ModelMap: an R Package for Model Creation and Map Production

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

September 10, 2018

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 **raster** package to read and predict over GIS raster data. Large maps are read in by row, to keep memory usage reasonable.

The current implementation of **ModelMap** builds predictive models using Random Forests, Quantile Regression Forests, and Conditional Inference Forests. Stochastic Gradient Boosting models are not currently supported. Random Forest models are constructed using the **randomForest** package (Liaw and Wiener, 2002). For more information on Quantile Regression Forests and Conditional Inference Forests see the additional vignette, "'Pick Your Flavor of Random Forest". The **ModelMap** package models both continuous and binary response variables. For binary response, the **PresenceAbsence** package (Freeman, 2007) package is used for model diagnostics.

Random Forest 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. 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 basic model validation diagnostics. The functions `model.importance.plot()` and `model.interaction.plot` provide additional graphical tools to examine the relationships between the predictor variable. 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. The **raster** package is used to read and write to the GIS image files.

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 test set or 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 mising data 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` in the training or test data. In Imagine image files, pixels 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. For model diagnostics, there are 2 options: (1) `na.action = "na.omit"` (the default) where any data point or pixel with any `NA` predictors is omitted from the model building process and the diagnostic predictions, or returned as `-9999` in the map predictions; (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. Currently, for map making only one option is available: `na.action = "na.omit"`.

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. For model diagnostics, with `na.action = "na.roughfix"` the most common category will be substituted for the unknown category. Again, for map making `na.action = "na.omit"` is the only available option. 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 omitting 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 models and 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 `NA`.

Note: in Imagine image files, if the specified `NODATA` is set as `-9999`, any `-9999` pixels will be read into R as `NA`.

2.6 The Model Object

The models built by the **ModelMap** package 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 has relatively few user set parameters and is not very 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 currently disabled in **ModelMap**.

2.7 Spatial Raster Layers

The **ModelMap** uses the **raster** 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 be a file type recognizeable by the **raster** package, for example 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.

To speed up processing of predictor layers, `model.mapmake()` builds a single raster brick object containing one layer for each predictor in the model. By default, this brick is stored as a temp file and deleted once the map is complete. If the argument `keep.predictor.brick = TRUE` then the brick will be saved in a native **raster** package format file, with the file name constructed by appending `'_brick.grd'` to the `OUTPUTfn`.

The function `model.mapmake()` by default outputs an ERDAS Imagine image file of map information suitable to be imported into a GIS. (Note: The file extension of the `OUTPPUTfn` argument can be used to specify other file types, see the help file for the `writeFormats()` function in the **raster** package for a list of possible file types and extensions.) Maps can then be imported back into R and view graphically using the **raster** package.

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 Imagine image files directly, (including multi band images). **ModelMap** also will verify that the extent of all rasters is identical 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.

3 Examples

These examples demonstrate some of the capabilities of the **ModelMap** package by building three Random Forest models: continuous response, binary response and categorical 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.The categorical response is vegetation category.

Next, model validation diagnostics are performed with three techniques: an independent test set, Out Of Bag estimation, and cross-validation. 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.

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 variables

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 data files for these examples are included in the **ModelMap** package instalation in the R library dirrectory. The datasets are under the 'external' then under 'vignette-examples'.

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 continuous response variables are percent cover of Pinyon and Sage. The binary response variables are presence of Pinyon and Sage. The categorical response variable is the vegetation category: TREE, SHRUB, OTHERVEG, and NONVEG.

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 13).

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

After installing the **ModelMap** package, find the sample datasets from the R istallation and copy them to your working directory. The data consists of five files and is located in the vignette directory of **ModelMap**, for example, in `C:\R\R-2.15.0\library\ModelMap\vignettes`.

There are 5 files:

VModelMapData.csv
VModelMapData_LUT.csv
VModelMapData_dem_ELEVM_250.img
VModelMapData_modis_STK2005097.img
VModelMapData_nlcd_NLCD01_250.img

Load the **ModelMap** package.

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

Next define some of the arguments for the models.

Specify model type. The choices are `"RF"` for Random Forest models, `"QRF"` for quantile regression forest models, and `"CF"` for conditional inference forest models. See the "Pick your flavor of Random Forest" vignette for further information on QRF and CF models.

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

Define training and test data file names. Note that the arguments `qdata.trainfn` and `qdata.testfn` will accept either 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, binary or categorical.

```
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,
                                  unique.rowname=unique.rowname,
                                  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,
                                  unique.rowname=unique.rowname,
                                  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).

For Random forest models, the model diagnostic graphs also include the out of bag model error as a function of the number of trees(Figure 5 and Figure 6)

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,
                                 # Model Validation Arguments
                                         prediction.type="TEST",
                                         device.type=c("pdf"),
                                         cex=1.2)
R> model.pred.ex1b <- model.diagnostics( model.obj=model.obj.ex1b,
                                         qdata.testfn=qdata.testfn,
                                         folder=folder,
                                         MODELfn=MODELfn.b,
                                         unique.rowname=unique.rowname,
                                  # Model Validation Arguments
                                         prediction.type="TEST",
                                         device.type=c("pdf"),
                                         cex=1.2)
```

3.2.4 Comparing Variable Importance

The `model.importance.plot()` function uses a back-to-back barplot to compare variable importance between two models built from the same predictor variables (Figure 7). Variable Importance is calculated in various ways depending on the model type, the response type and the importance type. Importance measures are summarized in (Table 2).

```
R> model.importance.plot( model.obj.1=model.obj.ex1a,
                          model.obj.2=model.obj.ex1b,
                          model.name.1="Pinyon",
                          model.name.2="Sage",
                          sort.by="predList",
                          predList=predList,
```

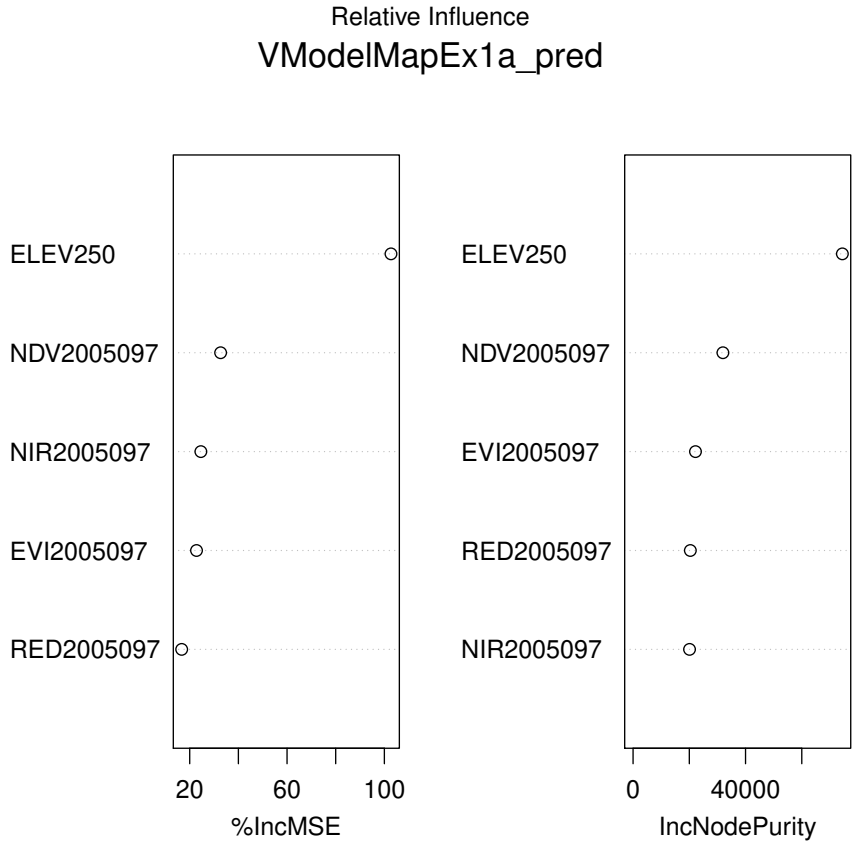

Figure 1: Example 1 - Variable importance graph for Pinyon percent cover (RF model).

Model	Response	Importance Type	Measured By
RF	continuous	1 - permutation	Percent Increase in MSE
RF	binary	1 - permutation	Mean Decrease in Accuracy
RF	categorical	1 - permutation	Mean Decrease in Accuracy
RF	continuous	2 - node impurity	Residual Sum of Squares
RF	binary	2 - node impurity	Mean Decrease in Gini
RF	categorical	2 - node impurity	Mean Decrease in Gini

Table 2: Importance Measures

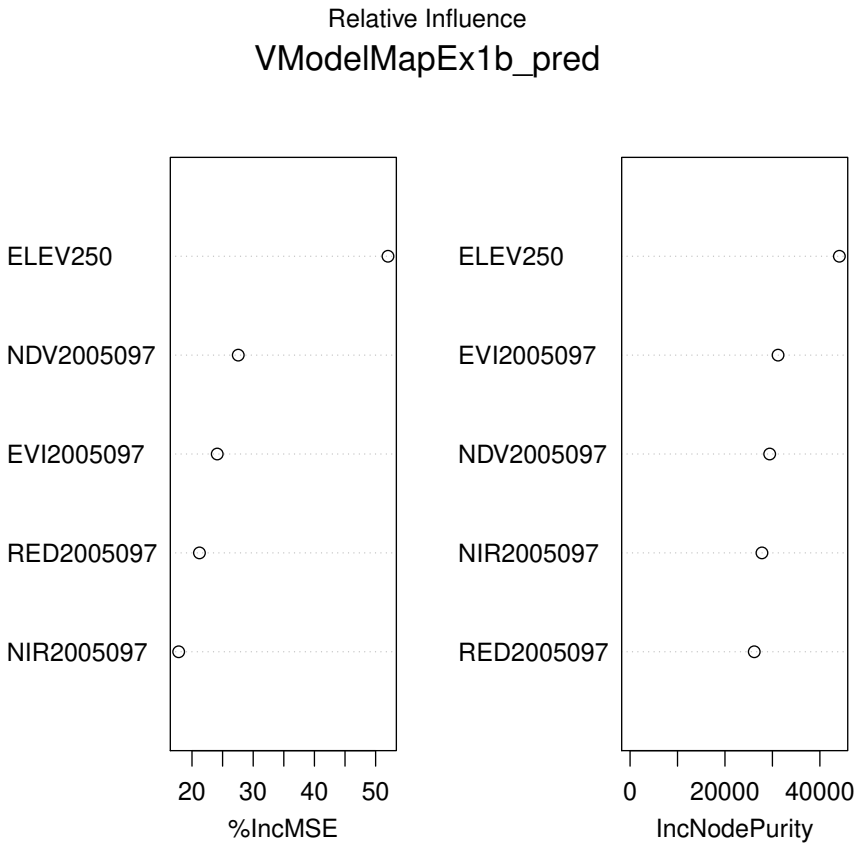

Figure 2: Example 1 - Variable importance graph for Sage percent cover (RF model).

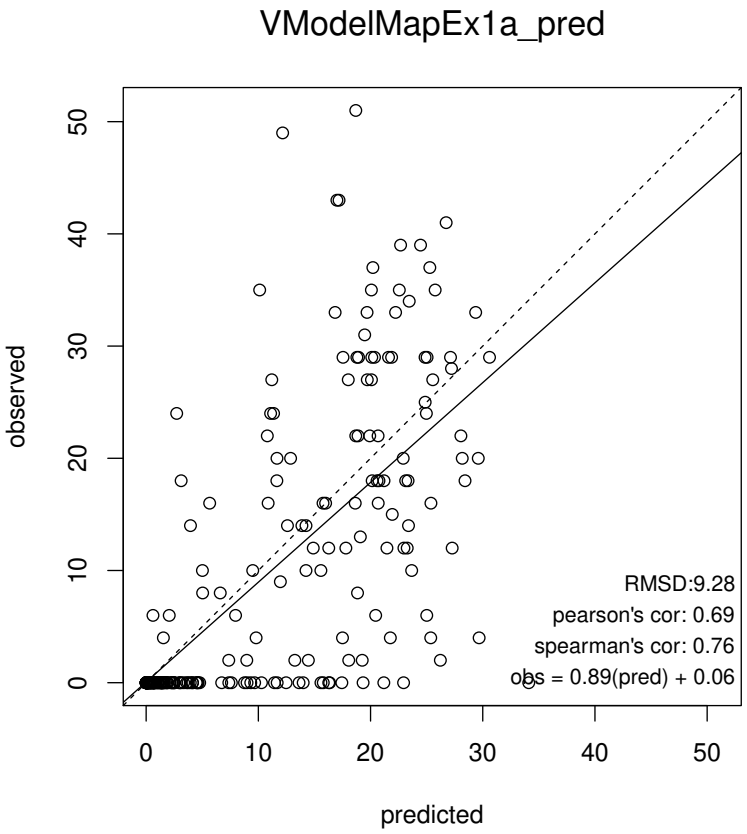

Figure 3: Example 1 - Observed verses predicted values for Pinyon percent cover (RF model).

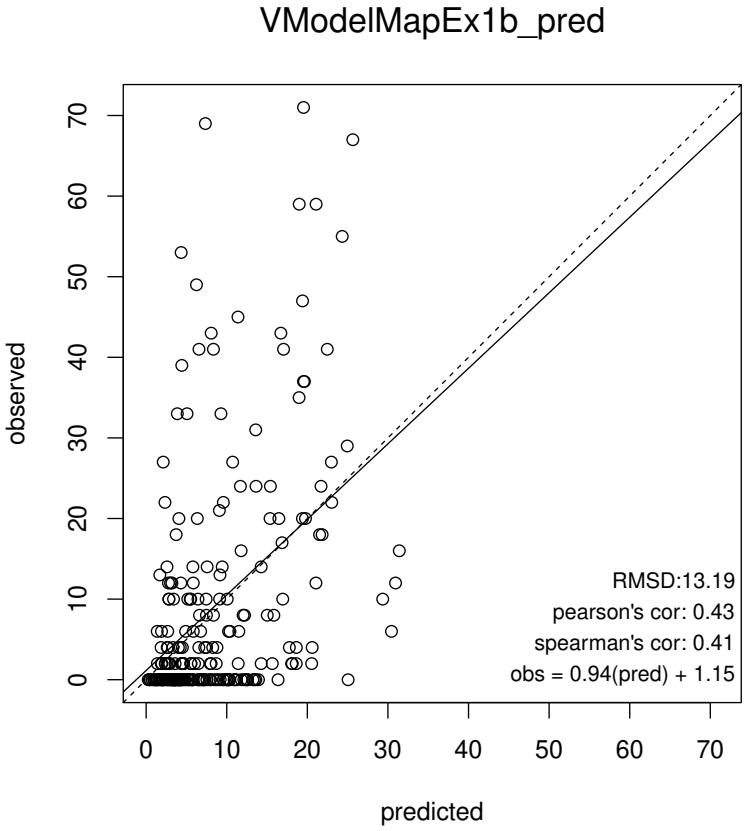

Figure 4: Example 1 - Observed verses predicted values for Sage percent cover (RF model).

