Manual for Remote Sensing Image Analysis in R:
Including Agnostic Image Analysis
Version 2

by

Rick Lawrence[*1], Shannon Savage[1], Emma Bode[1], John Long[2], Rachel Ulrich[1]

Revised 7 November 2019

* Corresponding author: PO Box 173120, Bozeman, Montana 59717, rickl@montana.edu

[1] Montana State University, Bozeman, Montana

[2] Northern State University, Aberdeen, South Dakota

1. **Introduction**

1.1 **An agnostic approach to image analysis**

More statistical methods are being used in remote sensing analyses than ever before, resulting in many cases in much higher accuracies than traditional methods (such as the maximum likelihood classifier for extracting thematic data and multiple linear regression for extracting continuous responses). Methods such as random forest (e.g., Lawrence et al., 2006) and support vector machines (e.g., Mountrakis et al., 2011) have found particular favor, even while not being available in most commercial image processing software packages.

The reliance on one or a few statistical methods, however, can lead to accuracies below what might otherwise be readily achieved. A study using 30 datasets and six classification methods (Lawrence and Moran, 2015), including random forest, C5.0, logistic model trees, support vector machines, multivariate adaptive regression splines, and classification tree analysis, demonstrated that each method, other than classification tree analysis, was superior to all other methods for at least one dataset. A recent unpublished experiment by one of the authors with a university remote sensing class had each student test five different statistical methods (from a suite of 12 methods) on both a classification dataset and a continuous response dataset. Most methods provided the best results with at least one dataset.

Every dataset has a distinctive, and often complex, data structure. Each statistical method has explicit or implicit assumptions with respect to data being analyzed. In-depth knowledge of the structure of a dataset and the assumptions underlying the wide range of statistical methods available could, theoretically, enable an analyst to select the best statistical method to apply to a particular dataset *a priori*. The reality is that neither the necessary knowledge with respect to the data nor the methods are commonly, if ever, fully available. An analyst is, therefore, best served by evaluating multiple methods, if such can be done with acceptable efficiency. We present in this paper one way to conduct this “agnostic” approach to image analysis, using the free statistical program R.

1.2 **Using R for image analysis**

There are many commercial and non-commercial image processing software packages, as well as many statistical and data analysis packages that can be used for remote sensing image analysis. We present an approach using R, but in no way purport to suggest that it is the only, or even the best, software solution. R does pose several advantages, however. First, the “raster” package in R provides the ability to readily handle a wide range of commonly used raster formats within an advanced statistical framework. Perhaps most important, a large number of statistical methods are available within R for creating predictive models. We are aware of 76 methods that can be used for classification or regression (e.g., continuous response modeling), an additional 91 methods that can only be used for classification, and an additional 50 that can only be used for regression. Many of these methods might not be appropriate for remote sensing applications and most likely have never been evaluated for that purpose. The availability in R of this wide range of methods (with more constantly being made available), however, makes it a compelling choice. A third important advantage we have found in using R is the “caret” package. The caret package provides an analyst with several features not commonly available, even with software that might otherwise have modern methods, such as random forest or support vector machines. The caret package function “train” tunes each model using a resampling approach, which should improve model performance when compared to accepting a method’s default parameters. The caret package also standardizes the format of model predictions to meet the needs of most remote sensing analyses. This is particularly important for some methods in R. Some neural network methods, for example, do not provide final predicted classes, but instead provide a probability matrix for each class for each observation. This might be useful for some analyses, but for most remote sensing applications, the desired output is the most probable class. The caret package standardizes predictions across methods to provide final class or continuous predictions, as appropriate.

