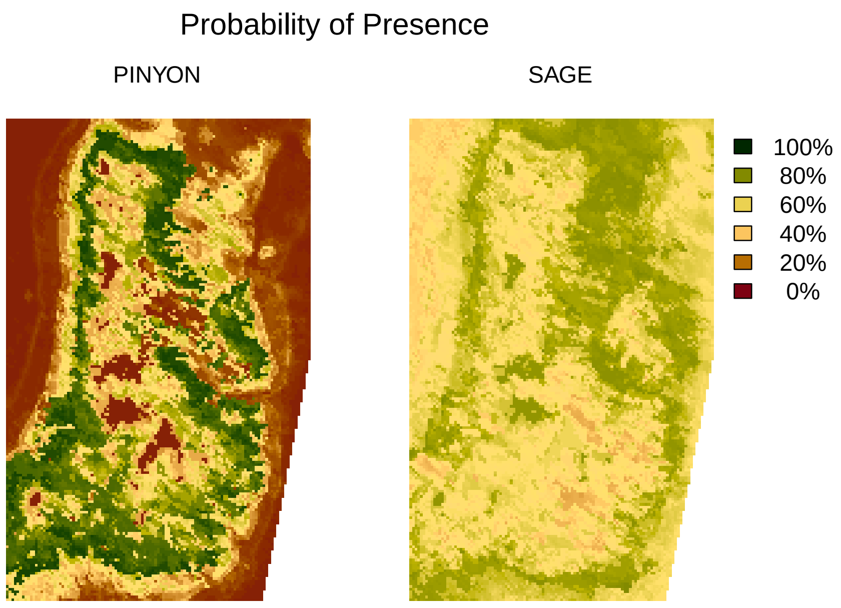

Figure 20: Example 3 - Probability surface maps for Pinyon and Sage presence (SGB models)

```
R> legend.subset <- c(100, 80, 60, 40, 20, 1)
R> legend.colors <- probpres.ramp[legend.subset]
R> legend.label <- c("100%", " 80%", " 60%", " 40%", " 20%", "  0%")
R> image(mapgrid.a, col = probpres.ramp, zlim = c(0, 1), asp = 1,
+     bty = "n", xaxt = "n", yaxt = "n")
R> mtext(response.name.a, side = 3, line = 1, cex = 1.2)
R> image(mapgrid.b, col = probpres.ramp, zlim = c(0, 1), asp = 1,
+     bty = "n", xaxt = "n", yaxt = "n")
R> mtext(response.name.b, side = 3, line = 1, cex = 1.2)
R> legend(x = max(mapgrid.b$x), y = max(mapgrid.b$y), legend = legend.label,
+     fill = legend.colors, bty = "n", cex = 1.2)
R> mtext("Probability of Presence", side = 3, line = 1, cex = 1.5,
+     outer = T)
R> par(opar)
```

4 Conclusion

In summary, the **ModelMap** software package for R creates sophisticated models from training data and validates the models with an independent test set, cross-validation, or in the case of Random Forest Models, with out-of-bag (OOB) predictions on the training data. It creates graphs and tables of the model diagnostics. It applies these models to GIS image files of predictors to create detailed prediction surfaces. It will handle large predictor files for map making, by reading in the GIS data in sections, and output the prediction for each of these sections, before reading the next section.

Appendices

A Argument List

Model building Arguments	
`model.type`	Model type: `"RF"` or `"SGB"`.
`qdata.trainfn`	Filename of the training data file for building model.
`folder`	Folder for all output.
`MODELfn`	Filename to save model object.
`predList`	Predictor short names used to build the model.
`predFactor`	Predictor short names from `predList` that are factors (i.e categorical predictors).
`response.name`	Response variable used to build the model.
`response.type`	Response type: `"binary"` or `"continuous"`.
`seed`	Seed to initialize randomization to build RF or SGB models.
`na.action`	Specifies the action to take if there are `NA` values in the prediction data
Random Forest Models:	
`ntree`	Number of random forest trees.
`mtry`	Number of variables to try at each node of Random Forest trees.
Stochastic Gradient Boosting Models:	
`n.trees`	Total number of stochastic gradient boosting trees for an SGB model. The `gbm` function `gbm.perf(method="OOB")` will be used to select the best number of trees from this total number of trees.
`shrinkage`	Shrinkage parameter applied to each tree in the expansion. Also known as the learning rate or step-size reduction.
`interaction.depth`	Maximum depth of variable interactions. `interaction.depth = 1` implies an additive model, `interaction.depth = 2` implies a model with up to 2-way interactions, etc...
`bag.fraction`	Fraction of the training set observations randomly selected to propose the next tree in the expansion. If `bag.fraction < 1` then running the same model twice will result in similar but different fits.
`train.fraction`	Fraction of observations used to fit the model with the remainder are used for computing out-of-sample estimates of the loss function. Not needed if `bag.fraction < 1`.
`n.minobsinnode`	Minimum number of observations in the trees terminal nodes. Note that this is the actual number of observations not the total weight.