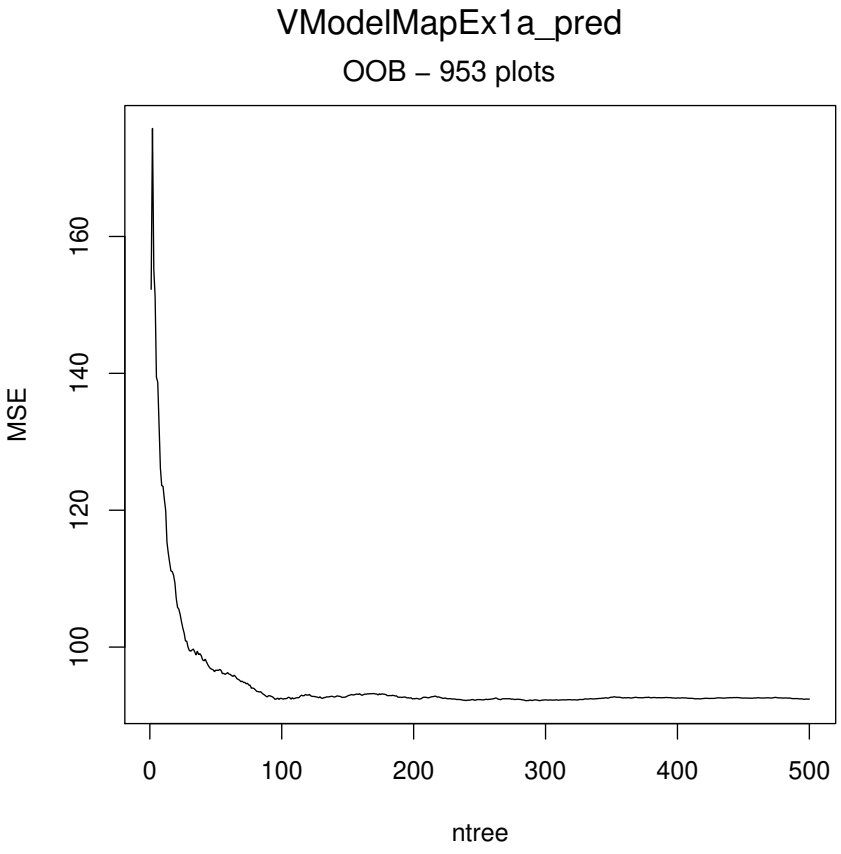

Figure 5: Example 1 - Out of Bag error as a function of number of trees for Pinyon (RF model).

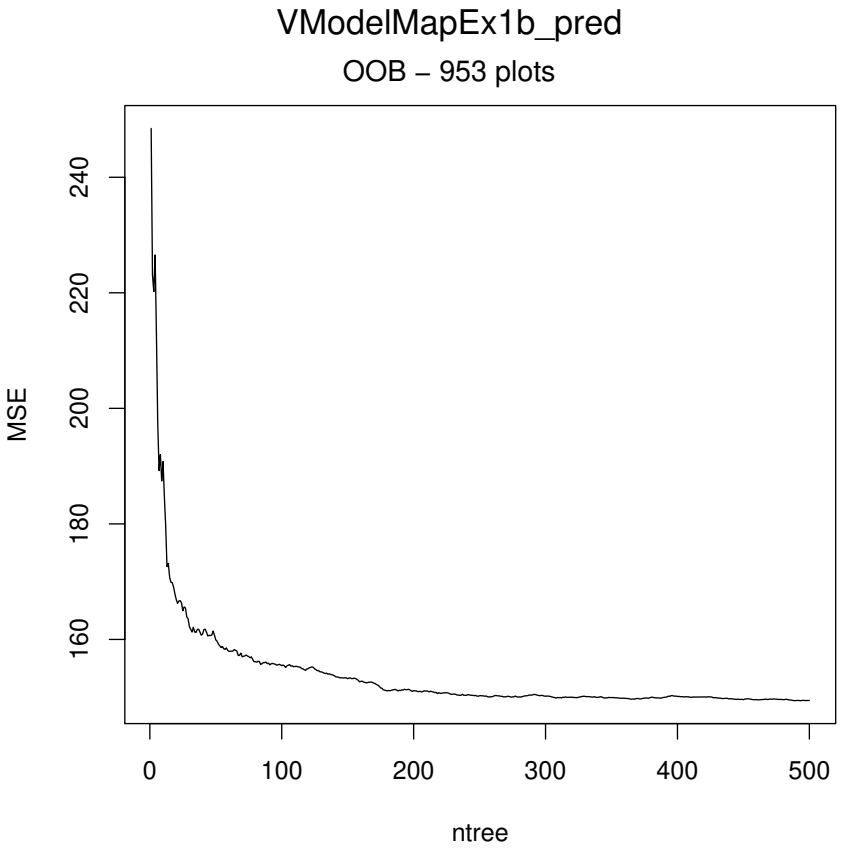

Figure 6: Example 1 - Out of Bag error as a function of number of trees for Sage (RF model).

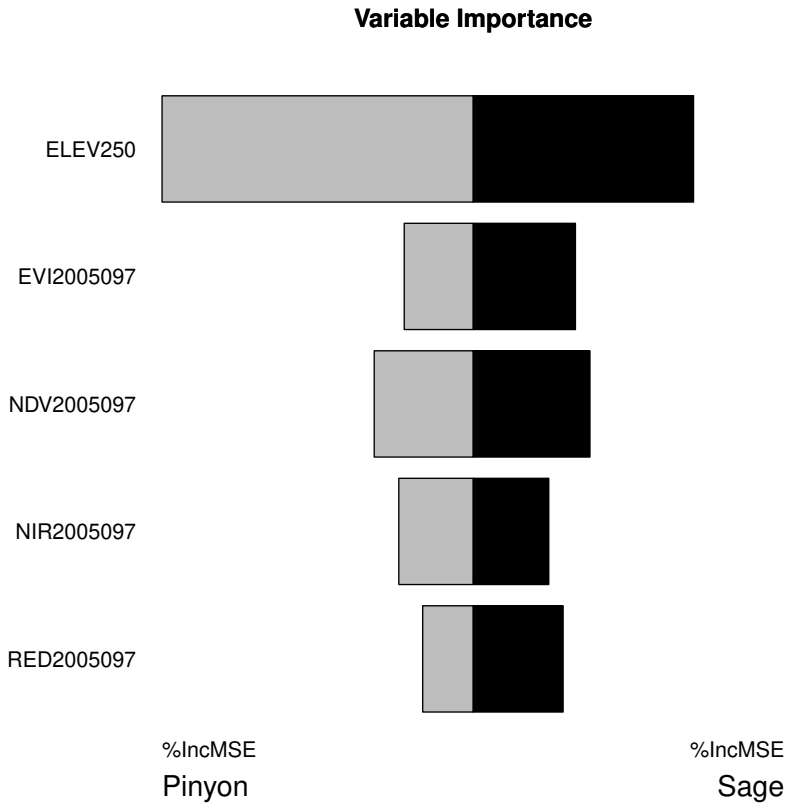

Figure 7: Example 1 - Variable Importances for Pinyon verses Sage percent cover models.

```
                        scale.by="sum",
                        main="Variable Importance",
                        device.type="pdf",
                        PLOTfn="VModelMapEx1CompareImportance",
                        folder=folder)
R>
```

The `model.importance.plot()` function can also be used to compare the two types of variable importance (Figure 8).

```
R> opar <- par(mfrow=c(2,1),mar=c(3,3,3,3),oma=c(0,0,3,0))
R> model.importance.plot( model.obj.1=model.obj.ex1a,
                        model.obj.2=model.obj.ex1a,
                        model.name.1="",
                        model.name.2="",
                        imp.type.1=1,
                        imp.type.2=2,
                        sort.by="predList",
                        predList=predList,
                        scale.by="sum",
                        main="Pinyon",
```

```
                        device.type="none",
                        cex=0.9)
R> model.importance.plot( model.obj.1=model.obj.ex1b,
                        model.obj.2=model.obj.ex1b,
                        model.name.1="",
                        model.name.2="",
                        imp.type.1=1,
                        imp.type.2=2,
                        sort.by="predList",
                        predList=predList,
                        scale.by="sum",
                        main="Sage",
                        device.type="none",
                        cex=0.9)
R> mtext("Comparison of Importance Types",side=3,line=0,cex=1.8,outer=TRUE)
R> par(opar)
```

3.2.5 Interaction Plots

The `model.interaction.plot()` function provides a diagnostic plot useful in visualizing two-way interactions between predictor variables. Two of the predictor variables from the model are used to produce a grid of possible combinations over the range of both variables. The remaining predictor variables are fixed at either their means (for continuous predictors) or their most common value (for categorical predictors). Model predictions are generated over this grid and plotted as the z axis. The `model.interaction.plot()` function was developed from the `gbm.perspec` function from the tutorial provided as appendix S3 in Elith et al. (2008).

The `model.interaction.plot()` function provides two graphical options: an image plot, and a 3-D perspective plot. These options are selected by setting `plot.type = "image"` or `plot.type = "persp"`

The `x` and `y` arguments are used to specify the predictor variables for the X and Y axis. The predictors can be specified by name, or by number, with the numbers referring to the order the variables appear in `predList`.

```
R> model.interaction.plot( model.obj.ex1a,
                         x="NIR2005097",
                         y="RED2005097",
                         main=response.name.a,
                         plot.type="image",
                         device.type="pdf",
                         MODELfn=MODELfn.a,
                         folder=folder)
R> model.interaction.plot( model.obj.ex1b,
                         x="NIR2005097",
                         y="RED2005097",
                         main=response.name.b,
                         plot.type="image",
                         device.type="pdf",
                         MODELfn=MODELfn.b,
                         folder=folder)
R> model.interaction.plot( model.obj.ex1a,
                         x=1,
                         y=3,
```

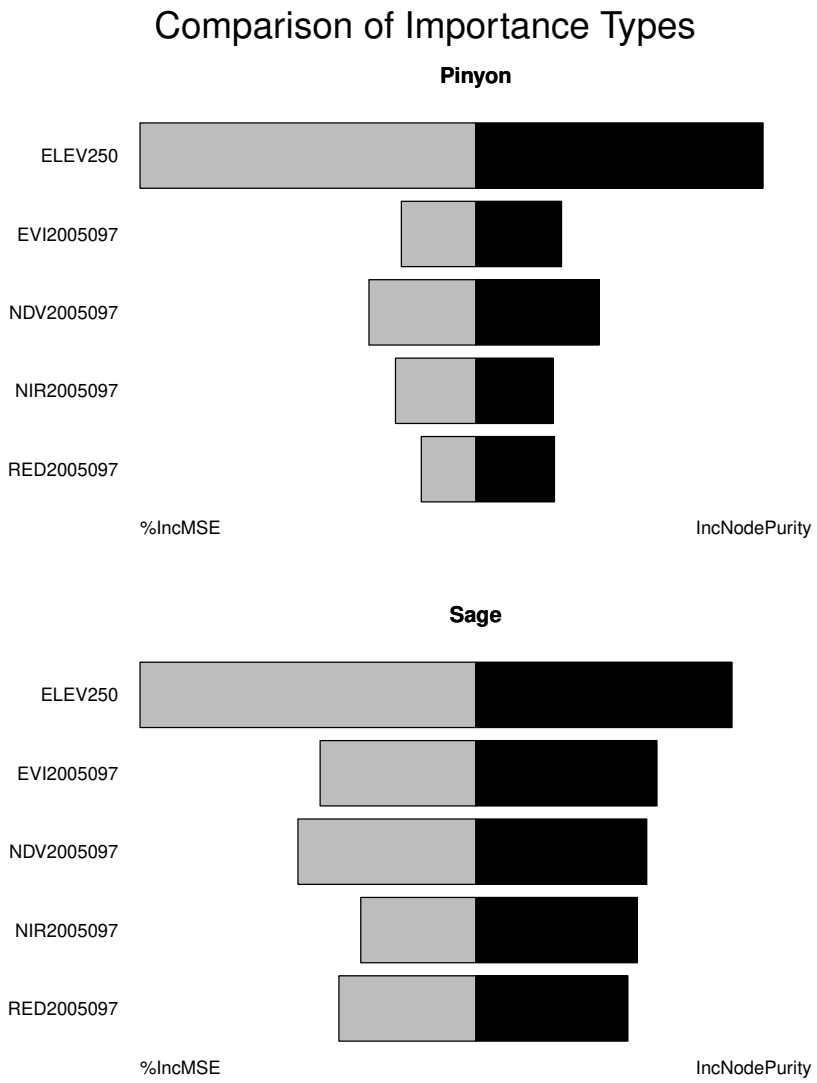

Figure 8: Example 1 - Variable Importances Types for continuous response models - the relative importance of predictor variables as measured by the mean decrease in accuracy from randomly permuting each predictor as compared to the decrease in node impurities from splitting on the variable.

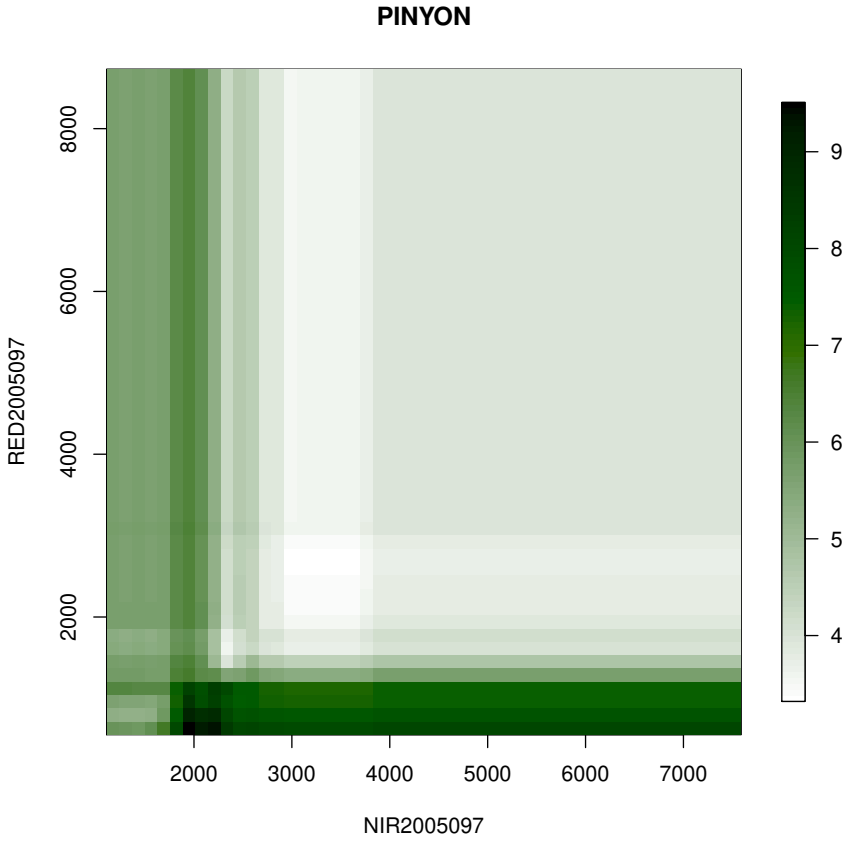

Figure 9: Example 1 - Interaction plot for Pinyon percent cover (RF model), showing interactions between two of the satellite based predictors (NIR2005097 and RED2005097). Image plot, with darker green indicating higher percent cover. Here we can see that predicted Pinyon cover is highest at low values of either NIR or RED. However low values of both predictors does not further raise the predicted cover.

```
                          main=response.name.a,
                          plot.type="image",
                          device.type="pdf",
                          MODELfn=MODELfn.a,
                          folder=folder)
R> model.interaction.plot( model.obj.ex1b,
                          x=1,
                          y=3,
                          main=response.name.b,
                          plot.type="image",
                          device.type="pdf",
                          MODELfn=MODELfn.b,
                          folder=folder)
```

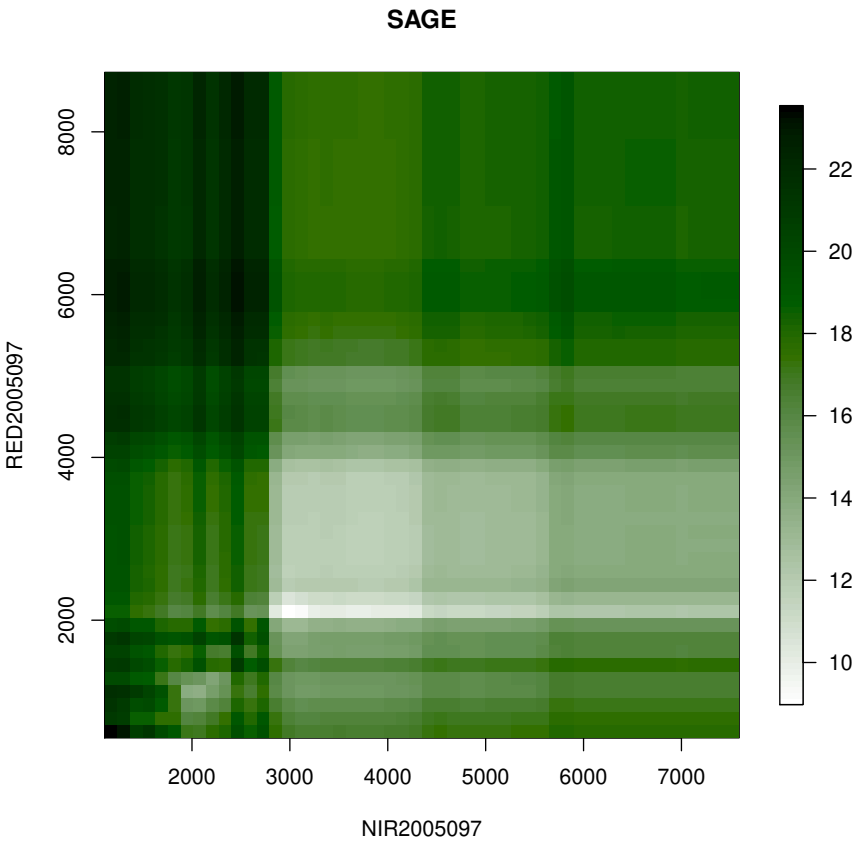

Figure 10: Example 1 - Interaction plot for Sage percent cover (RF model), showing interactions between two of the satellite based predictors (NIR2005097 and RED2005097). Image plot, with darker green indicating higher percent cover. Here we can see that predicted Sage cover is lowest when high values of NIR are combined with RED lying between 2000 and 5000. When NIR is lower than 2900, RED still has an effect on the predicted cover, but the effect is not as strong.