Using R also comes with some important challenges. R is a command line programming environment, and it therefore requires a steeper learning curve than most image processing packages. We have attempted to alleviate that issue by providing code in this article to meet most needs, but some working knowledge of R is still necessary. This is particularly true because R is updated often, and these updates will sometimes result in the need to make adjustments, almost always minor, to prior code. Some knowledge of R coding is needed, therefore, if only to adjust prior code, such as that provided here, for potential changes. There are also likely differences in many datasets than those we have tested that might require code adjustments. Third, R is limited in terms of data size handling ability, although the raster package should fulfill many remote sensing needs; an analyst with a very large dataset might find that R is unable to handle it. Anecdotally, we have been able to handle four mosaicked Landsat 8 images without issue, while a colleague was unable to analyze a 1-m NAIP mosaic for the state of Montana (which requires all or part of 37 Landsat images, by way of reference). Finally, R can be notoriously slow to run code when compared to many other programming languages. Various approaches exist to alleviate this issue somewhat, but we have had very limited success with most approaches. An approach that has been useful is to run multiple instances of R when comparing multiple methods, with different methods running in different instances, Success with this approach depends on the number of cores and amount of physical memory in the analyst’s computer.

2. **R Workspace Setup**

R commands in the following sections are indented in courier new font. It should be possible to copy and paste these commands into R, but doing this directly might cause issues. R, for example, only recognizes straight quotes, not Word's "smart quotes". Copying commands first to a plain text editor, such as Microsoft Notepad, can ensure that unintended formatting does not get copied into R.

Workspace setup should include at least the following four steps: (1) set up a folder for your analysis; (2) set your workspace in R; (3) install needed libraries; and (4) load needed libraries. We recommend that you establish a folder on your computer solely for the analysis being conducted and include in the folder only the files being used in the analysis. A classification using a single image, with shapefiles for training and accuracy assessment data, should contain only those files (and possibly applicable R code). This directory will also receive the output from your analysis.

The following libraries need to be installed to conduct the analyses presented in this paper:

```
caret, e1071, maptools, pbkrtest, raster, rgdal,
shapefiles, sp, sf, dplyr
```

Additional libraries will be needed depending on the statistical methods being used. We have, to date, tested this code with the following methods: C5.0, classification and regression tree analysis, conditional inference random forest, cubist, *k*-nearest neighbors, learning vector quantization, linear discriminant analysis, multivariate adaptive regression splines, naïve Bayes, neural networks, partial least squares, random forest, regression trees with gradient boosting, ridge regression, stacked autoencoder deep neural network, stepwise linear regression, and support vector machines (with linear, radial, and polynomial kernels) . The following libraries should be installed to enable all of these methods (some can be excluded if a certain method is not being used, additional libraries might be needed for other methods not listed):

```
C50, class, Cubist, deepnet, earth, elasticnet, foba, kernlab,
kknn, klaR, MASS, mboost, nnet, party, pls, plyr,
randomForest, rpart, xgboost
```

Load the base libraries from the R command line using the following:

```
library(raster)
library(shapefiles)
library(sp)
library(rgdal)
library(caret)
library(e1071)
library(maptools)
library(sf)
library(dplyr)
```

Load these additional libraries depending on the methods used.

```
library(C50)
library(class)
library(Cubist)
library(deepnet)
library(earth)
library(elasticnet)
library(foba)
library(kernlab)
library(kknn)
library(klaR)
library(MASS)
library(mboost)
library(nnet)
library(party)
library(pls)
library(plyr)
library(randomForest)
library(rpart)
library(xgboost)
```

3. **Image preparation**

Imagery that is in an acceptable format (Table 1) can often be used as is. Special considerations might arise, however, when either one of the layers is categorical or when the image study area is not rectangular.

Predicting responses, whether for validating a model or for creating a predicted raster image, requires that all categorical values be represented in both the data used to create the model and the data being predicted. It is, therefore, necessary to ensure that both training and validation data include all categories for any categorical variables. For example, an aspect layer might include categories for north, west, south, east, and flat; it would be necessary for all five categories to be represented in training and validation data. Failure to do so will result in the inability to generate an accuracy assessment confusion matrix or generate an output predicted image.

