`yaImpute`: An `R` Package for *k*-NN Imputation

Nicholas L. Crookston
Rocky Mountain Research Station
USDA Forest Service
ncrookston@fs.fed.us

Andrew Finley
Department of Forest Resources
University of Minnesota
afinley@stat.umn.edu

Date of last revision: December 31, 2006

Abstract: `yaImpute` is used to impute attributes measured on some observations to observations where they are not measured. Attributes measured on all observations are called X-variables and those measured only a subset observations are Y-variables. *Reference* observations have X- and Y-variables and *target* observations have only X-variables. `yaImpute` picks *k-references* that are nearby the targets in an appropriate *p*-dimensional space and, when *k*=1, imputes the Y-variables from the closest reference to the target (when *k*>1 other logic is used). How the *p*-dimensional space is computed depends on the method used. Relationships among the X-variables are used for some methods (e.g. Euclidean and Mahalanobis) while the relationships between the X- and Y-variables within the reference data may be used define the distance measure for other methods (e.g. most similar neighbor and gradient nearest neighbor). The package also includes a new method for using the Random Forest regression algorithm to define distances. Tools that support building the imputations, evaluating the quality of the imputed results, and applying imputation logic to very large maps are included. The package is implemented in **`R`**.

Keywords: multivariate, imputation, Mahalanobis, Random Forests, correspondence analysis, canonical correlation, independent component analysis, most similar neighbor, gradient nearest neighbor, mapping predictions.

Introduction

Imputing missing values to observations is nothing new. Analysts and statisticians use many strategies for filling in missing information on specific observations so that subsequent data compilations will be based on reasonable data using methods that assume the data are complete. Most often, one or two attributes are estimated for an observation by using information from observations where they are not missing or by using ancillary data.

The problem addressed here is a little different. Here, by design, there are many observations where an entire set of attributes are not measured. An example from forestry serves to illustrate the problem. Frequently, forested land is mapped into polygons of suitably similar vegetation and of a reasonable size. Traditionally, the mapping was done

by interpretation of aerial photographs but more recently automated methods of processing image data have become available. During the mapping process, attributes about vegetation in each polygon is measured and recorded. However, planning of forest management activities often requires more detailed information than can be measured from these relatively inexpensive sources. The additional information is collected by visiting a sample of polygons on the ground. When the detailed information is needed for non-sampled polygons, attributes are imputed from those that are ground-examined to those that are not. A method for doing this kind of imputation was published by Moeur and Stage (1995) who called it Most Similar Neighbor sampling. Their approach found neighbors in a canonical space based on the relationship between attributes measured on all sample units and those derived from the detailed measurements. Later, Crookston and others (2002) published a Fortran-based program to do the imputation that supports some additional methods for finding similar neighbors. Others, notably Ohmann and Gregory (2002) developed another method based on canonical correspondence analysis. Their interest was focused on doing imputation at the 30 m pixel level in LandSat images but otherwise they are addressing the same basic problem albeit with a novel ordination approach. Others have been involved with this work for several years (Korhonen and Kangas 1997, Holmström *et al*. 2003, LeMay and Temesgen, 2005).

As budgets for taking field measurements have continued to fall, and new techniques of collecting better remotely sensed data have been developed, interest in imputation methods as described here have increased. Furthermore, alternative distance measures have been proposed leading to the need to provide a covenant system for testing methods against each other. The `yaImpute` package was developed to meet these needs. That it be open source and in the **R** (R Development Core Team, 2006) environment appealed to the author's sensibilities.

This document begins with providing a fundamental set of definitions and a simple example. An overview of the package contents, listing functions and their purposes follows. Formula and code fragments are provided that illustrate how the distance calculations are done. The paper closes with a real-world example. However, before we continue one important point bears stating. The quality of the imputations depends on many factors; chief among them is that there exists a relationship between the X-variables and the Y-variables. If there isn't one, suitably scaled random numbers could be used. The choice of which method to use and the trade offs as to how to improve results are topics generally beyond the scope of this document.

Definitions and a Basic Example

We identify attributes measured on all observations as X-variables and attributes measured on a subset of observations as Y-variables. *Reference* observations have X- and Y-variables and *target* observations have only X-variables. `yaImpute` picks *k*-references that are nearby the target in an appropriate *p*-dimensional space and, when k=1, imputes the Y-variables from the closest reference to the target (when k>1 other logic is used).

Information about the relationship between the X- and Y-variables among the references is ether ignored or used to condition the measurement of distances between references and targets, depending on the method used to define the *p*-dimensional space. Seven methods for finding neighbors are supported, six of which include the major step of finding the *k* minimum sum-of-squared differences in the X-attributes between a target observation and the reference observations. All methods seek *k*-NNs, where nearness is

measured by a distance. The methods are listed with details on how the calculations are done in the section titled Details.

The famous iris data (Anderson 1935) is used in a simple example. This data includes 5 attributes measured on 150 observations. For this example, we will pretend that Sepal.Length, Sepal.Width, and Petal.Length, are measured on all observations, and are therefore our X-variables, while Petal.Width and Species are measured on a subset and are our Y-variables. The random number seed is set so that you will get the same results as displayed here if you choose to run the example.

```
> require(yaImpute)
> data (iris)
> set.seed(1)
> refs=sample(rownames(iris),50)
> x <- iris[,1:3]
> y <- iris[refs,4:5]
```

Two basic steps are taken for a complete imputation. The first step is to find the neighbors relationships (function `yai`) and the second is to actually do the imputations (function `impute`). Function `yai` first classifies observations as reference or targets using the definitions described above. Function `impute` is used to actually impute Y-variables measured on reference observations to target observations. The `yaImpute` plot function automatically calls `impute` and then provides a plot of observed over imputed for the reference observations, where a reference observation other than itself is used as a near neighbor (Fig. 1).

```
> Mahalanobis <- yai(x=x,y=y,method="mahalanobis")
> plot(Mahalanobis)
```