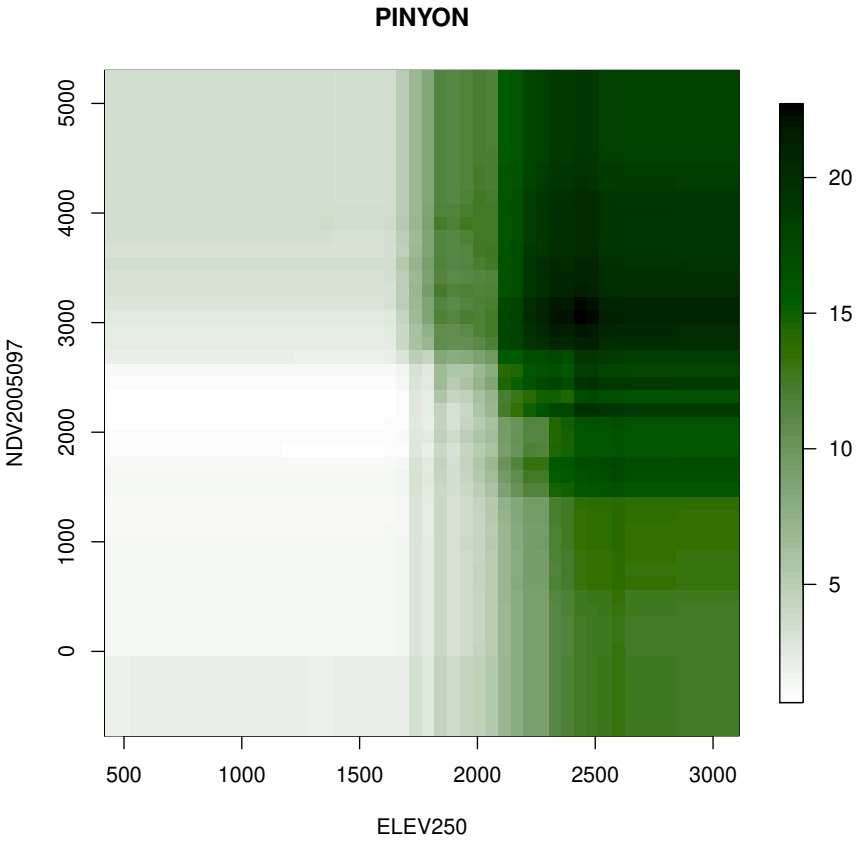

Figure 11: Example 1 - Interaction plot for Pinyon percent cover (RF model), showing interactions between elevation and a satellite based predictor (ELEV250 and NDV2005097). Image plot, with darker green indicating higher percent cover. Here we can see that predicted Pinyon cover is highest at elevation greater than 2000m. In addition, high values of NDV slightly increase the predicted cover, but there seems to be little interaction between the two predictors.

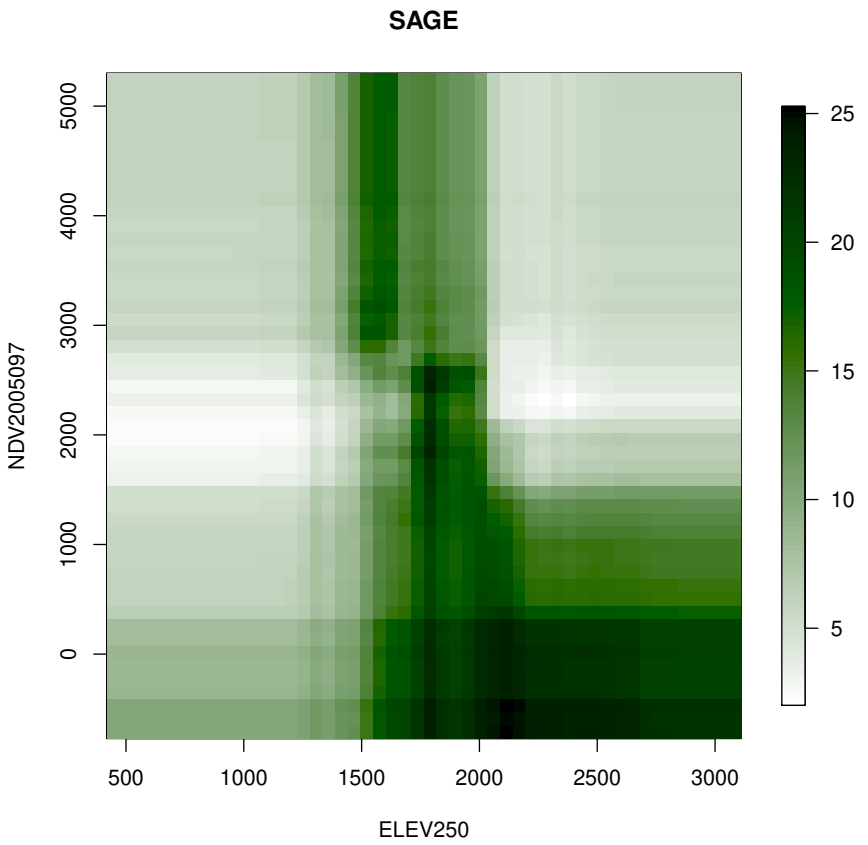

Figure 12: Example 1 - Interaction plot for Sage percent cover (RF model), showing interactions between elevation and a satellite based predictor (ELEV250 and NDV2005097). Image plot, with darker green indicating higher percent cover. Here we do see an interaction between the two predictors. At low elevations, predicted Sage cover is low throughout the range of NDV, and particularly low at mid-values. At mid elevations, predicted Sage cover is high throughout the range of NDV. At high elevations NDV has a strong influence on predicted Sage cover with high cover tied to low to mid values of NDV.

3.2.6 Map production

Before building maps of the responses, examine the predictor variable for elevation (Figure 13):

```
R> elevfn <- paste(folder,"/VModelMapData_dem_ELEVM_250.img",sep="")
R> mapgrid <- raster(elevfn)

R> opar <- par(mar=c(4,4,3,6),xpd=NA,mgp=c(3, 2, .3))
R> col.ramp<-terrain.colors(101)
R> zlim <- c(1500,maxValue(mapgrid))
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")
R> legend.colors<-col.ramp[c(100,75,50,25,1)]
R> image( mapgrid,
        col = col.ramp,
        xlab="", ylab="",
        zlim=zlim,
        asp=1, bty="n", main="")
R> legend( x=xmax(mapgrid),y=ymax(mapgrid),
         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 one row at a time.

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

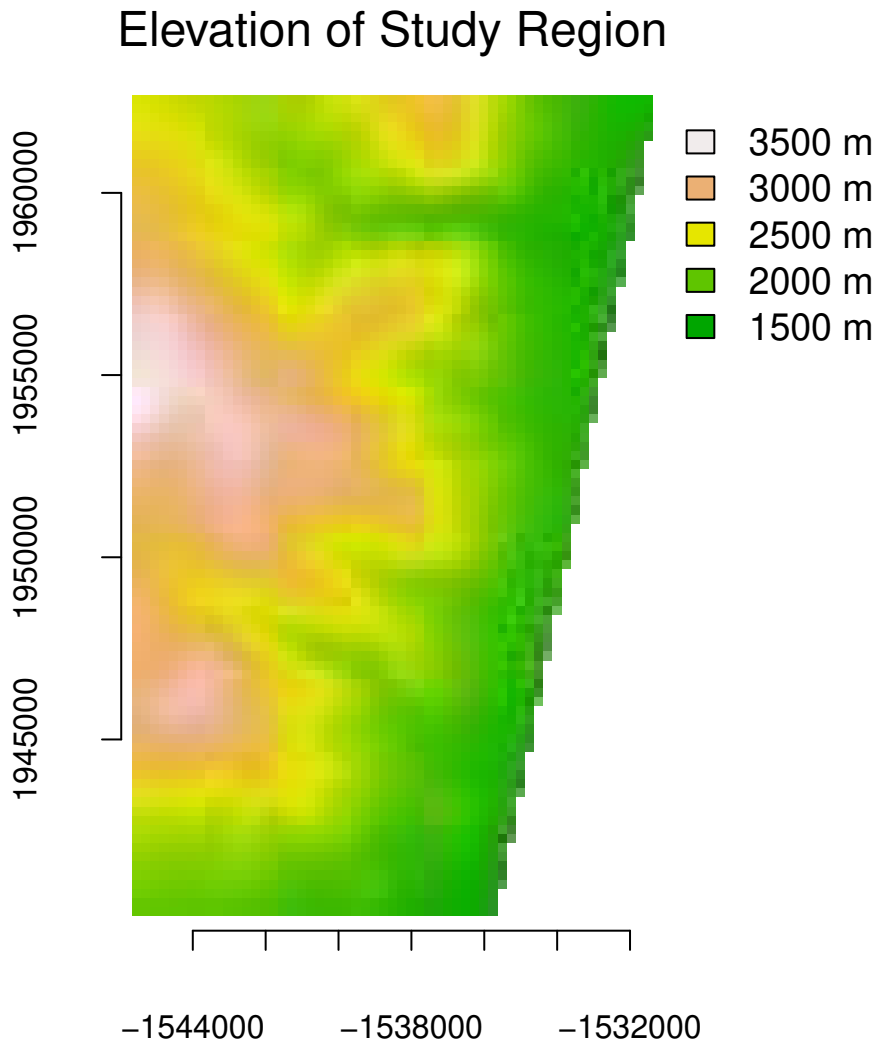

Figure 13: Elevation of study region. Projection: Universal Transverse Mercator (UTM) Zone 11, Datum: NAD83

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.

```
R>
```

```
R> model.mapmake(  model.obj=model.obj.ex1a,
                   folder=folder,
                   MODELfn=MODELfn.a,
                   rastLUTfn=rastLUTfn,
                   na.action="na.omit",
                # Mapping arguments
                   map.sd=TRUE)
R> model.mapmake(  model.obj=model.obj.ex1b,
                   folder=folder,
                   MODELfn=MODELfn.b,
                   rastLUTfn=rastLUTfn,
                   na.action="na.omit",
                # Mapping arguments
                   map.sd=TRUE)
```

The function `model.mapmake()` creates an Imagine image 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.

```
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 14). 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 <- raster(paste(MODELfn.a,"_map.img",sep=""))
R> mapgrid.b <- raster(paste(MODELfn.b,"_map.img",sep=""))
R> zlim <- c(0,max(maxValue(mapgrid.a),maxValue(mapgrid.b)))
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,
        xlab="",ylab="",xaxt="n",yaxt="n",
        zlim=zlim,
```

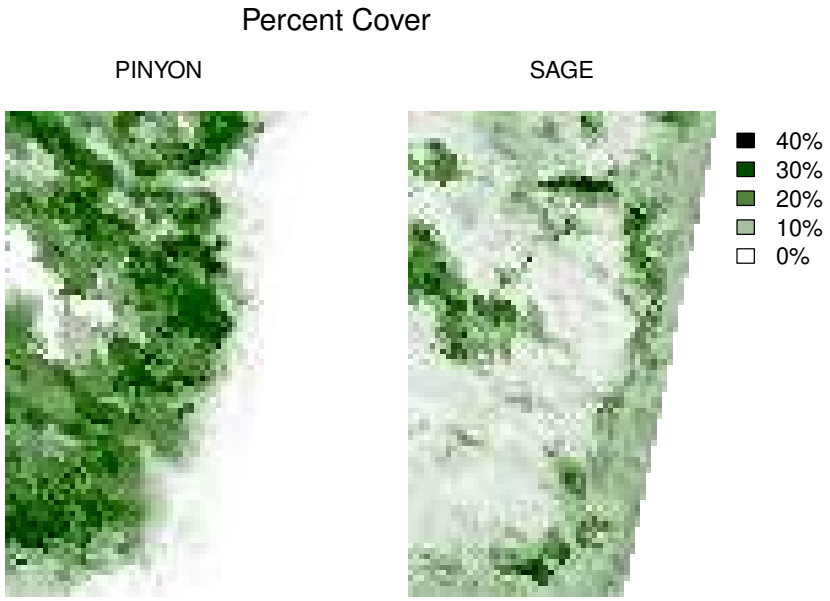

Figure 14: Example 1 - Maps of percent cover for Pinyon and Sage (RF models).

```
        asp=1,bty="n",main="")
R> mtext(response.name.a,side=3,line=1,cex=1.2)
R> image( mapgrid.b,
        col=col.ramp,
        xlab="",ylab="",xaxt="n",yaxt="n",
        zlim=zlim,
        asp=1,bty="n",main="")
R> mtext(response.name.b,side=3,line=1,cex=1.2)
R> legend( x=xmax(mapgrid.b),y=ymax(mapgrid.b),
                    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 15) 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 16). 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 <- raster(paste(MODELfn.a,"_map_stdev.img",sep=""))
```

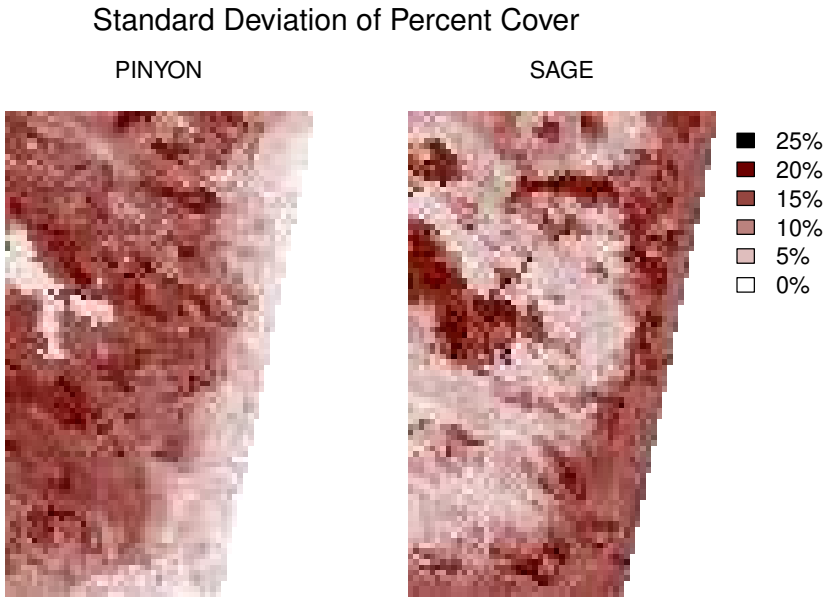

Figure 15: Example 1 - Map of standard deviation of Random Forest trees at each pixel for Pinyon and Sage (RF models).