Model Diagnostics Arguments	
`model.obj`	The model object to use for prediction, if the model has been previously created.
`qdata.trainfn`	Filename of the training data file for building model.
`qdata.testfn`	Filename of independent data set for testing (validating) model.
`folder`	Folder for all output.
`MODELfn`	Filename to save model object.
`response.name`	Response variable used to build the model.
`unique.rowname`	Name of column in training and test that uniquely identifies each row .
`seed`	Seed to initialize randomization to build RF or SGB models.
`prediction.type`	Type of prediction to use for model validation: `"TEST"`, `"CV"`, `"OOB"` or `"TRAIN"`
`MODELpredfn`	Filename for output of validation prediction `*.csv` file.
`na.action`	Specifies the action to take if there are `NA` values in the prediction data or if there is a level or class of a categorical predictor variable in the validation test set or the mapping data set, but not in the training data set.
`v.fold`	The number of cross-validation folds.
`device.type`	Vector of one or more device types for graphical output: `"default"`, `"jpeg"`, `"pdf"`, `"postscript"`, `"win.metafile"`. `"default"` refers to the default graphics device for your computer
`DIAGNOSTICfn`	Filename for output files from model validation diagnostics.
`jpeg.res`	Pixels per inch for jpeg output.
`device.width`	Device width for diagnostic plots in inches.
`device.height`	Device height for diagnostic plots in inches.
`cex`	Cex for diagnostic plots.
`req.sens`	Required sensitivity for threshold optimization for binary response model.
`req.spec`	Required specificity for threshold optimization for binary response model.
`FPC`	False Positive Cost for threshold optimization for binary response model.
`FNC`	False Negative Cost for threshold optimization for binary response model.

Map Production Arguments	
`model.obj`	The model object to use for prediction, if the model has been previously created.
`folder`	Folder for all output.
`MODELfn`	Filename to save model object.
`rastLUTfn`	Filename of `.csv` file for a raster look up table.
`na.action`	Specifies the action to take if there are `NA` values in the prediction data or if there is a level or class of a categorical predictor variable in the validation test set or the mapping data set, but not in the training data set.
`numrows`	Number of rows of rasters to import for each loop.
`map.sd`	Should maps of mean, standard deviation, and coefficient of variation of the predictions be produced: `T` or `F`. Only used if `response.type = "continuous"`.
`asciifn`	Filename of output file for map production. If `NULL`, `"`*`modelfn`*`_map.txt"`.
`asciifn.mean`	Filename of output file for mean of trees. If `NULL`, `"`*`modelfn`*`_map_mean.txt"`.
`asciifn.stdev`	Filename of output file for standard deviation of trees. If `NULL`, `"`*`modelfn`*`_map_stdev.txt"`.
`asciifn.coefv`	Filename of output file for coefficient of variation of trees. If `NULL`, `"`*`modelfn`*`_map_coefv.txt"`.

References

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

L. Breiman, R. A. Friedman, R. A. Olshen, and C. G. Stone. *Classification and Regression Trees.* Wadsworth, 1984.

G. De'ath and K. E. Fabricius. Classification and regression trees: a powerful yet simple technique for ecological data analysis. *Ecology*, 81:3178–3192, 2000.

E. R. DeLong, D. M. Delong, and D. L. Clarke-Pearson. Comparing areas under two or more correlated receiver operating characteristic curves: A nonparametric approach. *Biometrics*, 44(3):387–394, 1988.

J. Elith, J. R. Leathwick, and T. Hastie. A working guide to boosted regression trees. *Journal of Animal Ecology*, 77:802–813, 2008.

J. S. Evans and S. A. Cushman. Gradient modeling of conifer species using random forests. *Landscape Ecology*, 24(5):673–683, 2009.

A. H. Fielding and J. F. Bell. A review of methods for the assessment of prediction errors in conservation presence/absence models. *Environmental Conservation*, 24(1):38–49, 1997.