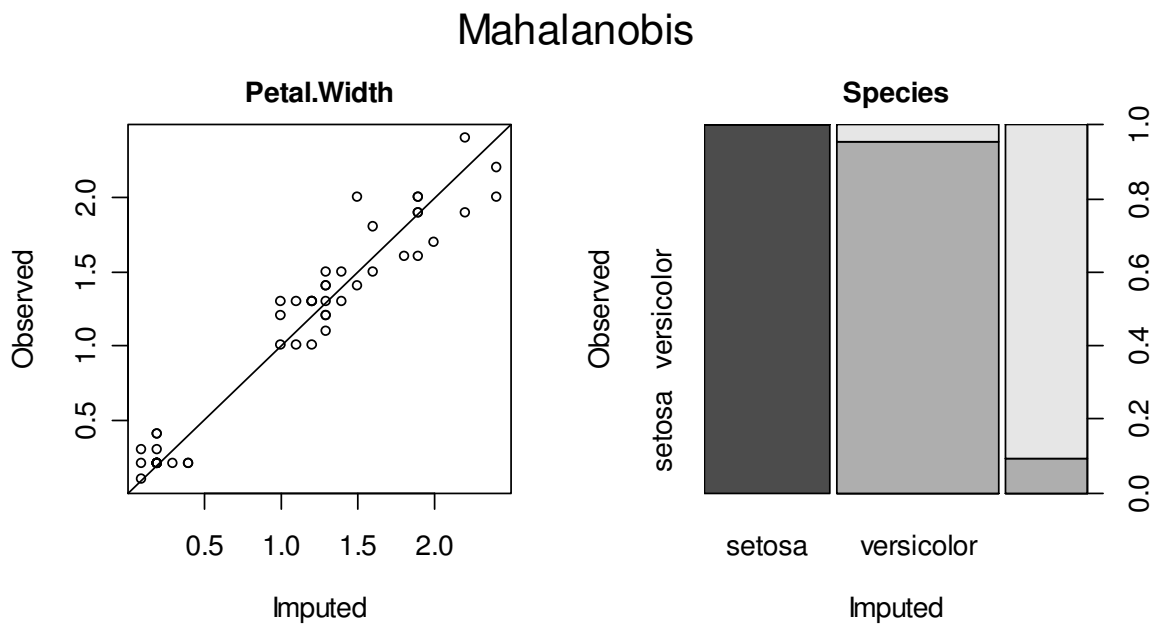

Figure 1: Output from running the plot command on the object generated with the simple code.

To see the entire list of imputations (including those for the target observations), the `impute` function is used alone. The reference observations appear in the result first, and the targets at the end of the result. Note that NA's are returned for the "observed" values (variable names with a .o appended), for reference observations (details on options controlling the behavior of the `impute` function are provided in the manual pages).

```
> head(impute(Mahalanobis))

    Petal.Width Petal.Width.o    Species  Species.o
40          0.2           0.2     setosa     setosa
56          1.2           1.3 versicolor versicolor
85          1.3           1.5 versicolor versicolor
134         1.6           1.5 versicolor  virginica
30          0.2           0.2     setosa     setosa
131         2.2           1.9  virginica  virginica

> tail(impute(Mahalanobis))

    Petal.Width Petal.Width.o    Species Species.o
144         2.2            NA  virginica      <NA>
145         2.4            NA  virginica      <NA>
146         2.0            NA  virginica      <NA>
147         1.6            NA versicolor      <NA>
149         2.4            NA  virginica      <NA>
150         1.8            NA versicolor      <NA>
```

Package Contents

Functions used to find neighbors, output results, and do the imputations:

`yai` finds *k*-NNs given reference and, optionally, target observations. This function is the main function in the package it turns an object of class "`yai`", described below.

`ann` provides access to approximate nearest neighbor search routines and is called by `yai`.

`impute` or `impute.yai` takes an imputation (object class `yai`) and an optional list of variables and returns a data frame of imputed values for specified variables. Observed values can be requested. In addition, new variables for reference, target, or both observations, are made for these variables using the neighbor relationships found in object.

`foruse` takes an imputation (object class `yai`) and returns a data frame with *k* columns. Row names are target observations identifications and the values are reference observation identifications for the *k* neighbors.

`newtargets` takes an imputation (object class `yai`) and a data frame of X-variables for new target observations and finds references for these new observations. A new `yai` object is returned.

Functions used to build maps:

`AsciiGridImpute` finds nearest neighbor reference observation for each point in the input grid maps and outputs maps of selected Y-variables in a set of output grid maps.

`AsciiGridPredict` provides an interface to `AsciiGridImpute` designed for use with models built using tools other than `yai`.

Functions used to display and evaluate results.

`compare.yai` takes several imputations (see `yai` and `impute.yai`) and provides a convenient display of the root mean square differences (see `rmsd.yai`) between observed and imputed values. Each column for the returned data frame corresponds to an imputation method and each row corresponds to a variable.

`cor.yai` takes an imputation object and computes the correlations between observed and imputed values. We do not believe correlation should be used to compare evaluate imputation results but decided to include this function because many people use correlation and this forum gives us an opportunity to present our position. The function outputs a warning against its use (see Diagnostics, below).

`errorStats` computes error statistics as proposed by Stage and Crookston (In press).

`mostused` returns a list of the most frequently used reference observations.

`notablyDistant` finds the *target* observations with relatively large distances from the closest *reference* observation. A suitable *threshold* is used to detect large distances is either specified or the function computes one.

`plot.yai` provides a matrix of plots of observed verses imputed for variables in an object created by `impute.yai`.

`plot.compare.yai` provides an informative plot of the data frame created from `compare.yai`.

`print.yai` outputs a summary of `yai` object (see below).

`rmsd.yai` takes an imputation object (see `yai` and `impute.yai`) and computes the root mean square difference between observed and imputed values.

Functions that directly support the use of `randomForest`:

`addXlevels` adds xlevels to `randomForest` objects.

`yaiRFsummary` builds summary data for method `randomForest`.

`yaiVarImp` plots Gini importance scores for method `randomForest`.

`whatsMax` finds the column that has the maximum value for each row, returns a data frame with two columns. The first is the column name corresponding to the maximum and the second is the maximum value.