```
R> mapgrid.b <- raster(paste(MODELfn.b,"_map_stdev.img",sep=""))
R> zlim <- c(0,max(maxValue(mapgrid.a),maxValue(mapgrid.b)))
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,
        xlab="",ylab="",xaxt="n",yaxt="n",
        zlim=zlim,
        asp=1,bty="n",main="")
R> mtext(response.name.a,side=3,line=1,cex=1.2)
R> image( mapgrid.b,
        col=stdev.ramp,xlab="",ylab="",xaxt="n",yaxt="n",
        zlim=zlim,
        asp=1,bty="n",main="")
R> mtext(response.name.b,side=3,line=1,cex=1.2)
R> legend( x=xmax(mapgrid.b),y=ymax(mapgrid.b),
                    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)
```

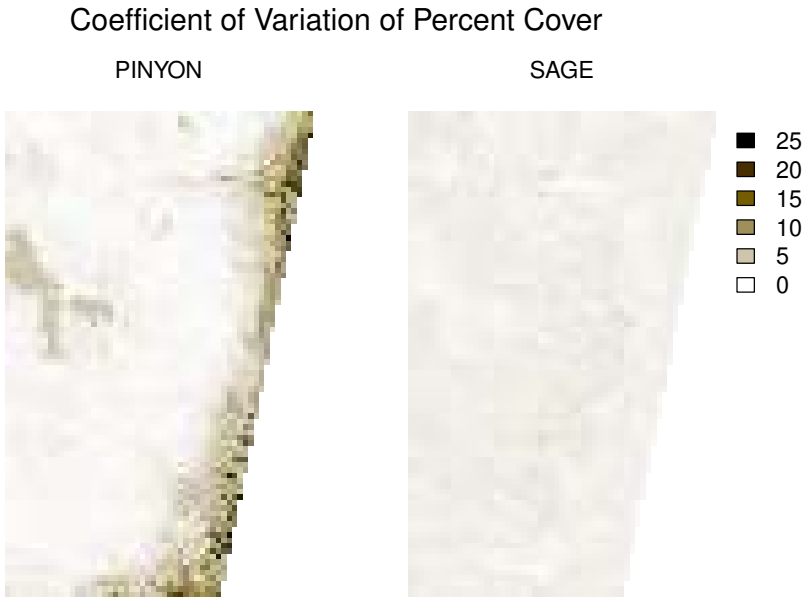

Figure 16: Example 1 - Map of coefficient of variation of Random Forest trees at each pixel for Pinyon and Sage (RF models).

```
R> mapgrid.a <-raster(paste(MODELfn.a,"_map_coefv.img",sep=""),as.image=TRUE)
R> mapgrid.b <- raster(paste(MODELfn.b,"_map_coefv.img",sep=""),as.image=TRUE)
R> zlim <- c(0,max(maxValue(mapgrid.a),maxValue(mapgrid.b)))
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,
        xlab="",ylab="",xaxt="n",yaxt="n",zlim=zlim,
        asp=1,bty="n",main="")
R> mtext(response.name.a,side=3,line=1,cex=1.2)
R> image( mapgrid.b,
        col=coefv.ramp,
        xlab="",ylab="",xaxt="n",yaxt="n",
        zlim=zlim,
        asp=1,bty="n",main="")
R> mtext(response.name.b,side=3,line=1,cex=1.2)
R> legend( x=xmax(mapgrid.b),y=ymax(mapgrid.b),
                    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"
```

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, binary or categorical. 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 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,
                                  unique.rowname=unique.rowname,
                                  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,
                                  unique.rowname=unique.rowname,
                                  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 17 and Figure 18) in addition to the variable importance graph (Figure 19 and Figure 20).

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,
                                  # Model Validation Arguments
                                         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,
                                   # Model Validation Arguments
                                         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
1       Default      0.50 0.92        0.92        0.92
```

```
2     Sens=Spec      0.52 0.92        0.92        0.92
3  MaxSens+Spec      0.62 0.92        0.91        0.93
4      MaxKappa      0.62 0.92        0.91        0.93
5        MaxPCC      0.64 0.92        0.90        0.94
6  PredPrev=Obs      0.57 0.92        0.91        0.92
7       ObsPrev      0.46 0.92        0.92        0.91
8      MeanProb      0.47 0.92        0.92        0.91
9    MinROCdist      0.62 0.92        0.91        0.93
10      ReqSens      0.76 0.90        0.85        0.95
11      ReqSpec      0.22 0.90        0.96        0.85
12         Cost      0.64 0.92        0.90        0.94
   Kappa
1   0.83
2   0.83
3   0.84
4   0.84
5   0.84
6   0.84
7   0.83
8   0.83
9   0.84
10  0.81
11  0.81
12  0.84
```

And for Sage:

```
R> opt.thresh.b

    opt.methods threshold  PCC sensitivity specificity
1       Default      0.50 0.66        0.81        0.48
2     Sens=Spec      0.61 0.67        0.67        0.66
3  MaxSens+Spec      0.61 0.67        0.67        0.66
4      MaxKappa      0.61 0.67        0.67        0.66
5        MaxPCC      0.60 0.67        0.69        0.65
6  PredPrev=Obs      0.59 0.66        0.70        0.61
7       ObsPrev      0.56 0.66        0.74        0.57
8      MeanProb      0.60 0.67        0.69        0.64
9    MinROCdist      0.61 0.67        0.67        0.66
10      ReqSens      0.44 0.66        0.86        0.40
11      ReqSpec      0.79 0.56        0.33        0.86
12         Cost      0.60 0.67        0.69        0.65
   Kappa
1   0.29
2   0.33
3   0.33
4   0.33
5   0.33
6   0.32
7   0.31
8   0.33
9   0.33
10  0.27
11  0.18
12  0.33
```

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      0.52           0.46 0.47
3            MaxSens+Spec      0.62           0.46 0.46
4                MaxKappa      0.62           0.46 0.46
5                  MaxPCC      0.64           0.46 0.45
6            PredPrev=Obs      0.57           0.46 0.46
7                 ObsPrev      0.46           0.46 0.47
8                MeanProb      0.47           0.46 0.47
9              MinROCdist      0.62           0.46 0.46
10               ReqSens      0.76           0.46 0.42
11               ReqSpec      0.22           0.46 0.52
12                  Cost      0.64           0.46 0.45
```

And for Sage:

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

```
   opt.thresh.opt.methods threshold Obs.Prevalence pred
1                 Default      0.50           0.56 0.69
2               Sens=Spec      0.61           0.56 0.53
3            MaxSens+Spec      0.61           0.56 0.53
4                MaxKappa      0.61           0.56 0.53
5                  MaxPCC      0.60           0.56 0.54
6            PredPrev=Obs      0.59           0.56 0.57
7                 ObsPrev      0.56           0.56 0.60
8                MeanProb      0.60           0.56 0.54
9              MinROCdist      0.61           0.56 0.53
10               ReqSens      0.44           0.56 0.74
11               ReqSpec      0.79           0.56 0.25
12                  Cost      0.60           0.56 0.54
```

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.

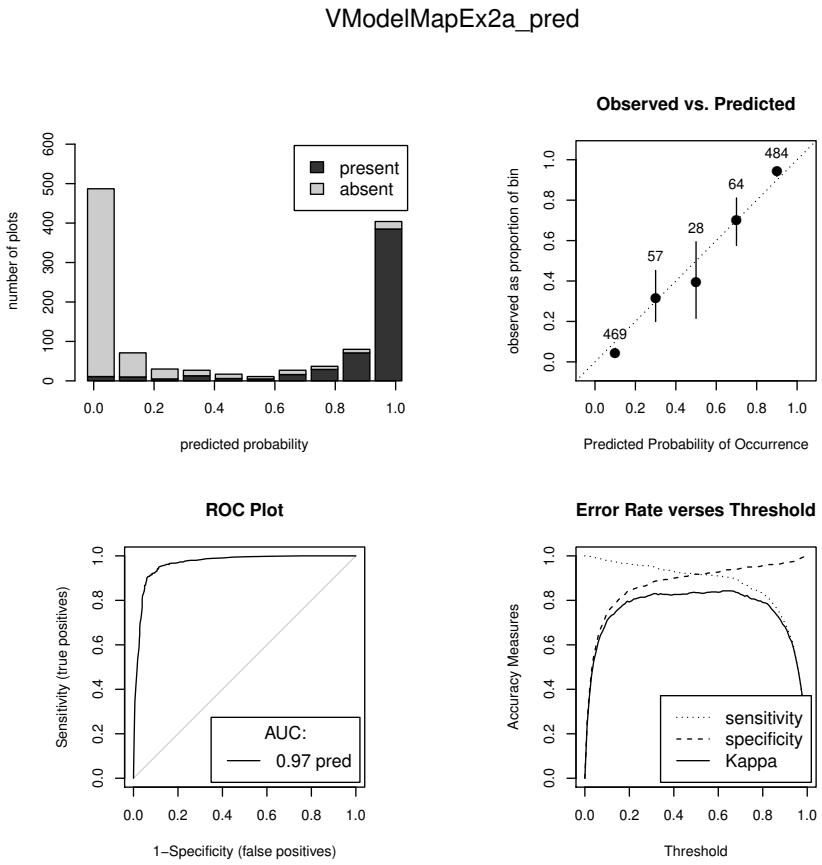

Figure 17: Example 2 - Model quality and threshold selection graphs for Pinyon presence (RF model).

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.

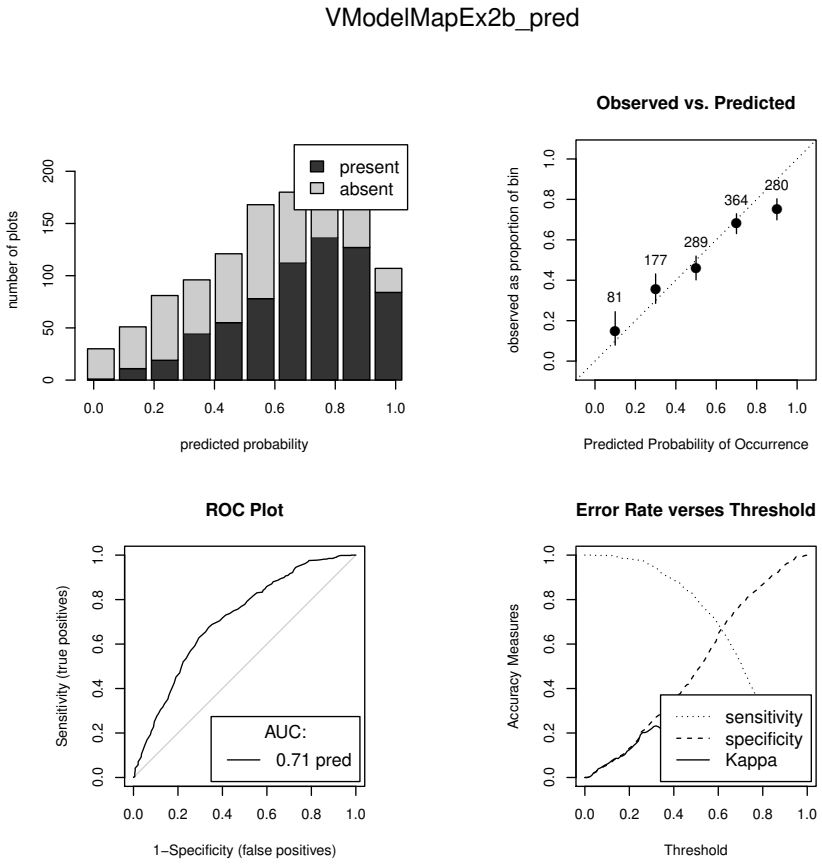

Figure 18: Example 2 - Model quality and threshold selection graphs for Sage presence (RF model).

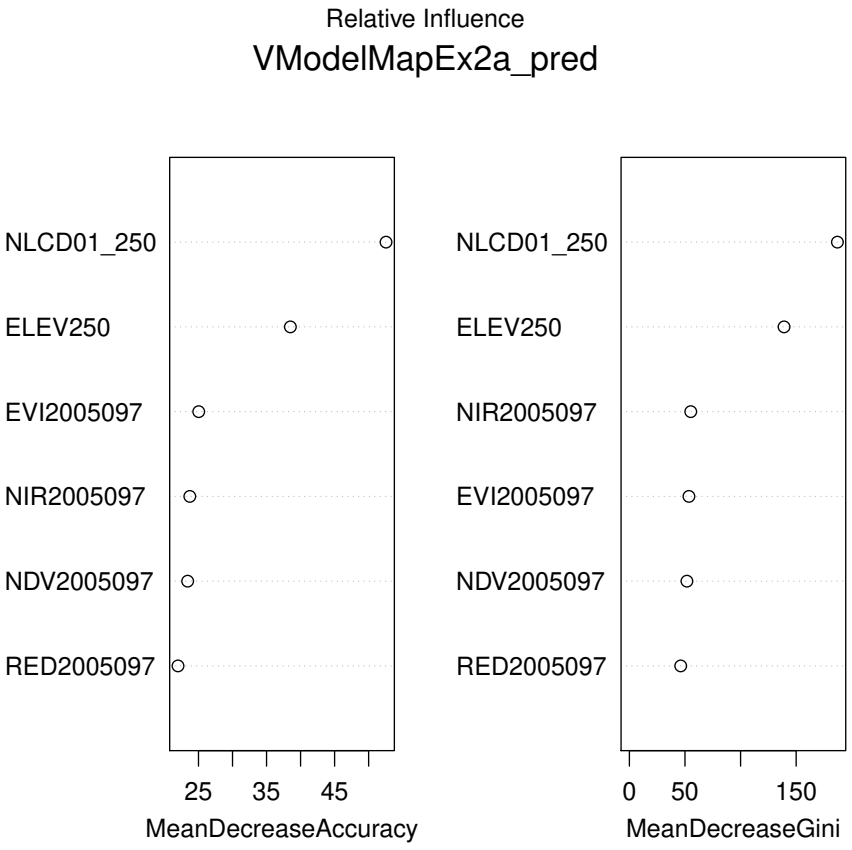

Figure 19: Example 2 - Variable importance graph for Pinyon presence (RF model).

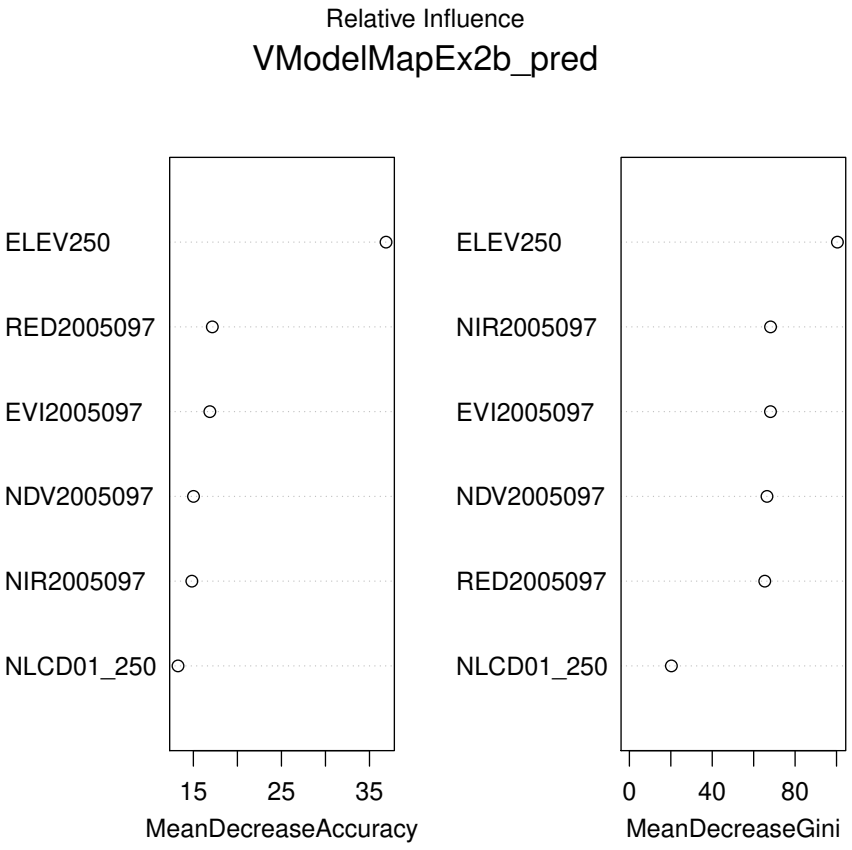

Figure 20: Example 2 - Variable importance graph for Sage presence (RF model).

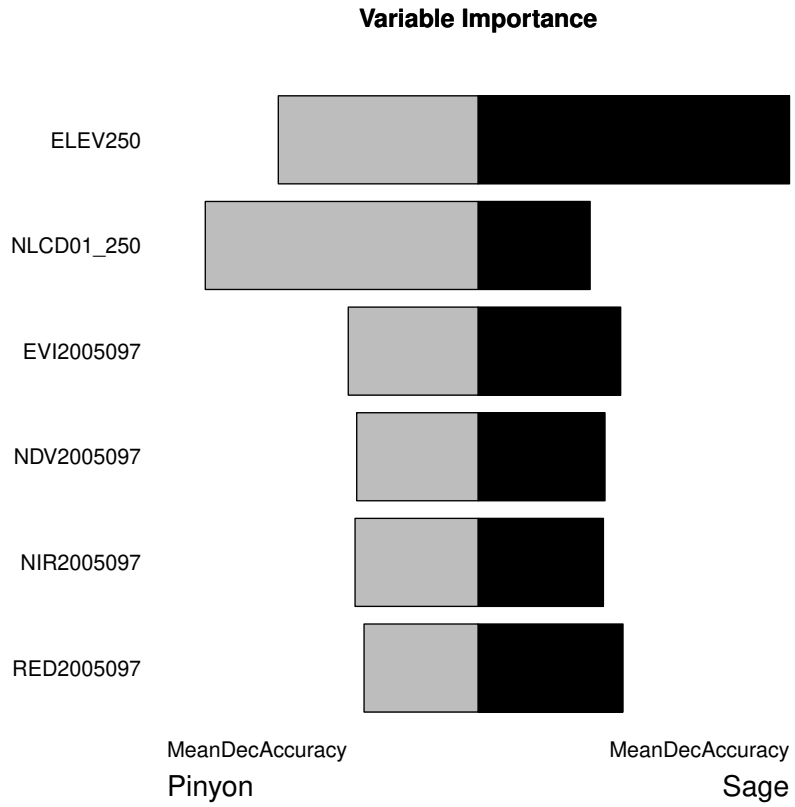

Figure 21: Example 2 - Variable Importances for Pinyon verses Sage presence models.

3.3.4 Comparing Variable Importance

The `model.importance.plot()` function uses a back-to-back barplot to compare variable importance between two models built from the same predictor variables (Figure 21).