Non-rectangular image study areas require a background value for areas outside the study area. A common practice is to set this value to zero; however, this practice creates two potential issues. True zero values (which can appear, for example, in NDVI data) cannot be differentiated from zero-designated background values. Ignoring zeros in the analysis will avoid confusion with the background, but will also result in the failure to predict output values where true zeros exist. Second, categorical layers, if present, will result in errors, because the background category of zero will not be present in the training data, resulting in prediction errors, per the preceding paragraph. We recommend, in this case, that an alternative background value be set that is not present in any raster layer, such as the largest value present in any raster layer plus 1. For example, if the raster image's largest value was an elevation of 2437 m, the background value could be reset to 2438 before bringing the image into R. The code we provide includes optional lines for designating such values as NA when producing output images (we assume that no such values will exist in training or validation data), which can then be removed from the analysis using na.action = na.omit as an argument.

4. **Importing Data**

Reference data can be in many different formats. We have included appropriate code for many common formats, including several shapefile arrangements and csv formatted text data (we recommend converting ASCII, spreadsheet, and text formats csv format; Excel Save As has this as an option, for example). Data may also be imported as a single dataset and randomly divided into training and validation, in which case the data should be imported using the code for training data and then divided using the code provided under instructions for importing validation data.

Our experience has resulted in a practice of combining all input raster data into a single, multi-layer file (we commonly use img file format). R includes code that should allow separate import of various raster data (e.g., a Landsat image and a separate digital elevation model) and stacking within R; however, we have never had success in doing so, receiving errors claiming different extents. The following sections provide sample code for importing data in various formats.

4.1 **Importing classification training data**

4.1.1 *Shapefiles, one shapefile for each class.* Each shapefile should be named with the name of the desired output class, followed by "train" (e.g., watertrain.shp). Lines that begin with “###” are comments that indicate what the following step in the code is doing but do not call any code.

```
### Get the names of the training shapefiles
train.files <- list.files(pattern="*train.*shp$")
### Read in the raster image.  Replace name of image file with
   the name of the image.
image <- "name of image file"
```

```
img <- stack(image)
### Create an object to receive the training data
train.full <- NULL
### This loop extracts the training data for each shapefile
   and adds it sequentially to the training dataset.
for (x in seq_along(train.files)) {
  ### Read in the a shapefile
  train.locations <- st_read(train.files[x])
  ### Extract values from the image for the shapefile area.
  train.predictors <- raster::extract(img, train.locations)
  ### Combine the data from multiple polygons in the
   shapefile.  If the data is in a point shapefile instead of
   polygons, skip this next line.
  train.predictors <- as.data.frame(do.call(rbind,
   train.predictors))
  ### Add a column for the class type and populate it.
  train.by.class <- as.data.frame(cbind(train.predictors,
   "types" = substr(train.files[x], 1, nchar(train.files[x])-
   9)))
  ### Combine the data with data from previous loops.
  train.full <- rbind(train.full, train.by.class)
  ### Make all your response variables numeric (categorical
   variables will be addressed later)
  train.full <- train.full %>% mutate_each(funs(as.numeric),
   starts_with(substr(image, 1, nchar(image)-4)))
}
```