E. Freeman. ***PresenceAbsence**: An R Package for Presence-Absence Model Evaluation.* USDA Forest Service, Rocky Mountain Research Station, 507 25th street, Ogden, UT, USA, 2007. URL `http://CRAN.R-project.org/`. eafreeman@fs.fed.us.

E. Freeman. ***ModelMap**: An R Package for Modeling and Map production using Random Forest and Stochastic Gradient Boosting.* USDA Forest Service, Rocky Mountain Research Station, 507 25th street, Ogden, UT, USA, 2009. URL `http://CRAN.R-project.org/`. eafreeman@fs.fed.us.

E. A. Freeman and G. Moisen. Presenceabsence: An r package for presence absence analysis. *Journal of Statistical Software*, 23(11):1–31, 2008a. URL `http://www.jstatsoft.org/v23/i11`.

E. A. Freeman and G. G. Moisen. A comparison of the performance of threshold criteria for binary classification in terms of predicted prevalence and kappa. *Ecological Modelling*, 217: 48–58, 2008b.

T. S. Frescino, G. G. Moisen, K. A. Megown, V. J. Nelson, Elizabeth, Freeman, P. L. Patterson, M. Finco, K. Brewer, and J. Menlove. Nevada photo-based inventory pilot(npip) photo sampling procedures. Gen. Tech. Rep. RMRSGTR-222, U.S. Departmentof Agriculture, Forest Service, Rocky Mountain Research Station., Fort Collins, CO, 2009.

J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001.

J. H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4): 367–378, 2002.

D. Gesch, M. Oimoen, S. Greenlee, C. Nelson, M. Steuck, and D. Tyler. The national elevation dataset. photogrammetric engineering and remote sensing. *Photogrammetric Engineering and Remote Sensing*, 68:5–11, 2002.

C. Homer, C. Huang, L. Yang, B. Wylie, and M. Coan. Development of a 2001 national land-cover database for the united states. *Photogrammetric Engineering and Remote Sensing*, 70:829–840, 2004.

A. Huete, K. Didan, T. Miura, E. P. Rodriguez, X. Gao, and L. G. Ferreira. Overview of the radiometric and biophysical performance of the modis vegetation indices. *Remote Sensing of Environment*, 83:195–213, 2002.

C. O. Justice, J. R. G. Townshend, E. F. Vermote, E. Masuoka, R. E. Wolfe, N. Saleous, D. P. Roy, and J. T. Morisette. An overview of modis land data processing and product status. *Remote Sensing of Environment*, 83:3–15, 2002.

A. Liaw and M. Wiener. Classification and regression by **randomForest**. *R News*, 2(3):18–22, 2002. URL `http://CRAN.R-project.org/doc/Rnews/`.

Y. Lin and Y. Jeon. Random forest and adaptive nearest neighbors. Technical Report 1055, Department of Statistics, University of Wisconsin, 1210 West Dayton St., Madison, WI 53706, 2002.

G. G. Moisen. Classification and regression trees. In S. E. Jørgensen and B. D. Fath, editors, *Encyclopedia of Ecology*, volume 1, pages 582–588. Elsevier, 2008.

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

J. Pearce and S. Ferrier. Evaluating the predicting performance of habitat models developed using logistic regression. *Ecological Modelling*, 133:225–245, 2000.

E. J. Pebesma and R. S. Bivand. Classes and methods for spatial data in R. *R News*, 5(2):9–13, November 2005. URL `http://CRAN.R-project.org/doc/Rnews/`.

R Development Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2008. URL `http://www.R-project.org`. ISBN 3-900051-07-0.

B. Reineking and B. Schröder. Constrain to perform: Regularization of habitat models. *Ecological Modelling*, 193:675–690, 2006.

G. Ridgeway. The state of boosting. *Computing Science and Statistics*, 31:172–181, 2002.

G. Ridgeway. ***gbm**: Generalized Boosted Regression Models*, 2007. URL `http://www.i-pensieri.com/gregr/gbm.shtml`. R package version 1.6-3.

C. Strobl, A.-L. Boulesteix, A. Zeileis, and T. Hothorn. Bias in random forest variable importance measures: Illustrations, sources and a solution. *Bioinformatics*, 8:25, 2007.

I. P. Vaughan and S. J. Ormerod. The continuing challenges of testing species distribution models. *Journal of Applied Ecology*, 42:720–730, 2005.

M. P. Vayssieres, R. P. Plant, and B. H. Allen-Diaz. Classification trees: An alternative non-parametric approach for predicting species distributions. *Journal of vegetation science*, 11: 679–694, 2000.

J. H. Zar. *Biostatistical Analysis.* Prentice Hall, 1996.