```
R> model.importance.plot( model.obj.1=model.obj.ex2a,
                          model.obj.2=model.obj.ex2b,
                          model.name.1="Pinyon",
                          model.name.2="Sage",
                          sort.by="predList",
                          predList=predList,
                          scale.by="sum",
                          main="Variable Importance",
                          device.type="pdf",
                          PLOTfn="VModelMapEx2CompareImportance",
                          folder=folder)
R>
```

Because a binary response model is a two-class example of a categorical response model, we can use categorical tools to investigate the class specific variable importances. (Figure 22) compares the relative importance of the predictor variables inr predicting Presences to their importances in predicting Absences.

```
R> opar <- par(mfrow=c(2,1),mar=c(3,3,3,3),oma=c(0,0,3,0))
R> model.importance.plot( model.obj.1=model.obj.ex2a,
                         model.obj.2=model.obj.ex2a,
                         model.name.1="Absence",
                         model.name.2="Presence",
                         class.1="0",
                         class.2="1",
                         sort.by="predList",
                         predList=predList,
                         scale.by="sum",
                         main="Pinyon Variable Importance",
                         device.type="none",
                         cex=0.9)
R> model.importance.plot( model.obj.1=model.obj.ex2b,
                         model.obj.2=model.obj.ex2b,
                         model.name.1="Absence",
                         model.name.2="Presence",
                         class.1="0",
                         class.2="1",
                         sort.by="predList",
                         predList=predList,
                         scale.by="sum",
                         main="Sage Variable Importance",
                         device.type="none",
                         cex=0.9)
R> mtext("Presence-Absence Variable Importance Comparison",side=3,line=0,cex=1.8,outer=TRUE)
R> par(opar)
```

3.3.5 Interaction Plots

Here we will look at how the `model.interaction.plot()` function works with a factored predictor variable.

In image plots the levels of the factored predictor are shown as vertical or horizontal bars across the plot region.

In 3-D perspective plots the levels are represented by ribbons across the prediction surface.

```
R> model.interaction.plot( model.obj.ex2a,
                          x="ELEV250",
                          y="NLCD01_250",
                          main=response.name.a,
                          plot.type="image",
                          device.type="pdf",
                          MODELfn=MODELfn.a,
                          folder=folder)
R> model.interaction.plot( model.obj.ex2b,
                          x="ELEV250",
                          y="NLCD01_250",
                          main=response.name.b,
                          plot.type="image",
                          device.type="pdf",
                          MODELfn=MODELfn.b,
                          folder=folder)
```

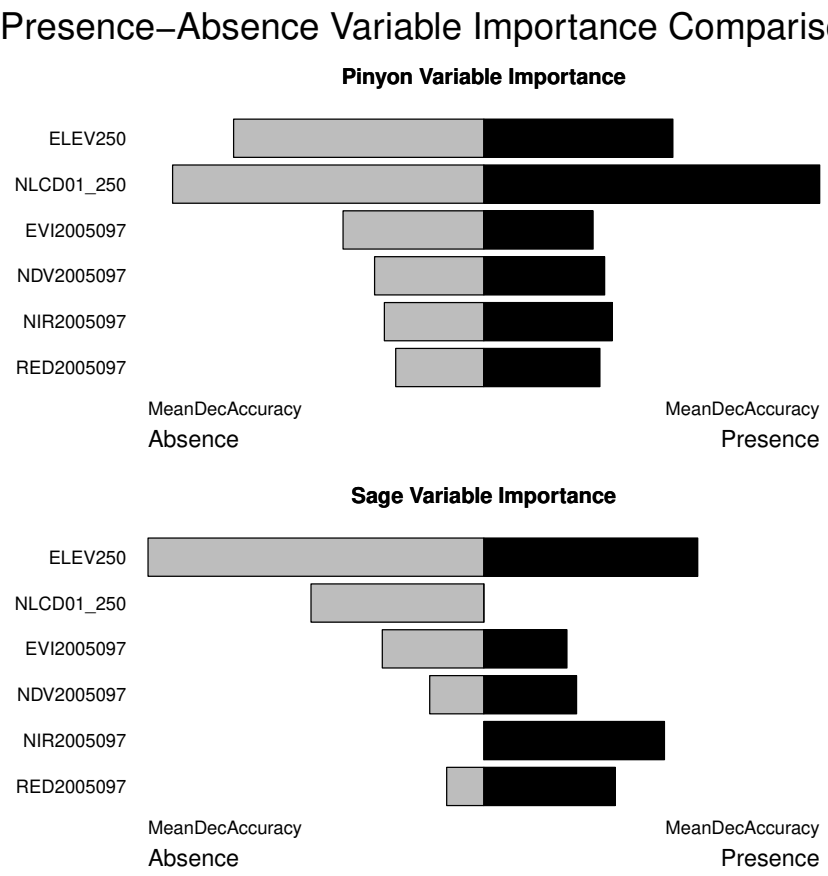

Figure 22: Example 2 - Relative variable importances (permutation based) for predicting Presences verses predicting Absences for Pinyon and Sage.

```
R> model.interaction.plot( model.obj.ex2a,
                           x="ELEV250",
                           y="NLCD01_250",
                           main=response.name.a,
                           plot.type="persp",
                           device.type="pdf",
                           MODELfn=MODELfn.a,
                           folder=folder,
                           theta=300,
                           phi=55)
R> model.interaction.plot( model.obj.ex2b,
                           x="ELEV250",
                           y="NLCD01_250",
                           main=response.name.b,
                           plot.type="persp",
                           device.type="pdf",
                           MODELfn=MODELfn.b,
                           folder=folder,
                           theta=300,
                           phi=55)
```

3.3.6 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,
                  na.action="na.omit")
R> model.mapmake( model.obj=model.obj.ex2b,
                  folder=folder,
                  MODELfn=MODELfn.b,
                  rastLUTfn=rastLUTfn,
                  na.action="na.omit")
R>
```