4.1.2 *One shapefile with multiple classes.* This shapefile must have at least one field that identifies the classes (in the code below, the field is called “types”). The shapefile can be of any feature type, but different code must be used for points versus any other type.

```
### Read in the raster image.  Replace name of image file with
   the name of the image.
image <- "name of image file"
img <- stack(image)
### Read in the training shapefile.  Replace shapefile with
   the name of the shapefile.  The shapefile should have an
   attribute named "types" for class designations (e.g.,
   water).
train.locations <- st_read("shapefile")
### Create an object to receive the training data.
train.full <- NULL
### This loop extracts the training data for each class and
   adds it sequentially to the training dataset.
for (i in 1:length(unique(train.locations[["types"]]))){
  ### Sequentially extract the name for each unique class.
  train.type <- unique(train.locations[["types"]])[i]
  ### Sequentially create a spatial polygon object for each
   class.
  train.type.locations <- train.locations
   [train.locations[["types"]] == train.type,]
  ### ### Extract values from the image for the subset of the
   shapefile area.
  train.by.class <- raster::extract(img, train.type.locations)
  ### Combine the data from multiple polygons in the
   shapefile.  If the data is in a point shapefile instead of
   polygons, skip this next line.
  train.by.class <- as.data.frame(do.call("rbind",
   train.by.class))
  ### Add a column for the class type and populate it.
  train.by.class <- cbind(train.by.class, "types" =
   as.character(unique(train.locations[["types"]])[i]))
  ### Combine the data with data from previous loops.
  train.full <- rbind(train.by.class, train.full)
  ### Make all your response variables numeric (categorical
   variables will be addressed later)
  train.full <- train.full %>% mutate_each(funs(as.numeric),
   starts_with(substr(image, 1, nchar(image)-4)))
}
```

4.1.3 *Csv files, one csv file for each class.* Each csv file should be named with the name of the desired output class, followed by "train" (e.g., foresttrain.csv) and should only contain the coordinates for the training data, with columns headed x and y.

```
### Get the names of the training csv files.
train.files <- list.files(pattern="*train.*csv")
### Read in the raster image.  Replace image with the name of
 the image.
img <- stack("image")
### Create an object to receive the training data.
train.full <- NULL
### This loop extracts the training data for each csv file and
 adds it sequentially to the training dataset.
for (x in seq_along(train.files)) {
  ### Read in the csv file.
  train.locations <- read.csv(train.files[x])
  ### Set the values in the csv data to be coordinates.
  coordinates(train.locations) <- c("x","y")
  ### Extract values from the image for the point locations.
  train.predictors <- as.data.frame(raster::extract(img,
 train.locations))
  ### Add a column for the class type and populate it.
  train.by.class <- cbind(train.predictors, "types" =
 substr(train.files[x], 1, nchar(train.files[x])-9))
  ### Combine the data with data from previous loops.
  train.full <- rbind(train.full, train.by.class)
  }
```

4.1.4 *Csv file with training data already extracted and assembled.* Data can, of course, be extracted using alternative methods and programs. Data assembled outside of R that is to be used with the code in this article should have a column for the data from each band and a column headed "types" containing class names for each observation. Do not change this code to use the word "class" as the variable name for the classes, as this is a reserved word in R.

```
### Read in the training data, replacing the filename with the
 name of the csv file.
train.full <- read.csv("my-data-file.csv")
```

4.1.5 *Object-oriented datasets.* Data preparation for object-oriented analysis will depend on the software being used to create the objects. One approach is to convert the objects to a shapefile and then to use the resulting shapefiles in a training/validation role as described earlier. Another approach would be to export the data to a text file, which can then be imported using the directions for csv files. The object-oriented approach is most useful when the landscape can be partitioned into homogeneous groupings of pixels based on an auxiliary layer, such as agricultural landscapes delineated at the field level by a using a cadastral layer (Long et al., 2013).

4.2 **Importing classification accuracy assessment data**

Importing accuracy assessment data is, for the most part, the same as training data. The corresponding code is provided in the following sections, where the object names have been changed to reflect that accuracy assessment validation data is being prepared. We have removed comments, since the processes are the same as with the training data. An exception is provided for cases when accuracy assessment data is created by extracting a subset from a full dataset that was imported as training data.

