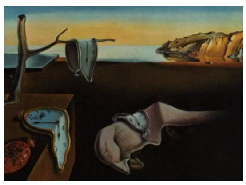

Journal of Statistical Software
January 2008, Volume 23, Issue 10. http://www.jstatsoft.org/

yaImpute: An R Package for *k*NN Imputation

Nicholas L. Crookston
USDA Forest Service

Andrew O. Finley
Michigan State University

Abstract

This article introduces **yaImpute**, an R package for nearest neighbor search and imputation. Although nearest neighbor imputation is used in a host of disciplines, the methods implemented in the **yaImpute** package are tailored to imputation-based forest attribute estimation and mapping. The impetus to writing the **yaImpute** is a growing interest in nearest neighbor imputation methods for spatially explicit forest inventory, and a need within this research community for software that facilitates comparison among different nearest neighbor search algorithms and subsequent imputation techniques. **yaImpute** provides directives for defining the search space, subsequent distance calculation, and imputation rules for a given number of nearest neighbors. Further, the package offers a suite of diagnostics for comparison among results generated from different imputation analyses and a set of functions for mapping imputation results.

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

1. Introduction

Natural resource managers and policymakers rely on spatially explicit forest attribute maps to guide forest management decisions. These maps depict the landscape as a collection of areal units. Each unit, or *stand*, delineates a portion of forest that exhibits similar attribute values (e.g., species composition, vegetation structure, age, timber volume, etc.). Collecting field-based forest measurements of these variables of interest is very expensive and this cost precludes exhaustive stand-level inventory across an entire landscape. Rather, to produce the desired maps, practitioners can use k-nearest neighbor (kNN) imputation which exploits the association between inexpensive auxiliary variables that are measured on all stands and the variables of interest measured on a subset of stands.

The basic data consist of a set of n *reference* points, $[p_i]_{i=1}^{n}$, strictly defined in real d-

dimensional space, $S \subset \mathbb{R}^d$, and a set of m *target* points $[q_j]_{j=1}^m \in \mathbb{R}^d$. A coordinate vector, X, of length d is associated with each point in the reference and target sets (i.e., auxiliary variables define points' coodinates). The Y vector of forest attributes of interest is associated with points in the reference set and holds values to be imputed to each target point q. The basic task is to find the subset of k reference points within S that have minimum distance to a given q, where distance is a function of the respective coordinate vectors X. Then, given this set of k nearest reference points, impute the vector of Y to q. When $k = 1$ the Y associated with the nearest p is assigned to q, but if $k > 1$ other rules are used. Note that imputation described here is different than the common imputation approach used in missing data problems, that is, when some p, q, or both, have missing X values. Here, missing values of X are not allowed in the target or reference set and missing values of Y are not allowed in the reference set.

Qualities of the kNN method make it an attractive tool for forest attribute estimation and mapping (Holmström and Fransson 2003). These qualities include multivariate imputation, preservation of much of the covariance structure among the variables that define the Y and X vectors (e.g., Moeur and Stage 1995; Tomppo 1990), and relaxed assumptions regarding normality and homoscedasticity that are typically required by parametric estimators. There are several popular variants of the kNN method. The basic method, within the forest attribute imputation context, is described by Tomppo (1990), McRoberts, Nelson, and Wendt (2002), and Franco-Lopez, Ek, and Bauer (2001). Others include most similar neighbor (MSN, Moeur and Stage 1995) and gradient nearest neighbor (GNN, Ohmann and Gregory 2002). All of these methods define *nearness* in terms of weighted Euclidean distance. The principal distinction is how the d-dimensional search space is constructed (i.e., how the weight matrix is defined). In addition to implementing these and other popular weighted Euclidean distance based methods, the **yaImpute** package offers a novel nearest neighbor distance metric based on the random forest proximity matrix (Breiman 2001). The package written in R (R Development Core Team 2007) and is available from the Comprehensive R Archive Network at `http://CRAN.R-project.org/`.

The remainder of the article is as follows. Section 2 details the various approaches used for defining nearness within the search space, offers several diagnostics useful for comparing imputation results, and describes the efficient nearest neighbor search algorithm use within **yaImpute**. Section 3 outlines functions available within **yaImpute**. Section 4 provides two example analyses. Finally, Section 5 provides a brief summary and future direction.