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
```

Begin by mapping the probability surface, in other words, the probability that the species is present at each grid point (Figure 27).

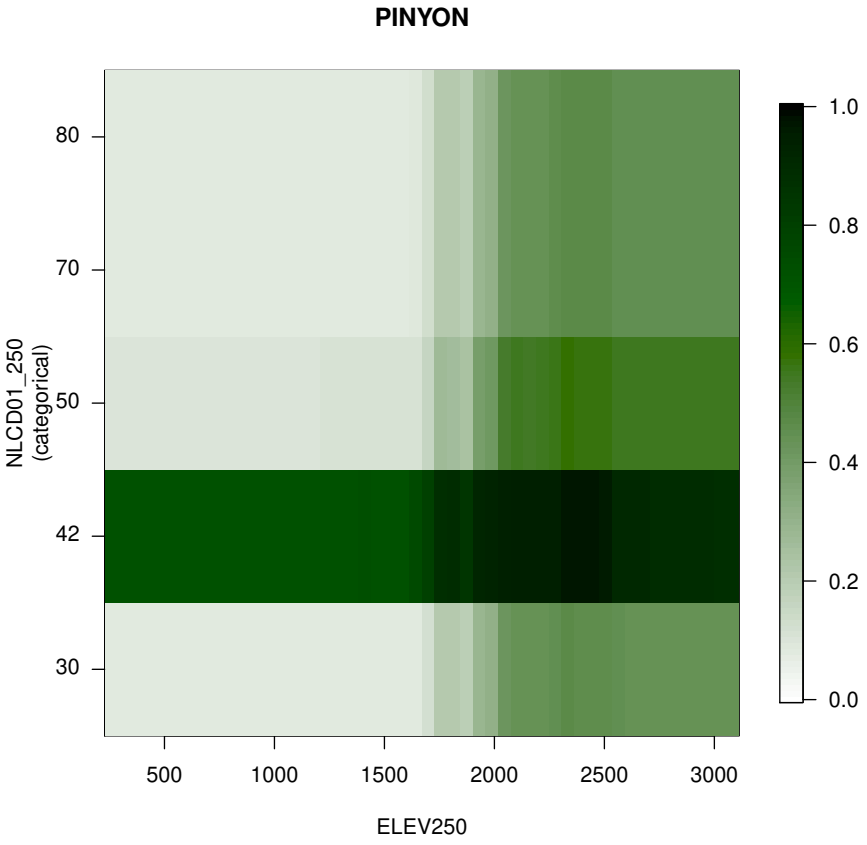

Figure 23: Example 2 - Interaction plot for Pinyon presence-absence (RF model), showing interactions between elevation and National Land Cover Dataset classes (ELEV250 and NLCD01_250). Image plot, with darker green indicating higher probability of presence. Here we see that in all NLCD classes predicted Pinyon presence is strongly tied to elevation, with low presence below 2000m, and moderate presence at higher elevations.

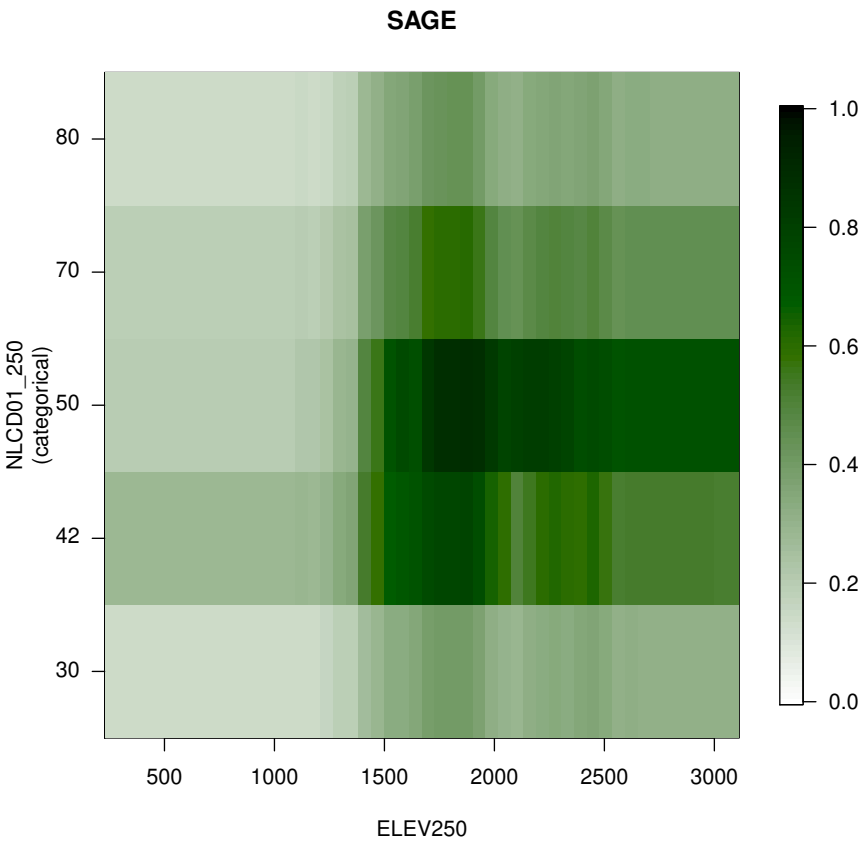

Figure 24: Example 2 - Interaction plot for Sage presence-absence (RF model), showing interactions between elevation and National Land Cover Dataset classes (ELEV250 and NLCD01_250). Image plot, with darker green indicating higher probability of presence. Here we see that predicted Sage presence is influenced by both elevation and NLCD class. In most NLCD classes predicted presence is highest between 1400m and 2000m, with very low presence at lower elevations and low presence at higher elevations. In contrast, in NLCD class 50 while predicted presence drops slightly as elevation increases, it remains quite high all the way to 3000m.

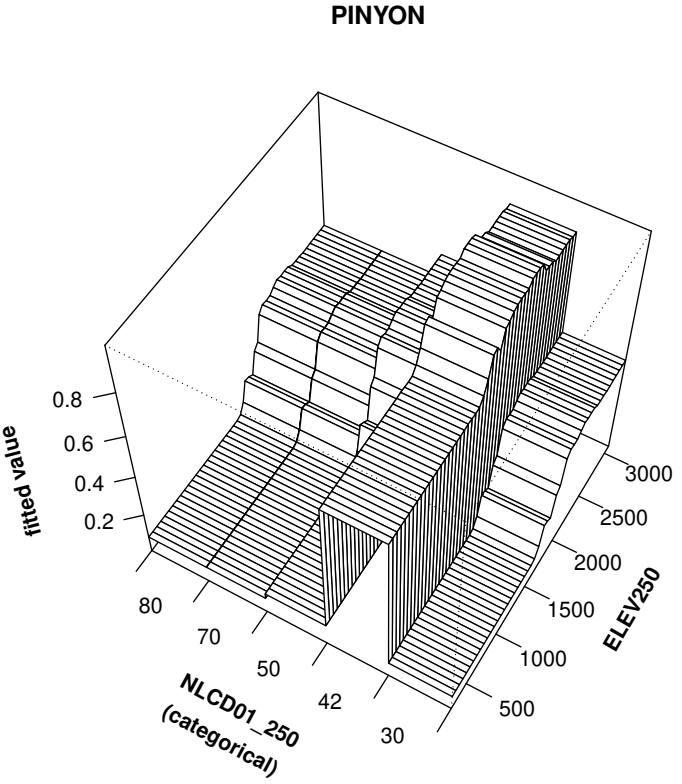

Figure 25: Example 2 - Interaction plot for Pinyon presence-absence (RF model), showing interactions between elevation and National Land Cover Dataset classes (ELEV250 and NLCD01_250). Perspective plot, with probability or presence shown on the Z axis. In the perspective plot (as compared to the image plot) it is easier to see that while predicted Pinyon presence is influenced by both elevation and NLCD class, the shape of the relationship between elevation and presence is similar in all NLCD classes, and while the overall probability is higher in some classes, the curves relating probability to elevation are generally parallel from class to class. Therefore there appears to be little 2-way interaction between these predictors.

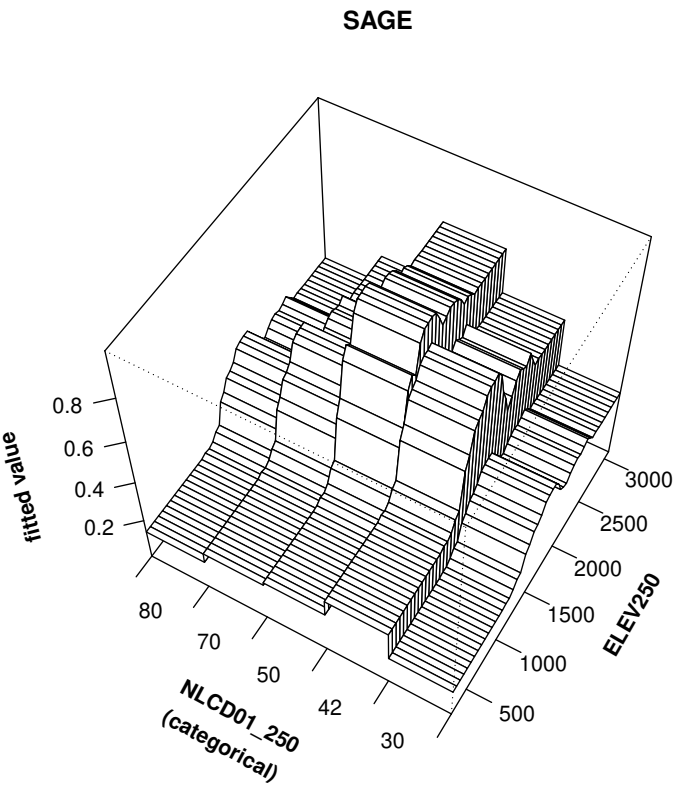

Figure 26: Example 2 - Interaction plot for Sage presence-absence (RF model), showing interactions between elevation and National Land Cover Dataset classes (ELEV250 and NLCD01_250). Perspective plot, with probability or presence shown on the Z axis. Here we can see that while all NLCD classes have low predicted Sage presence at low elevation, in NLCD class 50 at mid elevations predicted presence shoots higher than the other classes, and then does not drop as far as the other classes at high elevations. Resulting in a different shape of the elevation verses probability curve for class 50.

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 <- raster(paste(MODELfn.a,"_map.img",sep=""))
R> mapgrid.b <- raster(paste(MODELfn.b,"_map.img",sep=""))
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,
        xlab="",ylab="",yaxt="n",main="",zlim=c(0,1),
        asp=1,bty="n",xaxt="n")
R> mtext(response.name.a,side=3,line=1,cex=1.2)
R> image( mapgrid.b,
        col=probpres.ramp,
        xlab="",ylab="",xaxt="n",yaxt="n",
        zlim=c(0,1),
        asp=1,bty="n",main="")
R> mtext(response.name.b,side=3,line=1,cex=1.2)
R> legend( x=xmax(mapgrid.b),y=ymax(mapgrid.b),
     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

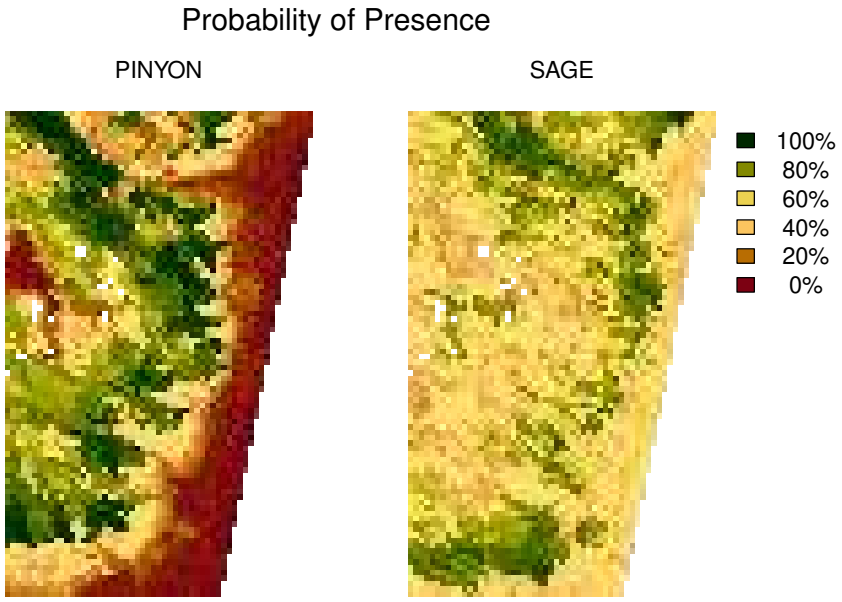

Figure 27: Example 2 - Probability surface map for presence of Pinyon and Sage (RF models).

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 28 and 29). 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 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 <- raster(paste(MODELfn.a,"_map.img",sep=""))
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
       v <- getValues(presencegrid)
         v <- ifelse(v > thresh,1,0)
       presencegrid <- setValues(presencegrid, v)

         image( presencegrid,
                col=c("white","forestgreen"),
                zlim=c(0,1),
                asp=1,
                bty="n",
                xaxt="n", yaxt="n",
```

```
            main="",xlab="",ylab="")
        if(i==2){
                legend( x=xmax(mapgrid),y=ymax(mapgrid),
                        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=.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 <- raster(paste(MODELfn.b,"_map.img",sep=""))
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
       v <- getValues(presencegrid)
         v <- ifelse(v > thresh,1,0)
       presencegrid <- setValues(presencegrid, v)

         image( presencegrid,
                col=c("white","forestgreen"),
                xlab="",ylab="",xaxt="n", yaxt="n",
                zlim=c(0,1),
                asp=1,bty="n",main="")
         if(i==2){
                 legend( x=xmax(mapgrid),y=ymax(mapgrid),
                         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=.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 - Random Forest - Categorical Response

Example 3 builds a categorical response model for vegetation category. The response variable consists of four categories: TREE, SHRUB, OTHERVEG, and NONVEG. This model will use the same predictors as Model 2. Out-of-bag estimates are used for model validation.

3.4.1 Set up

Define model type.

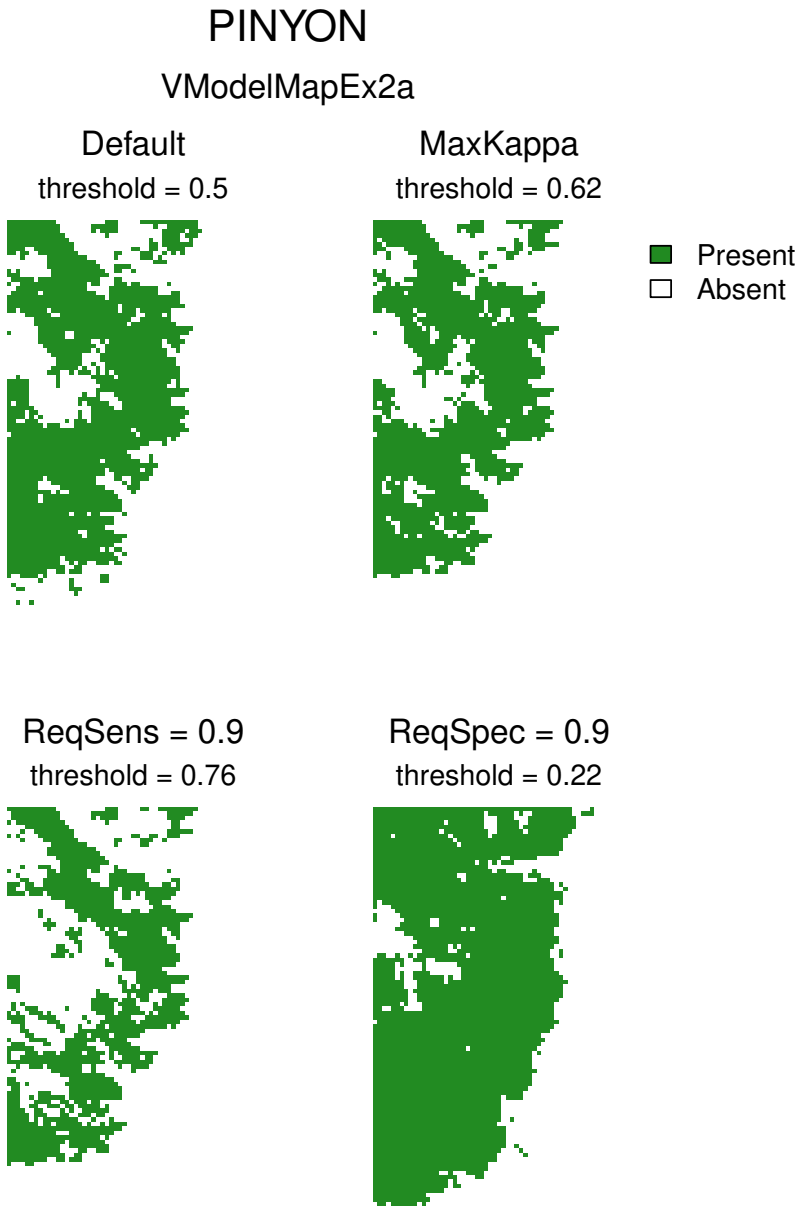

Figure 28: Example 2 - Presence-Absence maps by four different threshold selection criteria for Pinyon (RF model).

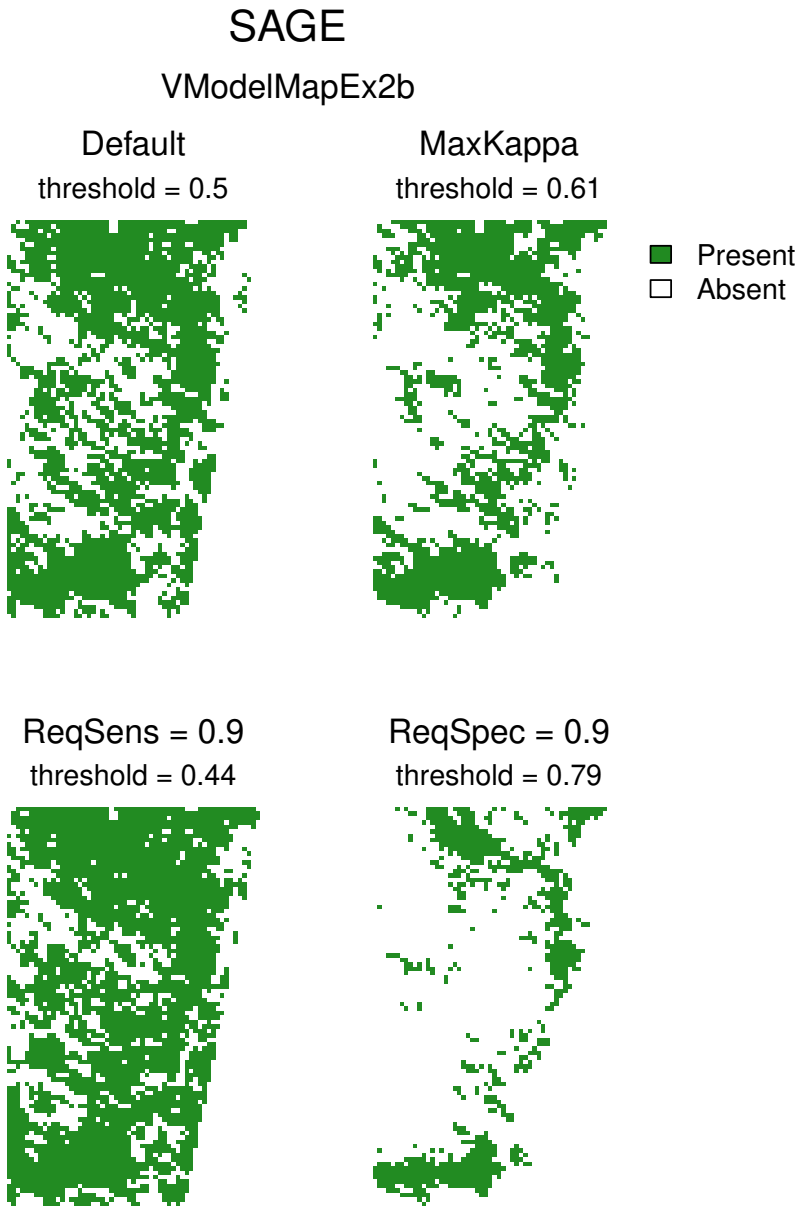

Figure 29: Example 2 - Presence-Absence maps by four different threshold selection criteria for Sage (RF model).

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

Define data.

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

Define `folder`.

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

Define model filenames.

```
R> MODELfn    <- "VModelMapEx3"
```

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, binary or categorical.

```
R> response.name <- "VEGCAT"
R> response.type <- "categorical"
```

Define the seeds for each model.

```
R> seed <- 44
```

Define the column that contains unique identifiers.

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

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. 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.ex3 <- model.build( model.type=model.type,
                                 qdata.trainfn=qdatafn,
                                 folder=folder,
                                 unique.rowname=unique.rowname,
                                 MODELfn=MODELfn,
                                 predList=predList,
                                 predFactor=predFactor,
                                 response.name=response.name,
                                 response.type=response.type,
                                 seed=seed)
R>
```

3.4.3 Model Diagnostics

Make Out-Of-Bag model predictions on the training data and run the diagnostics on these predictions. Save PDF 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 categorical response model model diagnostics include a CSV file the observed and predicted values, as well as a CSV file of the confusion matrix and its associated Kappa value and MAUC.

```
R> model.pred.ex3 <- model.diagnostics( model.obj=model.obj.ex3,
                                        qdata.trainfn=qdatafn,
                                        folder=folder,
                                        MODELfn=MODELfn,
                                        unique.rowname=unique.rowname,
                                 # Model Validation Arguments
                                        prediction.type="OOB",
                                        device.type="pdf",
                                        cex=1.2)
R>
```

Take a closer look at the text file output for the confusion matrix. Read this file into R now.

```
R> CMX.CSV <- read.table(   paste( MODELfn,"_pred_cmx.csv",sep=""),
                                header=FALSE,
                                sep=",",
                                stringsAsFactors=FALSE)
R> CMX.CSV

         V1               V2              V3
1     Kappa          Kappa.sd        observed
2  0.279154         0.0238329           NONVEG
3 predicted            NONVEG              492
4 predicted          OTHERVEG                7
5 predicted             SHRUB               80
```

```
6 predicted              TREE               90
7     total             total              669
8  Omission          Omission 0.26457399103139
9      MAUC 0.794949436880685              cmx
                 V4                V5                V6
1          observed          observed          observed
2          OTHERVEG             SHRUB              TREE
3                37               101               143
4                17                 8                 1
5                19               114                 1
6                 3                 2                76
7                76               225               221
8 0.776315789473684 0.493333333333333 0.656108597285068
9               cmx               cmx               cmx
     V7                V8
1 total        Commission
2 total        Commission
3   773 0.363518758085382
4    33 0.484848484848485
5   214 0.467289719626168
6   171 0.555555555555556
7  1191               PCC
8   PCC 0.586901763224181
9   cmx               cmx
```

The **PresenceAbsence** package function `Kappa()` is used to calculate Kappa for the confusion matrix. Note that while most of the functions in the **PresenceAbsence** package are only applicable to binary confusion matrices, the `Kappa()` function will work on any size confusion matrix.

The **HandTill2001** package is used to calculate the Multiple class Area under the Curve (MAUC) and described by Hand and Till (2001).