4.2.1 *Shapefiles, one shapefile for each class.* Each shapefile should be named with the name of the desired output class, followed by "valid" (e.g., watervalid.shp).

```
valid.files <- list.files(pattern="*valid.*shp$")
valid.full <- NULL
for (x in seq_along(valid.files)) {
  valid.locations <- st_read(valid.files[x])
  valid.predictors <- raster::extract(img, valid.locations)
  ### If the data are in a point shapefile instead of polygons,
   skip this next line.
  ### valid.predictors <- as.data.frame(do.call(rbind,
   valid.predictors))
  valid.by.class <- as.data.frame(cbind(valid.predictors,
   "types" = substr(valid.files[x], 1, nchar(valid.files[x])-
   9)))
  valid.full <- rbind(valid.full, valid.by.class)
  valid.full <- valid.full %>% mutate_each(funs(as.numeric),
   starts_with(substr(image, 1, nchar(image)-4)))
}
```

4.2.2 *One shapefile with multiple classes.* This shapefile must have at least one field that identifies the classes (in the code below, the field is called “types”). The shapefile can be of any feature type, but different code must be used for points versus any other type. The comments below indicate which code to use for points and which to use for other types.

```
valid.locations <- st_read("shapefile")
valid.full <- NULL
for (i in 1:length(unique(valid.locations[["types"]]))){
  valid.type <- unique(valid.locations[["types"]])[i]
  valid.type.locations <- valid.locations
   [valid.locations[["types"]] == valid.type,]
  valid.by.class <- raster::extract(img, valid.type.locations)
  ### If the data is in a point shapefile instead of polygons,
   skip this next line.
  valid.by.class <- as.data.frame(do.call("rbind",
   valid.by.class))
  valid.by.class <- cbind(valid.by.class, "types" =
   as.character(category))
  valid.full <- rbind(valid.by.class, valid.full)
  valid.full <- valid.full %>% mutate_each(funs(as.numeric),
   starts_with(substr(image, 1, nchar(image)-4)))
}
```

4.2.3 *Csv files, one csv file for each class.* Each csv file should be named with the name of the desired output class, followed by "valid" (e.g., forestvalid.csv) and should only contain the coordinates for the validation data, with columns headed x and y.

```
valid.files <- list.files(pattern="*valid.*csv")
valid.full <- NULL
for (x in seq_along(valid.files)) {
  valid.locations <- read.csv(valid.files[x])
  coordinates(valid.locations) <- c("x","y")
  valid.predictors <- as.data.frame(raster::extract(img,
   valid.locations))
  valid.by.class <- cbind(valid.predictors, "types" =
   substr(valid.files[x], 1, nchar(valid.files[x])-9))
  valid.full <- rbind(valid.full, valid.by.class)
}
```

4.2.4 *Csv file with accuracy assessment data already extracted and assembled.* Data assembled outside of R that are to be used with the code in this article should have a column for the data from each band and a column headed "types" containing class names for each observation. Do not change this code to use the word "class" as the variable name for your classes, as this is a reserved word in R.

```
valid.full <- read.csv("my-data-file.csv")
```

4.2.5 *Random extraction of accuracy assessment data from imported training data.* This is a common practice. The statistical appropriateness of this approach is not addressed in this article, although analysts should be aware of issues related to use of non-independent data for validating results.

```
### Get the number of entries (rows) in the data.
x <- nrow (train.full)
```

```
### Create index numbers for extracting the training data.
 The example is set for 75% training, but this can be changed.
train.index <- sample (1:x, (x * 0.75))
### Create index numbers for the rest for validation.
valid.index <- setdiff (1:x, train.index)
### Extract the training data to a temporary object.
train.temp <- train.full[train.index,]
### Extract the validation data.
valid.full <- train.full[valid.index,]
### Rename the training data to match the rest of the code.
train.full <- train.temp
```