Miscellaneous functions:

`unionDataJoin` takes several data frames, matrices, or any combination, and creates a data frame that has the rows defined by a union of all row names in the arguments and columns defined by a union of all column names in the arguments. The data are loaded into this new frame where column and row names match the individual inputs. Duplicates are tolerated with the last one specified being the one kept. NAs are returned for combinations of rows and columns where no data exist. A warning is issued when a column name is found in more than one data source.

`vars` takes an imputation object (see `yai`) and returns a list of X-variables by calling function `xvars` and a list of Y-variables by calling function `yvars`.

Data:

`TallyLake` is data from Tally Lake Ranger District, Flathead National Forest, Montana, USA.

`MoscowMtStJoe` is data from the Moscow Mountain area and the St. Joe Woodlands, northeast of Moscow, Idaho, USA.

Classes:

`yai` is a list returned by function `yai` that contains elements as listed in the manual entry for the package, for of special note here are 2 pairs of data frames. `neiDstTrgs` holds the distances between a target observations (identified by row names) and the *k* reference observations. There are k columns. `neiIdsTrgs` is a corresponding data frame of reference identifications. `neiDstRefs` and `neiIdsRefs` are counterparts for references.

`impute.yai` is a data frame of imputed values. The row names are target observation identifications and the columns are variables (X-variables, Y-variables, both, or new variables supplied by the call to `impute.yai`). When observed values are included, additional variables are included that have .o appended as a suffix to the original name. An attribute is attached with the scaling factors for each variable that is used in computing scaled *rmsd*.

`compare.yai` is a data frame of root mean square differences (scaled) values. Rows are variables and columns correspond to each imputation result passed as arguments.

Details

`yaImpute` offeres several methods for finding neighbors. For all methods except `randomForest`, which is discussed below, the *k* nearest neighbors for target observation *i* is defined as the set of indexes to the *k*-minimum distances to reference observations. Squared distance between observation *i* and *j* is defined as:

$$d^2_{ij} = (\mathbf{X}_i - \mathbf{X}_j)\ \mathbf{W}\ (\mathbf{X}_i - \mathbf{X}_j)'$$

where

$\mathbf{X}_i$ is the vector of X-variables for the i^{th} target observation,
$\mathbf{X}_j$ is the vector of X-variables for the j^{th} reference observation, and
W is a weight matrix.

Methods differ in how **W** is defined (Table 1). With method *raw* distance is computed in the unaltered X space, with method *euclidean* distance is computed in a normalized X space, with method *mahalanobis* distance is computed in its namesakes space, with method *ica* distance is computed in a space defined using Independent Component Analysis (package fastICA, Marchini *et al.* 2006), with methods *msn* and *msn2* (Moeur and Stage 1995, Crookston *et al*. 2002) distance is computed in projected canonical spaces, and with method *gnn* (Ohmann and Gregory 2002) distance is computed using a projected ordination of X's found using Canonical Correspondence Analysis (package vegan, Oksanen *et al*. 2005). In the last method, `randomForest` (Breiman 2002, Liaw and Wiener 2002), observations are considered similar if they tend to end up in the same terminal nodes in a suitably constructed collection of classification and regression trees. The distance measure is one minus the proportion of trees where a target observation is in the same terminal node as a reference observation. Similarly to the other methods, *k*-NNs are the *k* minimum of these distances. Notable advantages of method `randomForest` are first that it is non-parametric and second that the attributes can be a mixture of continuous and categorical variables. The other methods require continuous measures where categorical variables are transformed to some continuous space. A third advantage is that the data can be rank-deficient, having many more variables than observations, colinearities, or both.

In all methods, finding the exact minimums involves time consuming searches between every target and all the reference observations. The package includes a much faster approximate nearest neighbor (ANN) search for use when appropriate and when the method includes sum-of-squared differences as the distance measure, which is not the case for method `randomForest`.

Table 1: Summary of how **W** is computed.

Method	Value of **W**
Raw	Identity matrix, **I**
Euclidean	Inverse of the diagonal variances in X
Mahalanobis	Inverse of the covariance matrix in X
ica	**K Ω ' K Ω**, where **Ω** corresponds to what fastICA calls **W** and **K** is defined as is done in fastICA.
msn	$\mathbf{\Gamma \Lambda^2 \Gamma'}$, where **Γ** are the canonical vectors corresponding to the X's found by a canonical correlation analysis between X's and Y's, and **Λ** are the canonical correlations.
msn2	$\mathbf{\Gamma \Lambda (I - \Lambda^2)^{-1} \Lambda \Gamma'}$
gnn	**Θ**, weights assigned to environmental data in canonical correspondence analysis.
`randomForest`	not applicable

For methods *msn* and *msn2*, a question arises as to the number of canonical vectors to use in the calculations. One option is for the user to set this number and indeed that can be done in this package. Another option is to fine those canonical vectors that are significantly greater than zero and use them all. Rao (1973, p. 556) defined an approximate F statistic that is the basis for a test of the hypothesis that the current and all smaller canonical correlations are zero in the population. Gittins (1985, p 57) notes that Λ varies in the range from zero to one and if the value is sufficiently small the conclusion is drawn that the X- and Y-variables are linearly dependent. It turns out that if the first row is linearly dependent, we can test the second, as it is independent of the first. If the second likelihood ratio is significantly small we conclude that the X- and Y-variables are linearly dependent in a second canonical dimension. The tests proceed for all non-zero canonical coefficients until it fails signifying that the number of correlations that are non-zero in the population corresponds to the last coefficient that past the test. Obviously, the test requires specification of a *p* value, set as 0.05 in `yai`. The `yai` function contains code that computes the F statistic following the formulas found in SAS (SAS 2000).

For method `randomForest` a distance measure based on Breiman's (2001) proximity matrix is used. This matrix has a row and column for every observation (all *reference* plus *target* observations). The elements contain the proportion of trees where observations are found in the same terminal nodes. Since every observation is in the same terminal node as itself, the diagonal elements of this matrix have the value 1. Breiman (2001) has shown that the proximities are a kind of Euclidean distance between observations and uses them as a basis for imputing values to observations were some variables are not measured.