The text file output of the confusion matrix is designed to be easily interpreted in Excel, but is not very workable for carrying out analysis in R. However, it is relativly easy to calculate the confusion matrix from the CSV file of the observed and predicted values.

```
R> PRED <-read.table(   paste( MODELfn,"_pred.csv",sep=""),
                            header=TRUE,
                            sep=",",
                            stringsAsFactors=TRUE)
R> head(PRED)

  ID    obs   pred    NONVEG    OTHERVEG      SHRUB
1  1 NONVEG NONVEG 0.4730539 0.029940120 0.32934132
2  2 NONVEG   TREE 0.4406780 0.000000000 0.01129944
3  3 NONVEG NONVEG 0.7807487 0.000000000 0.02673797
4  4   TREE NONVEG 0.5606936 0.000000000 0.06936416
5  5 NONVEG NONVEG 0.5164835 0.005494505 0.12637363
6  6  SHRUB NONVEG 0.4480874 0.000000000 0.37704918
       TREE
1 0.1676647
2 0.5480226
3 0.1925134
4 0.3699422
5 0.3516484
6 0.1748634
```

For categorical models, this file contains the observed category for each location, the category predicted by majority vote, as well as one column for each category observed in the data, giving the proportion of trees that voted for that category.

To calculate the confusion matrix from the file we will use the observed and predicted columns. The `read.table()` function will convert columns containing character strings to factors. If the categories had been numerical, the `as.factor()` function can be used to convert the columns to factors. Because there may be catergories present in the observed data that are missing from the predictions (and vice versa), to get a symetric confusion matrix it is important to make sure all levels are present in both factors.

The following code will work for both numerical and character categories:

```
R> #
R> #these lines are needed for numeric categories, redundant for character categories
R> #
R> PRED$pred<-as.factor(PRED$pred)
R> PRED$obs<-as.factor(PRED$obs)
R> #
R> #adjust levels so all values are included in both observed and predicted
R> #
R> LEVELS<-unique(c(levels(PRED$pred),levels(PRED$obs)))
R> PRED$pred<-factor(PRED$pred,levels=LEVELS)
R> PRED$obs<- factor(PRED$obs, levels=LEVELS)
R> #
R> #calculate confusion matrix
R> #
R> CMX<-table( predicted=PRED$pred, observed= PRED$obs)
R> CMX

          observed
predicted  NONVEG OTHERVEG SHRUB TREE
  NONVEG      492       37   101  143
  OTHERVEG      7       17     8    1
  SHRUB        80       19   114    1
  TREE         90        3     2   76
```

To calculate the errors of Omission and Comission:

```
R> CMX.diag  <- diag(CMX)
R> CMX.OMISSION  <- 1-(CMX.diag/apply(CMX,2,sum))
R> CMX.COMISSION <- 1-(CMX.diag/apply(CMX,1,sum))
R> CMX.OMISSION

   NONVEG  OTHERVEG     SHRUB      TREE
0.2645740 0.7763158 0.4933333 0.6561086

R> CMX.COMISSION

   NONVEG  OTHERVEG     SHRUB      TREE
0.3635188 0.4848485 0.4672897 0.5555556
```

To calculate PCC:

```
R> CMX.PCC <- sum(CMX.diag)/sum(CMX)
R> CMX.PCC
```

```
[1] 0.5869018
```

To calculate Kappa:

```
R> CMX.KAPPA <- PresenceAbsence::Kappa(CMX)
R> CMX.KAPPA

           Kappa   Kappa.sd
NONVEG 0.2791541 0.02383285
```

The MAUC is calculated from the category specific predictions (the percent of trees that voted for each category):

```
R> VOTE <- HandTill2001::multcap(  response = PRED$obs,
                         predicted= as.matrix(PRED[,-c(1,2,3)]) )
R> MAUC  <- HandTill2001::auc(VOTE)
R> MAUC

[1] 0.7949483
```

Note, both the **PresenceAbsence** package and the **HandTill2001** package have functions named `auc()`. The `::` operator is used to specify that we are calling the `auc()` function from the **HandTill2001** package.

As in continuous and binary response models, the `model.diagnostics()` function creates a variable importance graph (Figure 30).

With categorical response models, the `model.diagnostics()` function alse creates category specific variable importance graphs, for example (Figure 31).

3.4.4 Comparing Variable Importance

In example 1 the `model.importance.plot()` function was used to compare the importance between two continuous Random Forest models, for percent cover of Pinyon and of Sage. Here we will compare the variable importances of the binary models from Example 2 with the categorical model we have just created in example 3 (Figure 32). We are examining the question "Are the same predictors important for determining vegetaion category as were important for determining species presence?" Keep in mind that to use `model.importance.plot()` the two models must be built from the same predictor variables. For example, we could not use it to compare the models from Example 1 and Example 3, because in Example 1 `"NLCD01_250"` was not included in the predictors.

```
R> opar <- par(mfrow=c(2,1),mar=c(3,3,3,3),oma=c(0,0,3,0))
R> model.importance.plot( model.obj.1=model.obj.ex2a,
                          model.obj.2=model.obj.ex3,
                          model.name.1="Pinyon",
                          model.name.2="VEGCAT",
                          type.label=FALSE,
                          sort.by="predList",
                          predList=predList,
                          scale.by="sum",
                          main="Pinyon Presence vs VEGCAT",
                          device.type="none",
                          cex=0.9)
R> model.importance.plot( model.obj.1=model.obj.ex2b,
```

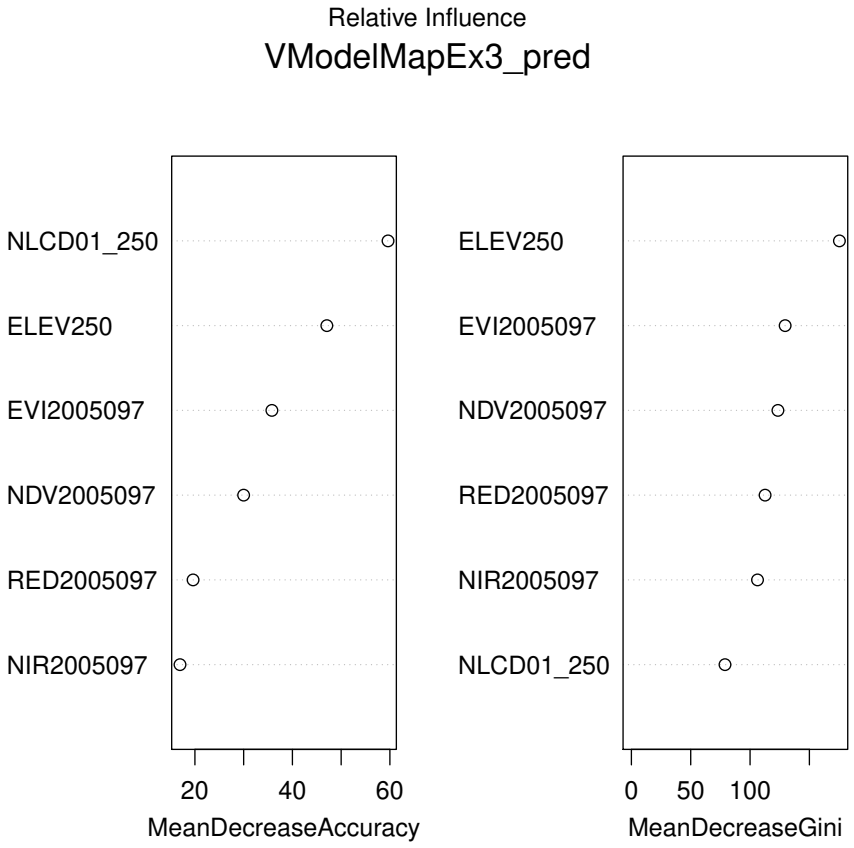

Figure 30: Example 3 - Overall variable importance graph for predicting vegetation category.

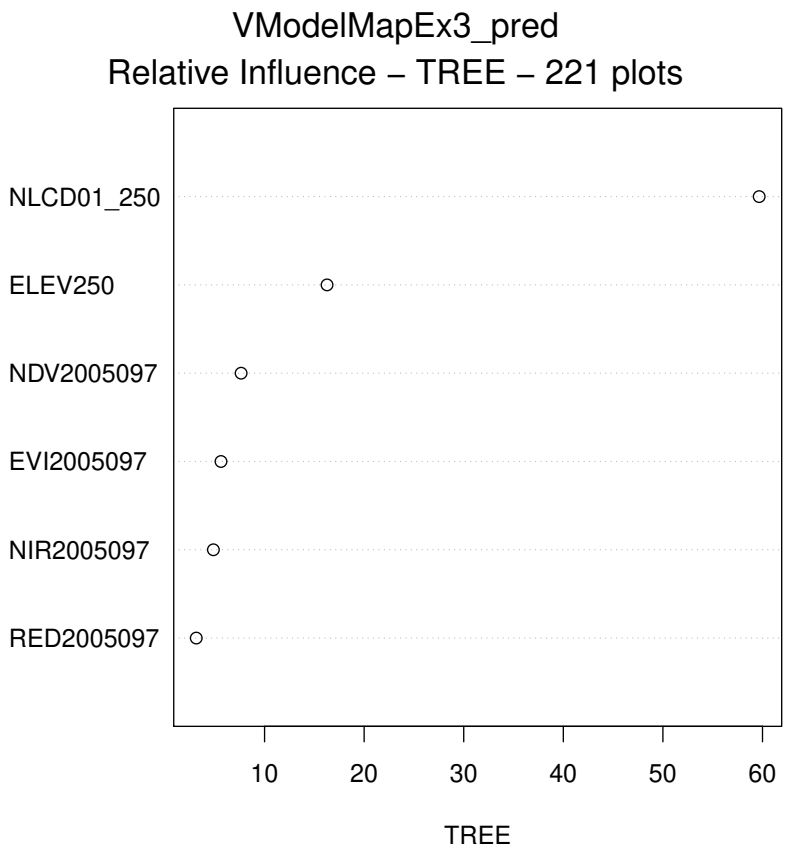

Figure 31: Example 3 - Category specific variable importance graph for vegetation category ”Tree”.

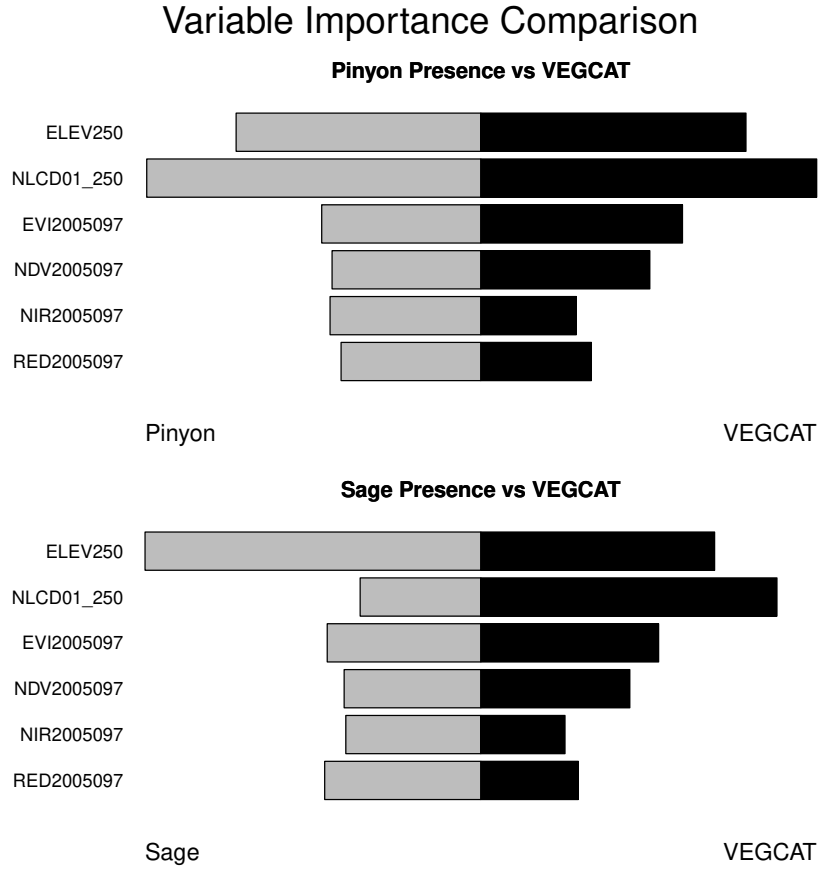

Figure 32: Example 3 - Comparison of variable importances between categorical model of vegetation category (VEGCAT) and binary models for Pinyon presence and Sage presence.

```
    model.obj.2=model.obj.ex3,
    model.name.1="Sage",
    model.name.2="VEGCAT",
    type.label=FALSE,
    sort.by="predList",
    predList=predList,
    scale.by="sum",
    main="Sage Presence vs VEGCAT",
    device.type="none",
    cex=0.9)
R> mtext("Variable Importance Comparison",side=3,line=0,cex=1.8,outer=TRUE)
R> par(opar)
```

With categorical models the `model.importance.plot()` function also can be usde to compare the variable importance between categories of the same model. The importance measure used for category specific importance is the relative influence of each variable, calculated by randomly permuting each predictor variable, and looking at the decrease in model accuracy associated with each predictor.

```
R> opar <- par(mfrow=c(2,1),mar=c(3,3,3,3),oma=c(0,0,3,0))
```

```
R> model.importance.plot( model.obj.1=model.obj.ex3,
                          model.obj.2=model.obj.ex3,
                          model.name.1="SHRUB",
                          model.name.2="TREE",
                          class.1="SHRUB",
                          class.2="TREE",
                          sort.by="predList",
                          predList=predList,
                          scale.by="sum",
                          main="VEGCAT - SHRUB vs. TREE",
                          device.type="none",
                          cex=0.9)
R> model.importance.plot( model.obj.1=model.obj.ex3,
                          model.obj.2=model.obj.ex3,
                          model.name.1="OTHERVEG",
                          model.name.2="NONVEG",
                          class.1="OTHERVEG",
                          class.2="NONVEG",
                          sort.by="predList",
                          predList=predList,
                          scale.by="sum",
                          main="VEGCAT - OTHERVEG vs. NONVEG",
                          device.type="none",
                          cex=0.9)
R> mtext("Category Specific Variable Importance",side=3,line=0,cex=1.8,outer=TRUE)
R> par(opar)
```

3.4.5 Interaction Plots

We will look at how the `model.interaction.plot()` function behaves with a categorical response variable. With categorical models, interactions can affect one prediction category, without influencing other categories. For example, if modelling disturbance type, it is possible that landslides might be influenced by an interaction between soil type and slope, while fires might be influenced by both variables individually, but without any interaction.

Therefore when calling `model.interaction.plot()` it is neccessary to specify a particular category. The function then will graph how the probability of that category varies as a function of the two specified predictor variables.