2. Methods

2.1. Measures of nearness

yaImpute offers several methods for finding nearest neighbors. For all methods, except those based on a random forest proximity matrix discussed below, nearness is defined using weighted Euclidean distance, $dist(p, q, \mathbf{W}) = [(p - q)^T \mathbf{W}(p - q)]^{1/2}$, where $\mathbf{W}$ is the weight matrix. Table 1 defines the weight matrices for the given methods. The key words in the first column of Table 1 are used as function directives within the **yaImpute** package. The method `raw` bases distance on the untransformed, or raw, values of the X variables, whereas, `euclidean` uses normalized X variables to define distance. The method `mahalanobis` transforms the

Method	Value of $\mathbf{W}$
`raw`	Identity matrix, $\mathbf{I}$
`euclidean`	Inverse of the direct sum of the X's covariance matrix.
`mahalanobis`	Inverse of the X's covariance matrix.
`ica`	$\mathbf{K\Omega}^T\mathbf{K\Omega}$, where $\mathbf{\Omega}$ and $\mathbf{K}$ correspond to $\mathbf{W}$ and $\mathbf{K}$ definitions given in the **fastICA** R package.
`msn`	$\mathbf{\Gamma\Lambda}^2\mathbf{\Gamma}^T$, where $\mathbf{\Gamma}$ is the matrix of canonical vectors corresponding to the X's found by canonical correlation analysis between X and Y, and $\mathbf{\Lambda}$ is the canonical correlation matrix.
`msn2`	$\mathbf{\Gamma\Lambda}(\mathbf{I} - \mathbf{\Lambda}^2) - 1\mathbf{\Lambda\Gamma}^T$.
`gnn`	$\mathbf{\Theta}$, weights assigned to environmental data in canonical corresponence analysis.
`randomForest`	No weight matrix.

Table 1: Summary of how $\mathbf{W}$ is computed.

search space by the inverse of the covariance matrix of the X variables, prior to computing distances. Method `ica` computes distance in a space defined using Independent Component Analysis calculated by the `fastICA` function in the R package **fastICA** (see Marchini, Heaton, and Ripley 2007, for details). For methods `msn` and `msn2`, (see e.g., Moeur and Stage 1995; Crookston, Moeur, and Renner 2002), distance is computed in projected canonical space, and with method `gnn` distance is computed using a projected ordination of Xs found using Canonical Correspondence Analysis. For the last method listed in Table 1, `randomForest`, 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 (see e.g., Breiman 2001; Liaw and Wiener 2002, for details). 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, kNNs are the k minimum of these distances. There are two notable advantages of method `randomForest`, it is non-parametric, and the variables can be a mixture of continuous and categorical variables. The other methods require continuous variables to define the search space axes. A third advantage is that the data can be rank-deficient, having many more variables than observations, colinearities, or both.

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 find those canonical vectors that are significantly greater than zero and use them all. Rao (1973) defines an approximate F-statistic for testing the hypothesis that the current and all smaller canonical correlations are zero in the population. Gittins (1985) notes that Λ^2 varies in the range from zero to one and if the F-statistic 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 F-statistic 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. The test requires a user specified p-value, which is commonly set as 0.05.

For method `randomForest` a distance measure based on the proximity matrix is used (Breiman 2001). 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) shows that the proximities are a kind of Euclidean distance between observations and uses them as a basis for imputing missing values to observations.

Note that the random forest proximity matrix is often too large to store in main memory. Therefore, the **yaImpute** code stores a much smaller matrix called the nodes matrix. This matrix is $n \times n_{tree}$ where n_{tree} is the number of trees, rather than $n \times n$ for the proximity matrix. The elements of this matrix are terminal node identifications. In **yaImpute**, the node matrix is partitioned into two matrices, one each for the reference and target observations. When finding the k nearest neighbors, only the necessary proximities are computed and used when needed thereby avoiding the allocation of the potentially prohibitorly large matrix.

The random forests algorithm implemented in the R package **randomForest** 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 (Liaw and Wiener 2002).

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, we devised an experimental way to handle multivariate Ys. The method is to build a separate forest for each Y and then join the nodes matrices of each forest. The forests are each conditioned to reduce the 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.

2.2. Diagnostics

There is a great 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. **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 the function `errorStats` to compute the statistics proposed by Stage and Crookston (2007).

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`). Following Warren (1971), we recommend against using correlation as a measure of the degree of association between observed and imputed values because correlation is dependent on the sample distribution along the scale of X-variables while root mean square difference (rmsd) and standard error of imputation (SSI, see Stage and Crookston (2007) 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, $y = f(\mathbf{X})$. If we used this formula in a nearest neighbor-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!

Because imputated values include different components of error than those estimated using regression, Stage and Crookston (2007) exchanged the word *error* with *difference* in defining root mean square difference (*rmsd*). In the case described above, *rmse* will approach zero as the regression approaches the true function, but *rmsd* can never be zero.