In `yaImpute`, the distance between observations is defined to be the compliment of the proximities. The diagonal elements are all zero meaning that every observation is zero distance from itself. To find the *k*-NN for an observation, a column from this that corresponds to a *target* observation is extracted that includes only *reference* observations. The vectors sorted and the row identifiers corresponding to the *k* smallest distances.

Note that the proximity matrix is often too large to store and furthermore, we don't need proximities between target observations, only between reference and targets and among references to support some kinds of validation. Therefore, the `yaImpute` code stores a much smaller matrix called the nodes matrix. This matrix is *n* x *ntree* where *ntree*

is the number of trees. The elements of this matrix are terminal node identifications. The node matrix is partitioned into two matrices, one each the reference and target observations. When finding *k*-NNs, only the needed proximities are computed thereby avoiding the allocation of a potentially huge matrix and with saving computer time by not computing values that are not needed.

The random forests algorithm implemented in R (package `randomForest`, Liaw and Wiener 2002) can currently be used to solve unsupervised classification (no Y-variable), regression on a single Y, and classification on a single Y so long the number of levels is 32 or less.

In `yaImpute` we have extended the `randomForest` package to serve our needs, as follows. First, in unsupervised classification the idea of making a "prediction" for a value for Y is nonsense and therefore not allowed. In `yaImpute`, however, we forced the `randomForest` to make a prediction only because we want to save the nodes matrix that results from attempting a prediction. Second, the `randomForest` package implementation does not save the nodes matrix when regression is being done unless the proximity matrix is also allocated and saved. This is not the case when classification is done. To get around this restriction, `yaImpute` converts regression problems into classification problems by creating classes along the continuous scale of the Y variable. This approach seems to work but it will be abandoned in future versions by rewriting parts `randomForest` package as needed to properly use regression for continuously measured Y's.

Lastly, we devised an experimental way to handle multivariate Y's. The method is to build a separate forest for each Y and then join the nodes matrices for each. Each forest is conditioned to reduce the classification error for a given Y and the proximities reflect choosing X-variables and split points that best accomplish that goal. If two or more Y-variables are used, the joint proximities reflect the joint set of X-variables and split points found to achieve the multiple goals. Since some Y-variables may be more important than others, users are allowed to specify a different number of trees in the forests corresponding to each Y-variable.

Diagnostics

There is a large need to develop techniques useful for diagnosing whether or not a given application of imputation is satisfactory for a specific purpose, or whether one method of computing distance results in better results than another. We anticipate that this field will progress quickly. `yaImpute` includes functions to plot results from a given run, compute root mean square differences, compare these differences and plot them, and so on. It also provides function `errorStats` to compute the statistics proposed by Stage and Crookston (In press).

The package includes a function to compute the correlations between observed and imputed (function `cor.yai`) even though we do not believe correlation should be used, preferring root mean square difference (function `rmsd.yai`). We recommend against using correlation as a measure of the degree of association between observed and imputed values because correlation can be manipulated by how the sample is distributed along the scale of X-variables (Warren 1971) and root mean square difference (rmsd), and standard error of imputation (SSI, see Stage and Crookston (In press) and function errorStats) are not biased by distribution of the sample. Furthermore, correlation is usually viewed as measuring the degree of association between two variables, say one X

with one Y. When used to measure association in imputation, it is used to measure the degree of association between paired observations of a single Y. While this may be an interesting statistic, it is easy to forget that it does not have the same statistical properties as those attributed to the relationship between the population attribute ρ and its sample estimate r.

In regression, R^2 is used to measure the degree of association between a predicted value of a given variable and an observed value. As the regression function gets closer to the true relationship, the value of R^2 approaches 1. Lets say that we have a perfectly true regression, $y = f(\boldsymbol{X})$. If we used this formula in a NN-style imputation, we would not be imputing the predicted value of *y* (its perfect estimate), we would be imputing a value of *y* measured on a second (nearby) sample from the population. If every member of the population has a different value of *y*, the correlation between observed an imputed would never be perfect, even if the regression used to order samples near each other were a perfect predictor, and if the sample were actually a complete census!

Stage and Crookston (In press) changed the word "error" to "difference" in defining root mean square difference (`rmsd`) because the value as used in imputation includes different components of error than the value (`rmse`) used in a regression context and therefore it has different statistical properties. Reconsider the case described above—`rmse` will approach zero as the regression approaches the true function, but `rmsd` can never be zero.

Another diagnostic is to find *notably* distant target observations (function `notablyDistant`). These are observations that are farther from the closest reference observation than is typical of distances between references. The cause may be that they are outside the range of variation of the references or because they fall in large gaps between references.

Given a threshold distance, it is a simple job to identify the notably distant target observations. The question then becomes, what is a reasonable threshold distance?

For all the distance methods except `randomForest`, it appears that the distribution of distances among references follows a lognormal distribution. Following that assumption, a threshold is defined as the distance corresponding to the fraction of distances that account for the *p* proportion of the properly parameterized lognormal probability density function. When `randomForest` is used, the distribution of distances is assumed to follow the Beta distribution. Alternatively, users may specify the threshold, perhaps by inspecting the frequency distribution and choosing the threshold visually. We used Evans *et al*. (2000) for formulas for computing the parameters of these distributions.

Real-world Example

To illustrate the tools in `yaImpute` we present a preliminary analysis of the Moscow Mountain St. Joe Woodlands (Idaho, USA) data, originally published by Hudak *et al*. (2006). The analysis is broken into two major steps. First, the reference observations are analyzed using several different methods, the results compared, and a selection of the best model is made. Second, imputations are made using ASCII grid map data as input and the maps are displayed. Note that these data are actively being analyzed by the research team that collected the data and this example is not intended to be a final result.