Here we look at the interaction between elevation and land cover class for two of our response categories (Figure 34, Figure 35).

```
R> model.interaction.plot( model.obj.ex3,
                           x="ELEV250",
                           y="NLCD01_250",
                           main=response.name,
                           plot.type="image",
                           device.type="pdf",
                           MODELfn=MODELfn,
                           folder=folder,
                           response.category="SHRUB")
R> model.interaction.plot( model.obj.ex3,
                           x="ELEV250",
                           y="NLCD01_250",
```

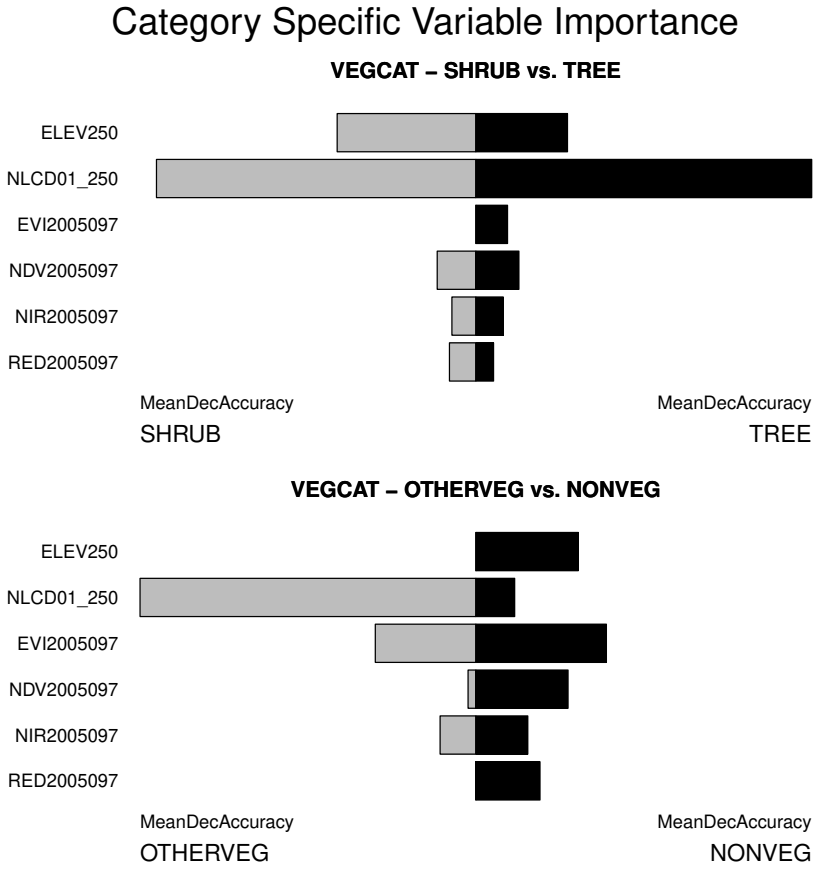

Figure 33: Example 3 - Category specific comparison of variable importances for the model of vegetation category (VEGCAT). National Land Cover DATA (NLCD) and elevation (ELEV) are the most important predictor variables for both TREE and SHRUB categories, though the relative importance of the remote sensing bands differs between these two categories. ELEV is relativly less important for the NONVEG and OTHERVEG categories, while ELEV is important for classifying OTHERVEG but relativly unimportant for the classifying NONVEG. In other words, if the model lost the information contained in NLCD and ELEV, the predictions for TREE and SHRUB categories would suffer, but there would be less of an effect on the prediction accuracy for NONVEG. The predictions for OTHERVEG would suffer if NLCD were removed from the model, but would lose relativly little accuracy if ELEV were removed.

```
                            main=response.name,
                            plot.type="image",
                            device.type="pdf",
                            MODELfn=MODELfn,
                            folder=folder,
                            response.category="NONVEG")
R>
```

3.4.6 Map production

The function `model.mapmake()` creates an ascii text files and an imagine image file of predictions for each map pixel.

```
R> model.mapmake( model.obj=model.obj.ex3,
                  folder=folder,
                  MODELfn=MODELfn,
                  rastLUTfn=rastLUTfn,
                  na.action="na.omit")
```

With categorical models, the `model.mapmake()` function outputs a map file, using integer codes for each category, along with a table relating these codes to the original categories. In this example `na.action` was set to `"na.omit"`, therefore pixels with factored predictors with values not found in the training data will be omited from the map.

Take a look at the codes:

```
R> MAP.CODES<-read.table( paste(MODELfn,"_map_key.csv",sep=""),
                          header=TRUE,
                          sep=",",
                          stringsAsFactors=FALSE)
R> MAP.CODES

  row category integercode
1   1   NONVEG           1
2   2 OTHERVEG           2
3   3    SHRUB           3
4   4     TREE           4
```

Column one gives the row number for each code. Column two gives the category names. Column three gives the integer codes used to represent each category in the map output. In this example the categories in the training data are character strings, and `model.mapmake()` assigned the integers 1 through the Number of categories. If the training data categories were already numeric codes, for example, `c(30,42,50,80)`, then `model.mapmake()` would keep the original values in the map output, and columns two and three would contain the same values.

Next we define a color for each category. The `colors()` function will generate a list of the possible color names. Some are quite poetical.

```
R> MAP.CODES$colors<-c("bisque3","springgreen2","paleturquoise1","green4")
R> MAP.CODES
```

Import the map output and transform the values of the map output to intergers from 1 to n (the number of map categories).

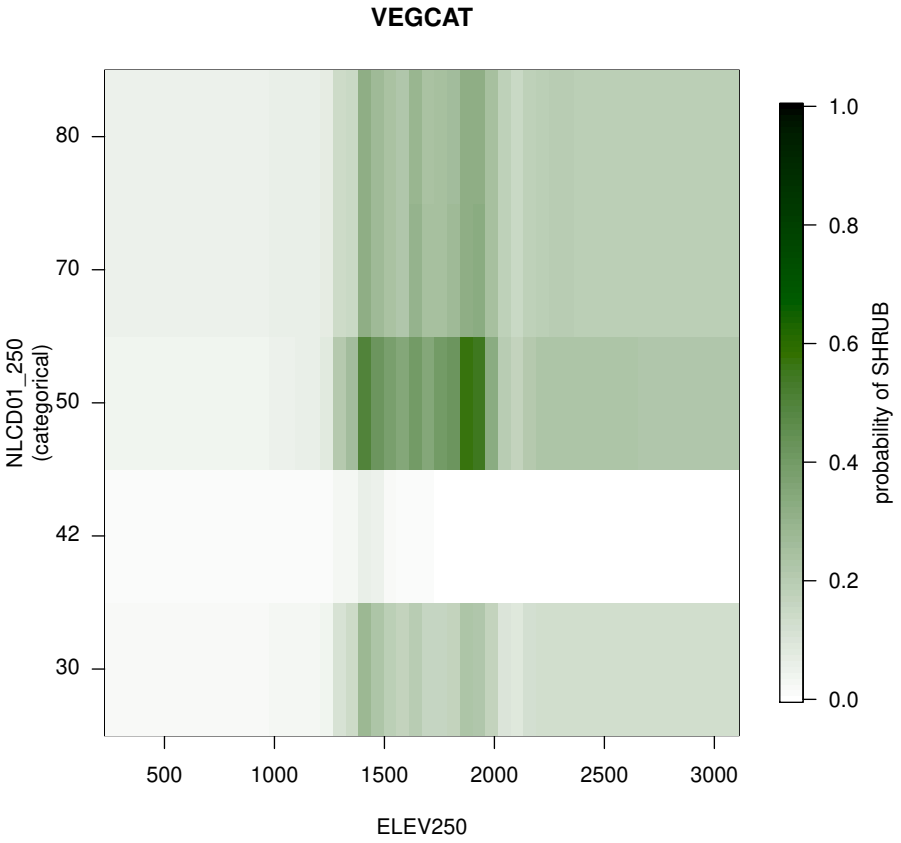

Figure 34: Example 3 - Interaction plot for SHRUB vegetation category (RF model), showing interactions between elevation and National Land Cover Dataset classes (ELEV250 and NLCD01_250). Image plot, with darker green indicating higher probability of being assigned to the specified category. Here we see the direct effect of NLCD class: SHRUB has a low probability of being assigned to landcover class 42. We also see a direct effect of elevation, with SHRUB having a slightly higher probability of being assigned to middle elevations. There is little evidence of interaction between these two predictors, since relationship of probability to elevation is similar for all land cover classes.

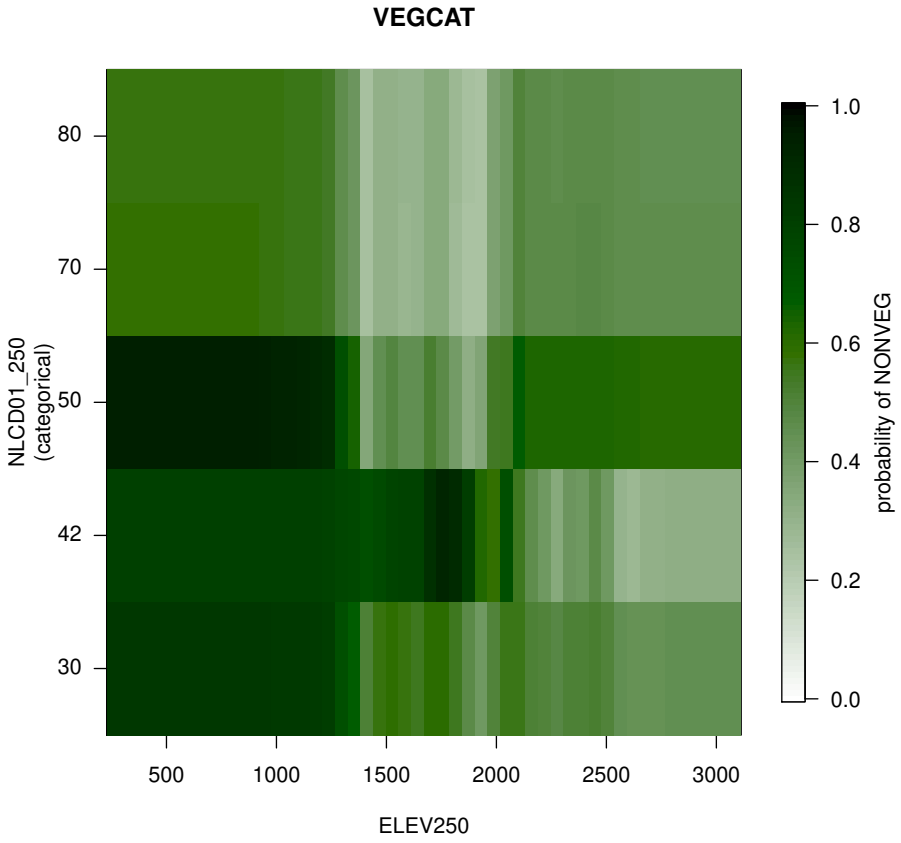

Figure 35: Example 3 - Interaction plot for NONVEG vegetation category (RF model), showing interactions between elevation and National Land Cover Dataset classes (ELEV250 and NLCD01_250). Image plot, with darker green indicating higher probability of being assigned to the specified category. NONVEG has a much higher chance of being assigned than SHRUB in all combinations of the two predictor variables, reflecting its higher prevalence in the training data (56% as opposed to 19%). NONVEG does show some interaction between landcover class and elevation. In all land cover classes NONVEG has a higher chance of being predicted at low elevations, but while in most land cover classes the probability goes down at elevations above 1400m, in land cover class 42 NONVEG matains a high chance of being predicted to over 2000m.

Note that here in example 3, where the categorical responses were character strings, the `model.mapmake()` function generated integer codes from 1 to n for the raster output. So for this example this step is not actually neccessary. However, if the categories in the training data had been unevenly spaced numeric codes, for example, `c(30,42,50,80)`, then `model.mapmake()` would keep these original numeric codes in the raster output. In such cases, to produce a map in R with the `image()` function (as opposed to viewing the image in a GIS environment) creating a new raster where the numeric codes are replaced with the numbers 1 to n makes assigning specific colors to each category simpler.

```
R> mapgrid <- raster(paste(MODELfn,"_map.img",sep=""))
R> integergrid <- mapgrid
R> v <- getValues(mapgrid)
R> v <- MAP.CODES$row[match(v,MAP.CODES$integercode)]
R> integergrid <- setValues(integergrid, v)
```

Produce the map (Figure 36).

```
R> opar <- par(mfrow=c(1,1),mar=c(3,3,2,1),oma=c(0,0,3,8),xpd=NA)
R> image(        integergrid,
                                  col = MAP.CODES$colors,
                                  xlab="",ylab="",xaxt="n",yaxt="n",
                                  zlim=c(1,nrow(MAP.CODES)),
                                  main="",asp=1,bty="n")
R> mtext(response.name,side=3,line=1,cex=1.2)
R> legend( x=xmax(mapgrid),y=ymax(mapgrid),
         legend=MAP.CODES$category,
         fill=MAP.CODES$colors,
         bty="n",
         cex=1.2)
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.

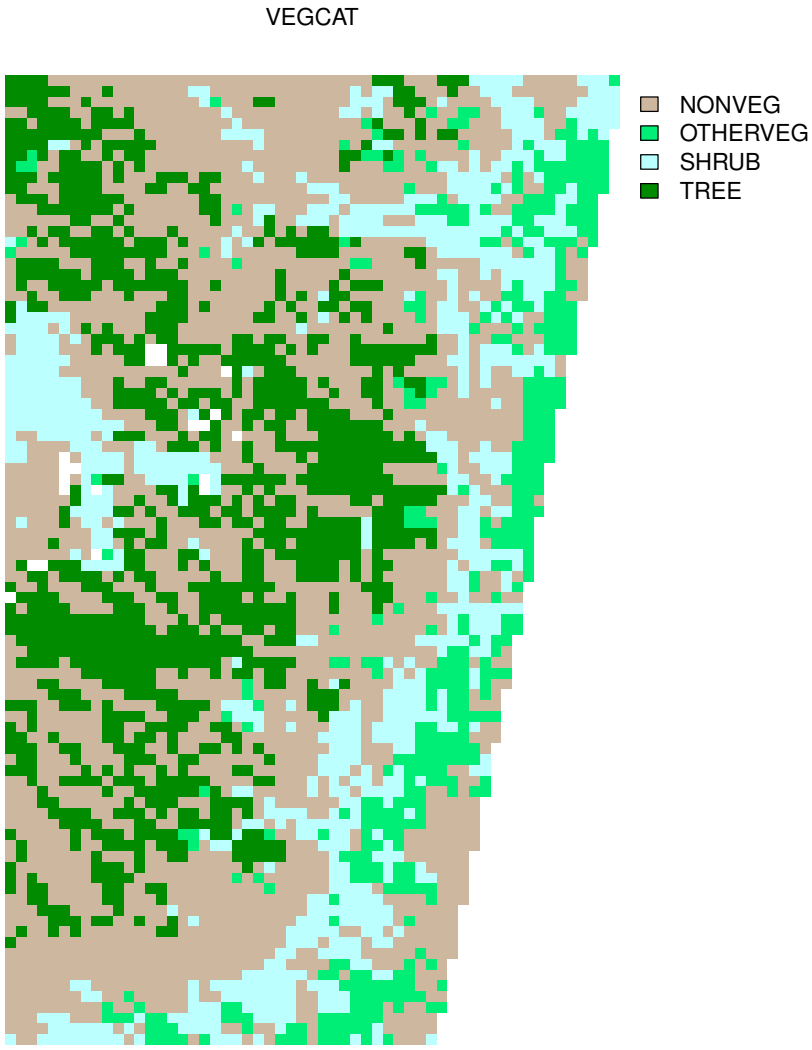

Figure 36: Example 3 - Map of predicted vegetation category (RF model). The white pixels found just below the center of this map idicate pixels with factored predictor variables that have values not found in the training data.

Appendices

Arguments for `model.build()`	
`model.type`	Model type: `"RF"` or `"QRF"` or `"CF"`.
`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`	Predictors from `predList` that are factors (i.e categorical).
`response.name`	Response variable used to build the model.
`response.type`	Response type: `"binary"` or `"continuous"`.
`unique.rowname`	Unique identifier for each row in the training data.
`seed`	Seed to initialize randomization to build stochastic models.
`na.action`	Specifies the action to take if there are `NA` values in the prediction data
`keep.data`	Should a copy of the predictor data be included in the model object. Useful if `model.interaction.plot` will be used later.
Random Forest Models:	
`ntree`	Number of random forest trees.
`mtry`	Number of variables to try at each node of Random Forest trees.
`replace`	Should sampling be done with or without replacement.
`strata`	A (factor) variable that is used for stratified sampling.
`sampsize`	For classification, if strata provided, sampling is stratified by strata. For binary response models, if argument strata is not provided then sampling is stratified by presence/absence.

Arguments for `model.diagnostics`	
`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 .
`diagnostic.flag`	Name of column in training that indicates subset of data to use for diagnostics.
`seed`	Seed to initialize randomization to build stochastic 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.

Arguments for `model.mapmake()`	
`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.
`keep.predictor.brick`	If `TRUE` then the raster brick containing the predictors from the model object is saved as a native raster package format file.
`map.sd`	Should maps of mean, standard deviation, and coefficient of variation of the predictions be produced: `TRUE` or `FALSE`. Only used if `response.type = "continuous"`.
`OUTPUTfn`	Filename for output raster file for map production. If `NULL`, "*modelfn*`_map.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.

D. J. Hand and R. J. Till. A simple generalisation of the area under the roc curve for multiple class classification problems. *Machine Learning*, 45(2):171–186, 2001.

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.

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.

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.