Another useful diagnostic is to identify 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`, we assume that the distribution of distances among randomly sampled references follows a lognormal distribution. The justification for using the lognormal is that distances (d) are constrained by zero ($0 \leq d < \infty$) and have few large values in practice. Following that assumption, a threshold is defined as the value of d 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 as $0 \leq d \leq 1$. We used the formulas provided by Evans, Hastings, and Peacock (1997) for estimating the parameters of these distributions. Alternatively, users may specify the threshold, perhaps by inspecting the frequency distribution and choosing the threshold visually. This alternative enables user's to use the `notablyDistant` function without using our distributional assumptions.

Our experience using this diagnostic tool with earlier implementations of method `msn` (Crookston *et al.* 2002) proved quite valuable in pointing out to users that forest types existed in their target data with no representative samples in the reference data. This situation prompted additional field sampling to correct the situation. A paradox emerged when the additional references were added to the analysis. That is, the threshold values of d would decrease when they were computed using the lognormal distribution assumption outlined above, resulting in *more*, rather than fewer notably distant neighbors being identified. When the threshold was held constant between analyses made with and without the augmented reference data sets, an improvement due to adding reference data was evident.

2.3. Efficient nearest neighbor search

For any given target observation (point), the n distance calculations and subsequent distance vector sort are computationally demanding steps in the nearest neighbor *brute force* solution. Specifically, the complexity of the brute force solution increases linearly as a function of the product of d and n (i.e., with complexity on the order of dn, $O(dn)$, assuming the distance between points can be calculated in $O(d)$ time). This computational burden makes kNN model parameterization and imputation over a large set of targets very time-consuming and therefore represents a substantial disadvantage of the kNN technique.

Several methods have been proposed to improve nearest neighbor search efficiency (see e.g., literature on partial codebook search algorithms Cheng and Lo 1996; Ra and Kim 1993, and references therein). These methods, however, provide only modest improvements in search efficiency, especially within the context of kNN mapping of forest attributes (Finley, McRoberts, and Ek 2006). Finley and McRoberts (2008) found that kd-tree data structures and associated exact and approximate search algorithms (see e.g., Friedman, Bentley, and Finkel 1977; Sproull 1991; Arya, Mount, Netanyahu, Silverman, and Wu 1998) provide substantial gain in search efficiency over conventional search algorithms. For kd-tree construction and subsequent efficient nearest neighbor searches, the high-level R functions within the **yaImpute** call low-level C++ routines in the approximate nearest neighbor (**ANN**) library written by Mount (1998).

3. Package contents

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

`yai` finds kNNs given reference and, optionally, target observations. This function is the main function in the package it returns an object of class `yai`, described below. A key role played in this function is to separate the observations into reference and target observations. Target observations are those with values for X variables and not for Y variables, while reference observations are those with no missing values for X and Y variables.

`ann` provides access to approximate nearest neighbor search routines and is called by `yai` and `newtargets` (see below).

`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 using function `yai`.

`foruse` takes an imputation (object class `yai`) and returns a data frame with 2 columns. Row names are target observation identifications, the first column is the row name of the reference observations used to represent it, and the second column is the distance between the reference and target observations.

`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 grid point in the input grid maps and outputs maps of selected Y-variables (or other data) 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.

`errorStats` computes error statistics as proposed by Stage and Crookston (2007).

`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 threshold is used to detect large distances is either specified or the function computes a suitable 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**:

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

`yaiVarImp` outputs (optionally plots) 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. `NA`s 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 Stage and Crookston (2007).

`MoscowMtStJoe` is data from the Moscow Mountain area and the St. Joe Woodlands, northeast of Moscow, Idaho, USA (Hudak, Crookston, Evans, Falkowski, Smith, Gessler, and Morgan 2006).

Classes:

`yai` is a list returned by function `yai` that contains elements as listed in the manual entry for the package. 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) and `neiIdsTrgs` is a corresponding data frame of target 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 (ancillary data) supplied in 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.

4. Examples

4.1. Iris data

The famous iris data described by 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.