We start by building `x` and `y` data frames and running four alternative methods.

```
> require (yaImpute)
```

```
> data(MoscowMtStJoe)
> x=MoscowMtStJoe[,c("EASTING","NORTHING","ELEVMEAN","SLPMEAN",
+                    "ASPMEAN","INTMEAN","HTMEAN","CCMEAN")]
> x[,5]=(1-cos((x[,5]-30)*pi/180))/2
> names(x)[5]="TrASP"
> y=MoscowMtStJoe[,c(1,9,12,14,18)]

> mal <- yai(x=x, y=y, method="mahalanobis", k=1)
> msn <- yai(x=x, y=y, method="msn", k=1)
> gnn <- yai(x=x, y=y, method="gnn", k=1)
> ica <- yai(x=x, y=y, method="ica", k=1)
```

Method `randomForest` works best when there are few variables and when factors are used rather than continuous variables. The `whatsMax` function is used to create a data frame of containing a list of the species of maximum basal area, and two other related variables.

```
> y2=cbind(whatsMax(y[,1:4]),y[,5])
> names(y2)=c("MajorSpecies","BasalAreaMajorSp","TotalBA")
> rf <- yai(x=x, y=y2, method="randomForest", k=1)
> head(y2)

  MajorSpecies BasalAreaMajorSp   TotalBA
1      PSME_BA        47.716568 47.941832
2      ABGR_BA        20.904731 59.307392
3         zero         0.000000 77.123193
4      ABGR_BA         2.079060  3.740631
5      ABGR_BA        22.814781 67.938562
6      THPL_BA        11.129221 32.982188

> levels(y2$MajorSpecies)

[1] "ABGR_BA" "PIPO_BA" "PSME_BA" "THPL_BA" "zero"
```

The `plot` command is used to plot the observed over imputed values for the variables used in the `randomForest` result (Fig. 2).

```
> plot(rf)
```

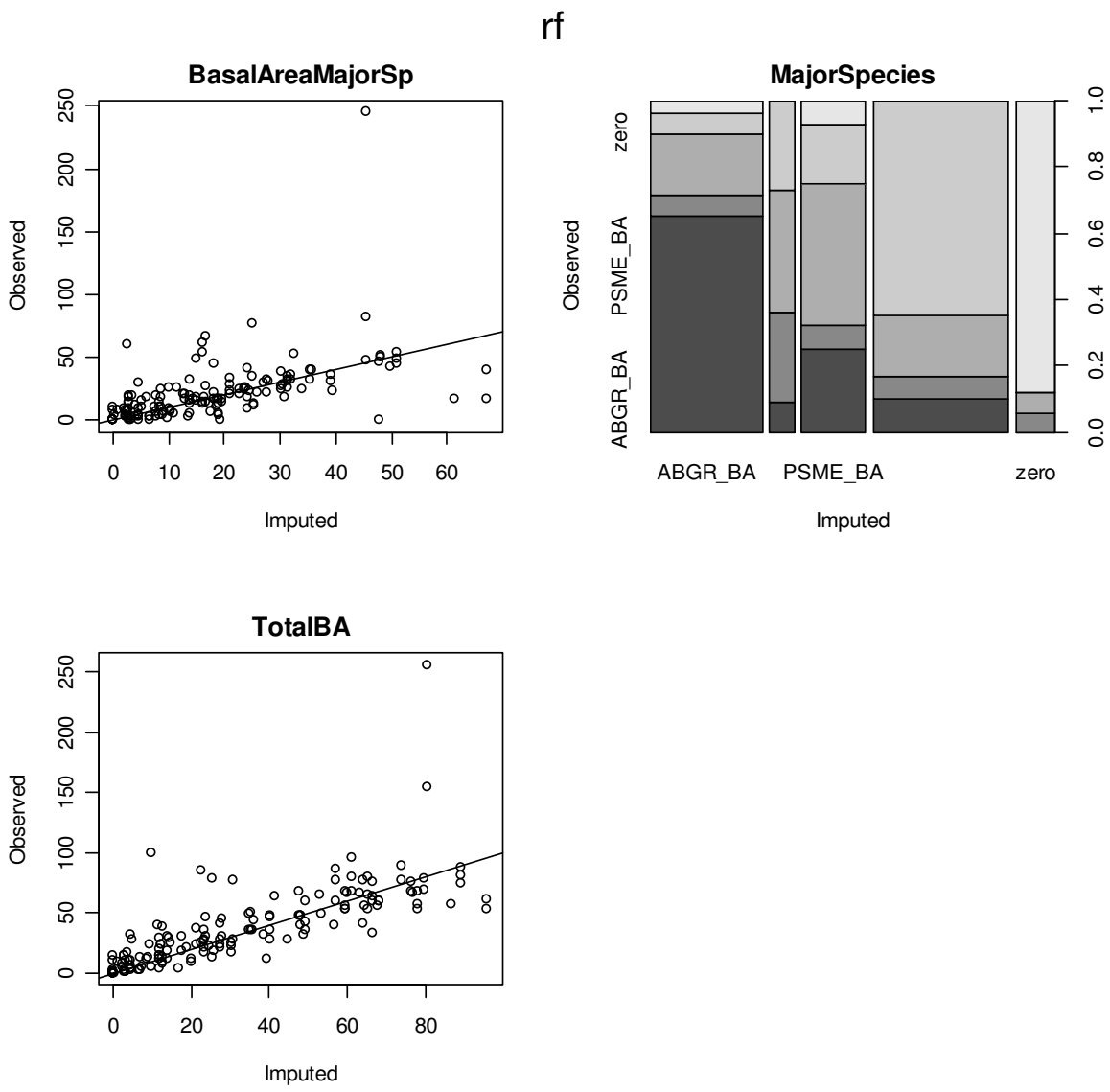

Figure 2: Plot of the `yai` object generated with `method=randomForest`.