4.3 **Importing continuous training and validation data**

4.3.1 *Point shapefile with values as an attribute.* These shapefiles must have at least one field that identifies the continuous values (in the code below, the field is called “amount”).

```
### Create objects to receive the training and validation
  data.
train.full <- NULL
valid.full <- NULL
### Read in the raster image.  Replace name of image file with
  the name of the image.
image <- "name of image file"
img <- stack(image)
### This process extracts the training data within the
  shapefile.
### Read in the shapefile.  Replace the shapefile argument
  with the full name of the training shapefile.
train.locations <- st_read("shapefile")
### Extract values from the image for the shapefile area.
preidctors.train <- raster::extract(img, train.locations,
  df=TRUE)
### Add continuous amount to the "amount" column in the data
  frame directly from the shapefile.
train.full <- cbind(train.predictors, "amount" =
  train.locations$amount)
### Make all your response variables numeric (categorical
  variables will be addressed later)
train.full <- train.full %>% mutate_each(funs(as.numeric),
  starts_with(substr(image, 1, nchar(image)-4)))
### Omit the following steps if validation data will be
  randomly extracted from the full training dataset.
### Read in the shapefile.  Replace the shapefile argument
  with the full name of the validation shapefile.
valid.locations <- st_read("shapefile")
valid.predictors <- raster::extract(img, valid.locations,
    df=TRUE)
valid.full <- cbind(valid.predictors, "amount" =
    valid.locations$amount)
### Make all your response variables numeric (categorical
    variables will be addressed later)
valid.full <- valid.full %>% mutate_each(funs(as.numeric),
    starts_with(substr(image, 1, nchar(image)-4)))
```

4.3.2 *Csv files with location and value data.* The csv file should contain three columns. The following code assumes these are labeled x, y, and amount, and that the files are named "train.csv" and "valid.csv", respectively.

```
### Read in training text file.
train.locations <- read.csv("train.csv")
### Read in validation text file.
valid.locations <- read.csv("valid.csv")
### Read in the raster image.  Replace image with the name of
   the image.
img <- stack ("image")
### Identify the locational information.
coordinates(train.locations) <- c("x", "y")
### Extract predictor data from the image.
train.predictors <- raster::extract(img, train.locations)
### Extract the training values.
amount <- train.locations$amount
### Combine the response and predictor data.
train.full <- cbind(amount, train.predictors)
### Repeat the steps for the validation data.
coordinates(valid.locations) <- c("x", "y")
valid.predictors <- raster::extract(img, valid.locations)
amount <- valid.locations$amount
valid.full <- as.data.frame(cbind(amount, valid.predictors))
```

4.3.3 *Csv files with data already extracted and assembled.* Follow the instructions for importing classification data in this format. The response variable should be named "amount".

4.3.4 *Random extraction of validation data from imported training data.* Follow the instructions for random extraction of accuracy assessment data from imported classification training data.

5. **Agnostic Classification Analysis**

5.1 **Model building**

5.1.1 *Create your equation.* The equation can be simple or a bit complex depending on how the data were imported and the nature of the variables. Important matters to remember are (1) categorical variables that are represented by numbers must be preceded by "factor" (e.g., if the classes are designated 1, 2, 3 . . ., the equation response variable should be "factor(types)") and (2) variable names (and everything else in R) are case sensitive. The following are several potential ways to set your equation.

```
### If the data were imported using the commands above for
  shapefiles, all bands will be used as potential predictors,
  and none of the variables are categorical.
equation <- factor(types) ~ .
equation <- as.formula(equation)
### If there are four bands, and band 3 is categorical.  Use
  the "names" function to get the actual names of the
  variables and replace B1, B2, etc. with these names.
names(train.full)
equation <- factor(types) ~ B1 + B2 + factor(B3) + B4
equation <- as.formula(equation)
### If externally prepared data have been imported, the classes
  are designated by numbers, all bands will be used as
  potential predictors, and there are no categorical predictor
  variables.
equation <- factor(types) ~ .
equation <- as.formula(equation)
### If externally prepared data have been imported, the classes
  are designated by numbers, and there are four bands, with
  band 3 being categorical.
equation <- factor(types) ~ B1 + B2 + factor(B3) + B4
equation <- as.formula(equation)
```