However, the variables used to build this result are not those that are of interest. To make the ultimate comparison, the original Y-variables are imputed using the neighbor relationships in object `rf`, and then a comparison is built and plotted (Fig. 3) for all the methods:

```
> rfImp <-impute(rf,newdata=y)
> rmse=compare.yai(mal,msn,gnn,rfImp,ica)
> apply(rmse,2,mean)

  mal.rmsdS   msn.rmsdS   gnn.rmsdS rfImp.rmsdS   ica.rmsdS
  1.2058184   1.0738770   1.1185645   0.9892645   1.1198709

> plot(rmse)
```

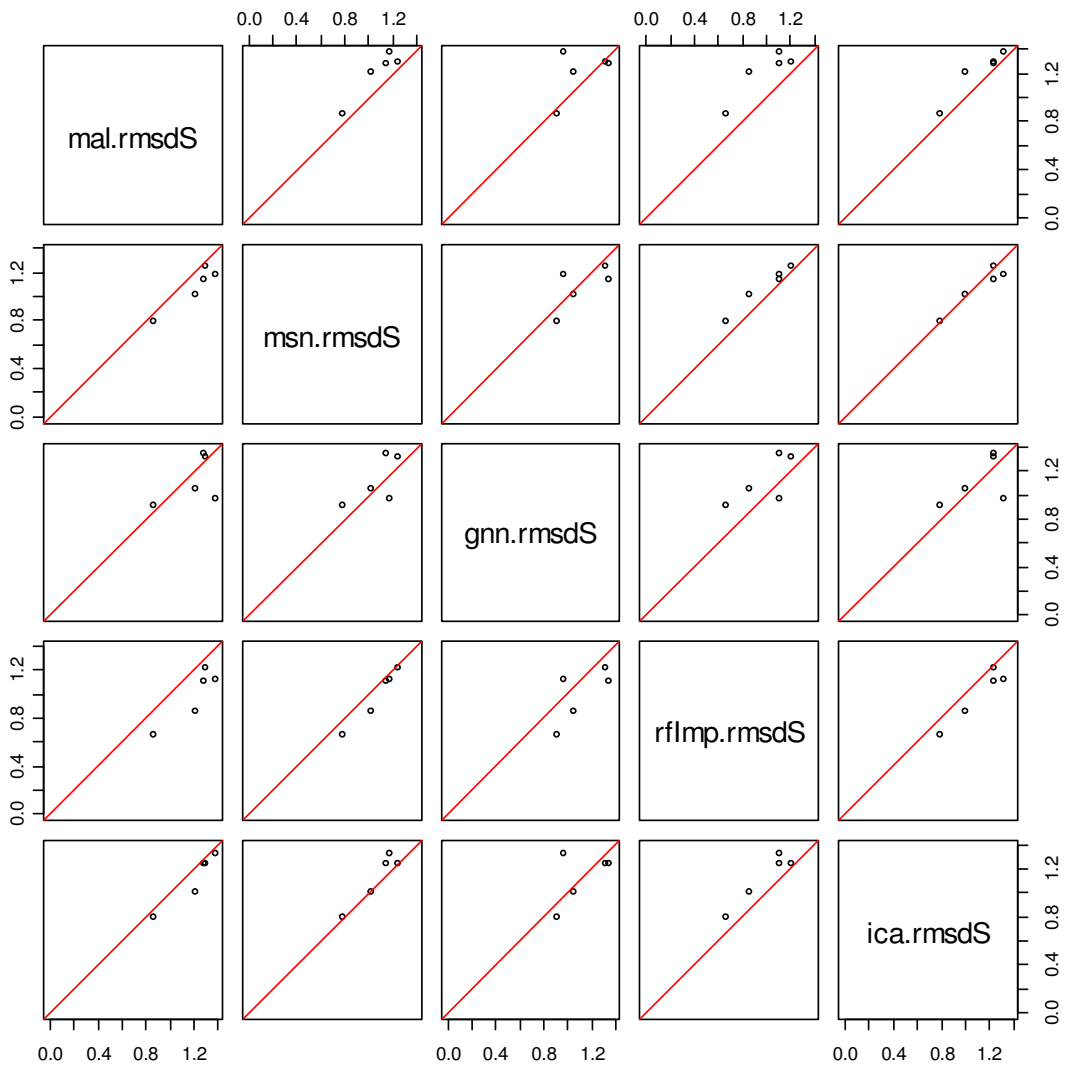

Figure 3: Comparison of the scaled `rmsd` for each method. Most of the values for method `rfImp` (imputed Y-variables build using method `randomForest`) are below the 1:1 line indicating that they are generally lower than those for other methods.

The steps presented so far can be repeated using the data available in the package distribution. However, following steps require that ASCII grid maps of the X-variables be available and they are not part of the distribution. The following commands are used to create the input arguments for function `AsciiGridImpute` to build output maps of the imputed values. Note that object `rf` was built using data transformed from that in the original set of Y-variables. Therefore, the original `Y` data frame is passed to function `AsciiGridImpute` as ancillary data.

```
> xfiles=list(CCMEAN="canopy.asc",ELEVMEAN="dem.asc",
+    HTMEAN="heights.asc",INTMEAN="intense.asc",SLPMEAN="slope.asc",
+    TrASP="trasp.asc",EASTING="utme.asc",NORTHING="utmn.asc")
> outfiles=list(ABGR_BA="rf_abgr.asc",PIPO_BA="rf_pipo.asc",
+    PSME_BA="rf_psme.asc",THPL_BA="rf_thpl.asc",
+    Total_BA="rf_totBA.asc")
> AsciiGridImpute(rf,xfiles,outfiles,ancillaryData=y)
```

Package `sp` (Pebesma and Bivand 2005) contains functions designed to read and manipulate ASCII grid data and are used to plot part of the total image of both X- and Y-variables (Figs. 4 and 5).

```
> require(sp)
> elev     =read.asciigrid("dem.asc")    [100:450,400:700]
> canopy   =read.asciigrid("canopy.asc") [100:450,400:700]
> TrAsp    =read.asciigrid("trasp.asc")  [100:450,400:700]
> intensity=read.asciigrid("intense.asc")[100:450,400:700]

> par(mfcol=c(2,2),plt=c(.05,.95,.05,.85))
> image(elev,col=hcl(h=140,l=seq(100,0,-10)))
> title("Elevation (DEM)")
> image(canopy,col=hcl(h=140,l=seq(100,0,-10)))
> title("LiDAR mean canopy cover")
> image(TrAsp,col=hcl(h=140,l=seq(100,0,-10)))
> title("Transformed aspect")
> image(intensity,col=hcl(h=140,l=seq(100,0,-10)))
> title("LiDAR mean intensity")


> totBA=read.asciigrid("rf_totBA.asc")[100:450,400:700]
> abgr =read.asciigrid("rf_abgr.asc") [100:450,400:700]
> pipo =read.asciigrid("rf_pipo.asc") [100:450,400:700]
> psme =read.asciigrid("rf_psme.asc") [100:450,400:700]
> thpl =read.asciigrid("rf_thpl.asc") [100:450,400:700]

> par(mfcol=c(2,2),plt=c(.05,.95,.05,.85))
> image(totBA,col=hcl(h=140,l=seq(100,0,-10)))
> title("Total basal area")
> image(abgr,col=hcl(h=140,l=seq(100,0,-10)))
> title("Grand fir basal area")
> image(pipo,col=hcl(h=140,l=seq(100,0,-10)))
> title("Ponderosa pine basal area")
> image(psme,col=hcl(h=140,l=seq(100,0,-10)))
> title("Douglas fir basal area")
> image(thpl,col=hcl(h=140,l=seq(100,0,-10)))
> title("Western red cedar basal area")
```

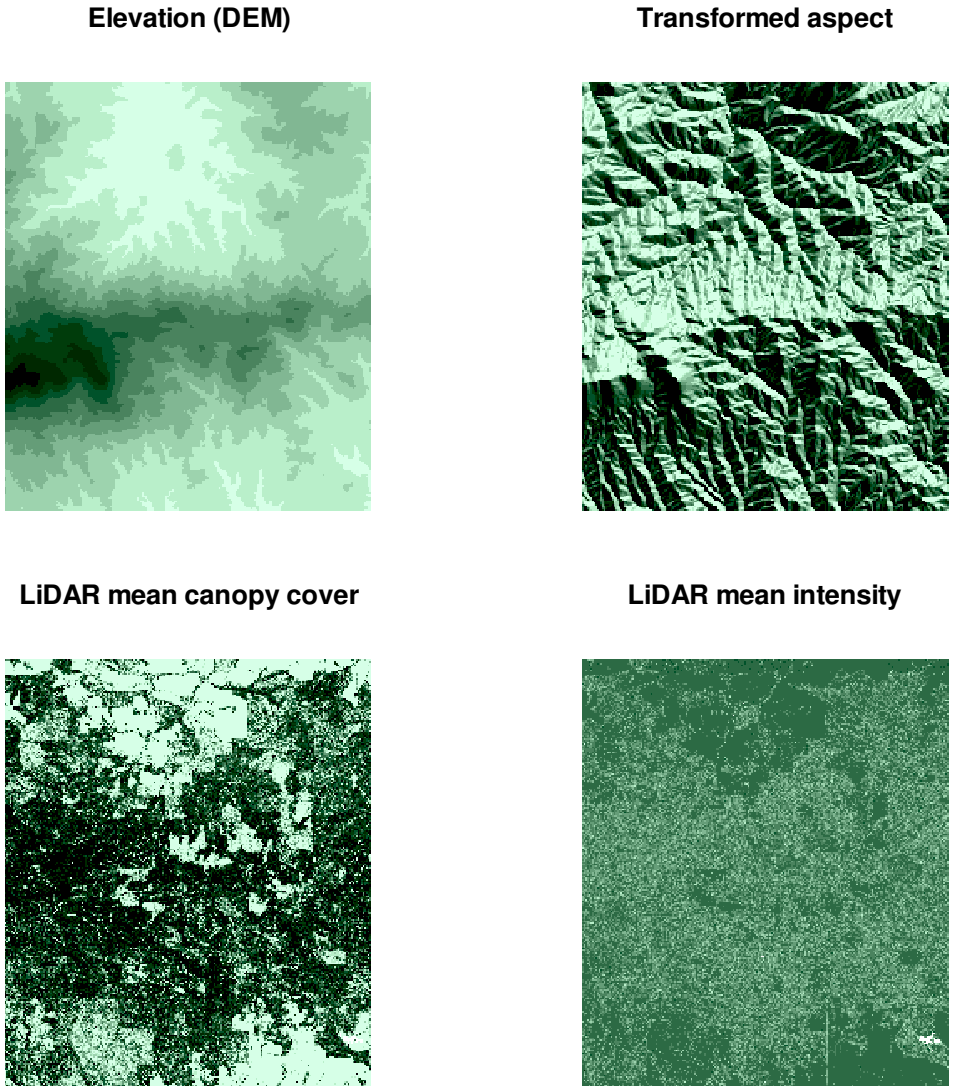

Figure 4: Grid maps of four of the predictor variables.