5.1.2 *Compare classification models.* The first step in comparing methods is to determine which methods to compare. There are, as noted above, at least 167 methods for classification that can be used with the caret package in R, and it is unlikely that even the most agnostic analyst will want to evaluate them all. The full list can be found at http://topepo.github.io/caret/modelList.html, and for the following code the methods used should be designated using the "method Argument Value" listed there. The following sample code includes random forest (rf), support vector machines (with linear (svmLinear), radial (svmRadial), and polynomial (svmPoly) kernels), classification tree analysis (rpart), C5.0 (C5.0), *k*-nearest neighbors (kknn), extreme gradient boosting (xgbTree), and linear discriminant analysis/maximum likelihood (lda). Output is to a text file. This step can take considerable time to run, depending on the data size and the particular methods being compared, primarily because of the cross validation used by the caret train function to tune model parameters (svmPoly and especially xgbTree seem to take the longest time to run; in our experience, xgbTree can take 4-12 hours for datasets with 5,000 to 60,000 observations, obviously depending on each computer; these can be very good classifiers, however). It is very important, however, for many models that these parameters be tuned, because the default parameters in many cases will yield substantially inferior results. The output can be compared in terms of overall and class accuracies (along with full error matrices) to select the desired classification method for creating a final output image.

```
### Define which methods you want to analyze.  Add and/or
   subtract methods as desired.
mthd <- c("rf", "rpart", "svmLinear", "svmRadial", "svmPoly",
   "C5.0", "kknn", "xgbTree", "lda")
### Step through each method, saving the results in a text
   file.
for (x in mthd) {
  train.model <- train(equation, method = x, data = train.full)
  predict.model <- predict (train.model, newdata = valid.full)
  results <- confusionMatrix(predict.model, valid.full$types)
  capture.output(x, results, file = "results.txt", append =
   TRUE)
}
```

5.2 **Output a final image**

The raster package predict function is used to create the final output image. The following code outputs an unsigned, 8-bit, ERDAS Imagine img file, but this can be changed using the "format" and "datatype" parameters (Tables 1 and 2).

Table 1: Available file formats for the raster package predict function.

File type	Long name	default extension
raster	'Native' raster package format	.grd
ascii	ESRI Ascii	.asc
SAGA	SAGA GIS	.sdat
IDRISI	IDRISI	.rst
CDF	netCDF (requires ncdf4)	.nc
GTiff	GeoTiff (requires rgdal)	.tif
ENVI	ENVI .hdr Labelled	.envi
EHdr	ESRI .hdr Labelled	.bil
HFA	Erdas Imagine Images (.img)	.img

Table 2: Available datatypes for the raster package predict function.

Datatype definition	Minimum possible value	Maximum possible value
LOG1S	FALSE (0)	TRUE (1)
INT1S	-127	127
INT1U	0	255
INT2S	-32,767	32,767
INT2U	0	65,534
INT4S	-2,147,483,647	2,147,483,647
INT4U	0	4,294,967,296
FLT4S	-3.4e+38	3.4e+38
FLT8S	-1.7e+308	1.7e+308

```
### Recreate the desired model, replacing x with the value for
  the desired method (in quotation marks).
train.model <- train(equation, method = "x", data = train.full)
### Create the output image, changing the filename as desired.
   This code uses the clusterR function to implement parallel
   processing and increase the speed of the predictions (the
    first line sets this up for four cores; change this as
    desired).
beginCluster(4)
### Use the following line only if you have a designated NA
    value; insert the NA value you are using in place of x.
NAvalue(img) <- x
clusterR(img,raster::predict, args=list(model=train.model),
    filename="output", format="HFA", datatype="INT1U" ,
    overwrite=TRUE, na.action=na.omit)
endCluster()
```

6. **Agnostic Continuous Analysis**

6.1 **Model building**

6.1.1 *Create your equation.* Equation creation follows the same pattern as with classification, and that section should be reviewed. The response, of course, will never be categorical, but some predictor variables might be. The following are two ways to set your equation.

```
### If the data were imported using the commands above for
   shapefiles or externally prepared data were imported, all
   bands will be used as potential predictors, and none of the
   variables are categorical.
equation <- amount ~ .
equation <- as.formula(equation)
### If there are four bands, and band 3 is categorical.  Use
   the "names" function to get the actual names of the
   variables and replace B1, B2, etc. with these names.
names(train.full)
equation <- amount ~ B1 + B2 + factor(B3) + B4
equation <- as.formula(equation)
```