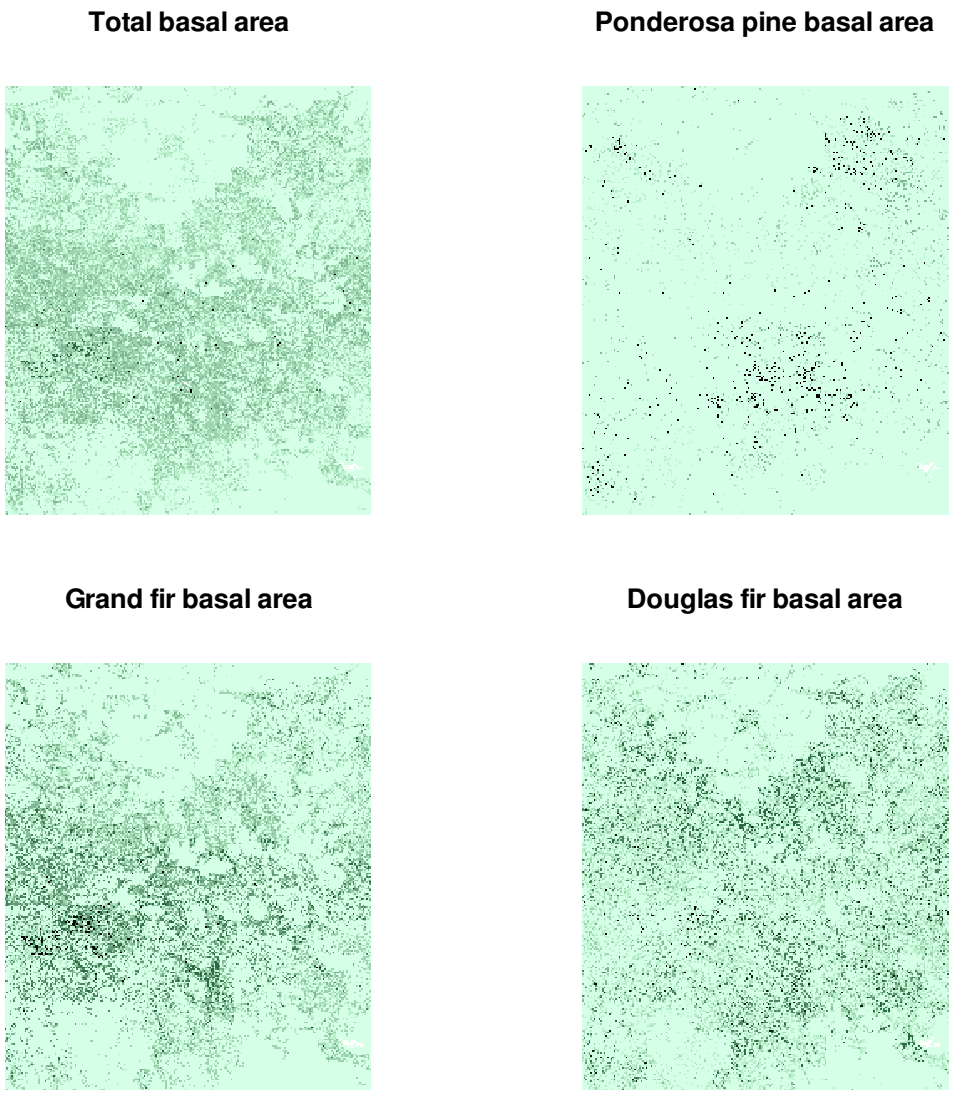

Figure 5: Maps of the basal area of four species. Note that any variable that is known for the reference observations can be imputed.

Conclusions

Package `yaImpute` was built to provide an integrated set of tools designed to meet specific challenges in forestry. It provides alternative methods for finding neighbors, integrates a fast search method, and introduces a novel and experimental application of `randomForest`. A function for computing the error statistics suggested by Stage and Crookston (In press) is included. We anticipate that progress in this field will continue, particularly in the area of discovering better X-variables and transformations improving the essential requirements for applying these methods: that there be a relationship between the X- and Y-variables.

References

Anderson, Edgar (1935). The irises of the Gaspe Peninsula, *Bulletin of the American Iris Society* 59:2–5.

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

Crookston, Nicholas L.; Moeur, Melinda; Renner, David. (2002). *Users guide to the Most Similar Neighbor Imputation Program Version 2*. Gen. Tech. Rep. RMRS-GTR-96. Ogden, UT: U.S. Department of Agriculture, Forest Service, Rocky Mountain Research Station. 35 p.

Evans, M.; Hastings, N.; Peacock, J.B. (2000). *Statistical Distributions.* New York: John Wiley & Sons, Inc. 219 p.

Holmström, H., Kallur, H., Støahl, G., (2003). Cost-Plus-Loss analyses of Forest Inventory Strategies based on kNN-Assigned Reference Sample Plot Data. *Silva Fennica* 37 (3), 381–398.

Korhonen, K.; Kangas; A. (1997). Application of nearest-neightbor regression for generating sample tree information. *Scandavian Journal of Forest Research* 12, 97–101.

Gittins, Robert. (1985). *Canonical Analysis A review with applications in ecology*. Volume 12 in Biomathematics. New York: Springer-Verlag. 351 p.

Hudak, A.T.; Crookston, N.L.; Evans, J.S.; Falkowski, M.J.; Smith, A.M.S.; Gessler, P.E.; Morgan, P. (2006). Regression modeling and mapping of coniferous forest basal area and tree density from discrete-return lidar and multispectral satellite data. *Can. J. Remote Sensing*. 32(2):126-138.

Liaw, A.; Wiener, M. (2002). Classification and Regression by `randomForest`. R News 2(3):18-22.

LeMay, V.; Temesgen, H. (2005). Comparison of Nearest Neighbor Methods for Estimating Basal Area and Stems per Hectare Using Aerial Auxiliary Variables. *For. Sci.* 51(2):109 –119.

Marchini, J.L.; Heaton, C.; Ripley, B.D. (2006). fastICA: FastICA algorithms to perform ICA and Projection Pursuit. R package version 1.1-8. http://www.stats.ox.ac.uk/~marchini/software.html

Moeur, Melinda; Stage, Albert R. (1995). Most similar neighbor: an improved sampling inference procedure for natural resource planning. *Forest Science*. 41:337-359.

Ohmann, J.L.; Gregory, M.J. (2002). Predictive mapping of forest composition and structure with direct gradient analysis and nearest neighbor imputation in coastal Oregon, U.S.A. *Can. J. For. Res.* 32:725–741

Oksanen, J.; Kindt, R.; O'Hara, R.B. (2005). vegan: Community Ecology Package version 1.6-10. http://cc.oulu.fi/~jarioksa/.

Pebesma, E.J.; Bivand, R.S. (2005). Classes and methods for spatial data in R. R News 5 (2), http://cran.r-project.org/doc/Rnews/.

R Development Core Team (2006). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

Rao, C.R. (1973). *Linear Statistical Inference*. New York: John Wiley & sons, Inc.

SAS. (2000). SAS Version Eight. Cary, NC: SAS Institute Inc.

Stage, A.R. (1976). An Expression for the effect of aspect, slope, and habitat type on tree growth. *For. Sci.* 22(4):457-460.

Stage, A.R.; Crookston, N.L. (In press). Partitioning error components for accuracy-assessment of near neighbor methods of imputation. *For. Sci.*

Warren, W.G. (1971). Correlation or Regression: Bias or Precision. *Applied Statistics*. 20(2):148-164.