6.1.2 *Compare continuous models.* The first step in comparing methods, as with classifications, is to determine which methods to compare. There are at least 126 methods for regression analyses that can be used with the caret package in R. The full list can be found at http://topepo.github.io/caret/modelList.html, and for the following code the methods used should be designated using the "method Argument Value" listed there. The following sample code includes random forest (rf), support vector machines (with linear (svmLinear), radial (svmRadial), and polynomial (svmPoly) kernels), classification tree analysis (rpart), neural networks (nnet), *k*-nearest neighbors (kknn), partial least squares (pls), ridge regression (foba), cubist (cubist), conditional inference random forest (cforest), generalized linear model with step AIC feature selection (glmStepAIC), and multivariate adaptive regression splines (earth). Methods are evaluated by comparing predicted and observed validation data using a paired *t*-test (it might be necessary to change this if the data do not meet the assumptions underlying a *t*-test, for example when comparing proportions). Output is to a text file. This step, again, can take considerable time to run, depending on the data size and the particular methods being compared.

```
### Define which methods to analyze.  Add and/or subtract
   methods as desired.
mthd <- c("rf", "rpart", "svmLinear", "svmRadial", "svmPoly",
   "pls", "nnet", "kknn", "foba", "cubist", "cforest",
   "glmStepAIC", "earth")
### Step through each method, saving the results in Word file.
for (x in mthd) {
  train.model <- train(equation, method = x, data=train.full)
  predicted <- predict (train.model, newdata = valid.full)
  observed <- valid.full$amount
  results <- t.test(observed, predicted, paired=TRUE)
  capture.output(x, results, file = "results.txt", append =
   TRUE)
}
```

6.2 **Output a final image**

The raster package predict function is used to create the final output image. The following code outputs a floating point, ERDAS Imagine img file, but this can be changed using the "format" and "datatype" parameters (Tables 1 and 2).

```
### Recreate the desired model, replacing x with the value for
  the desired method (in quotation marks).
train.model <- train(equation, method = "x", data = train.full)
### Create your output image, changing the filename as desired.
   This code uses the clusterR function to implement parallel
   processing and increase the speed of the predictions (the
   first line sets this up for four cores; change this as
   desired).
beginCluster(4)
### Use the following line only if you have a designated NA
   value; insert the NA value you are using in place of x.
NAvalue(img) <- x
clusterR(img,raster::predict, args=list(model=train.model),
   filename="output", format="HFA", datatype="FLT4S",
   overwrite=TRUE, na.action=na.omit)
endCluster()
```

Acknowledgements

The project described in this publication was supported by Grant/Cooperative Agreement Number 08HQGR0157 from the United States Geological Survey. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the USGS. This material is based upon work that is supported by the National Institute of Food and Agriculture, U.S. Department of Agriculture, Hatch project under accession number 0185725.

References

Lawrence, R.L., and C.J. Moran, 2015. The AmericaView classification methods accuracy comparison project: A rigorous approach for model selection. Remote Sensing of Environment, 170:115-120.

Lawrence, R. L., Wood, S., & Sheley, R. (2006). Mapping invasive plants using hyperspectral imagery and Breiman Cutler classifications (random forest). Remote Sensing of Environment, 100, 356–362.

Long, J.A, Lawrence, R.L., Greenwood, M.C., Marshall, L. and Miller, P.R. (2013). Object-oriented crop classification using multitemporal ETM+ SLC-off Imagery and Random Forest. GIScience and Remote Sensing, 50(4), 418–436.

Mountrakis, G., Im, J., & Ogole, C. (2011). Support vector machines in remote sensing: a review. ISPRS Journal of Photogrammetry and Remote Sensing, 66, 247–259.