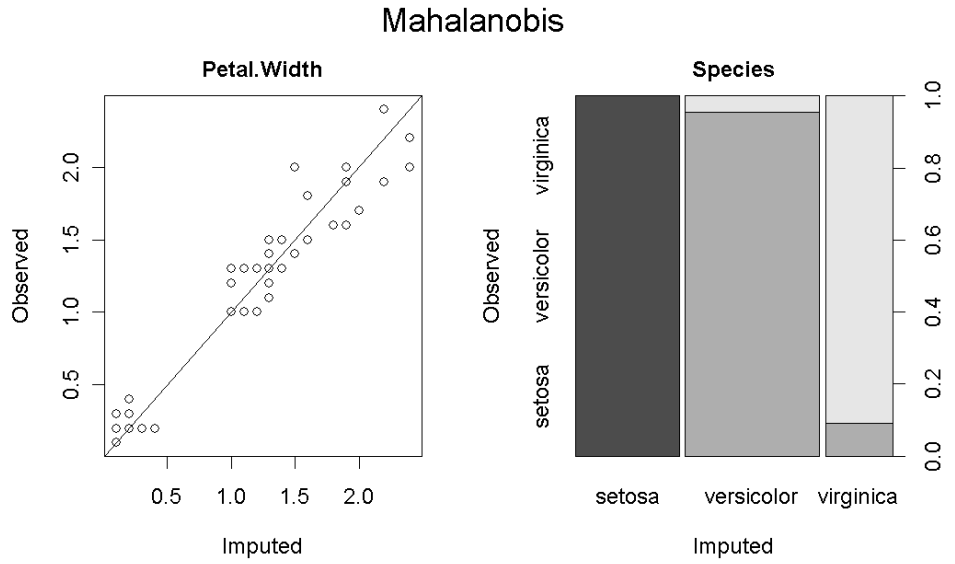

Figure 1: Output from running the plot command on the object generated with the simple code.

```
R> library("yaImpute")
R> data("iris")
R> set.seed(1)
R> refs = sample(rownames(iris), 50)
R> x <- iris[, 1:3]
R> 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`). 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 (Figure 1).

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

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 target observations (details on options controlling the behavior of the `impute` function are provided in the manual pages).

```
R> 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

R> 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>
```

4.2. Forest inventory data

To further 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 model (method) is selected. 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.

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

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.

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

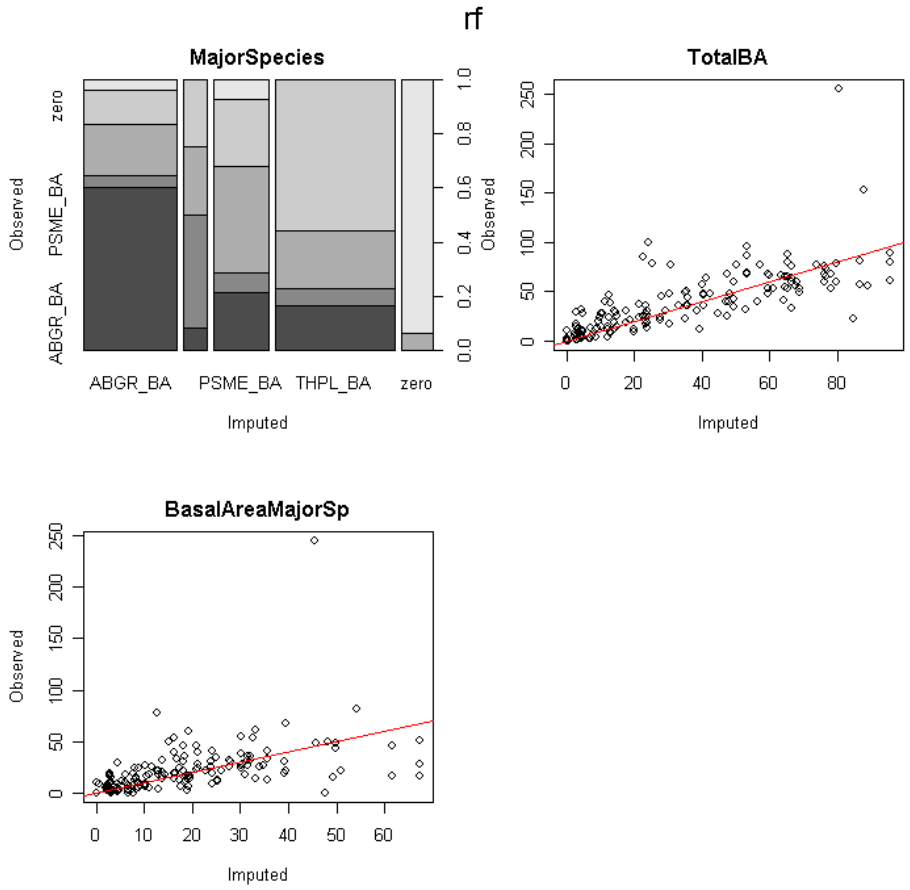

Figure 2: Plot of the `yai` object generated with `method=randomForest`.

```
  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
```

```
R> 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 (Figure 2).

```
R> plot(rf, vars = yvars(rf))
```

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 (Figure 3) for all the methods:

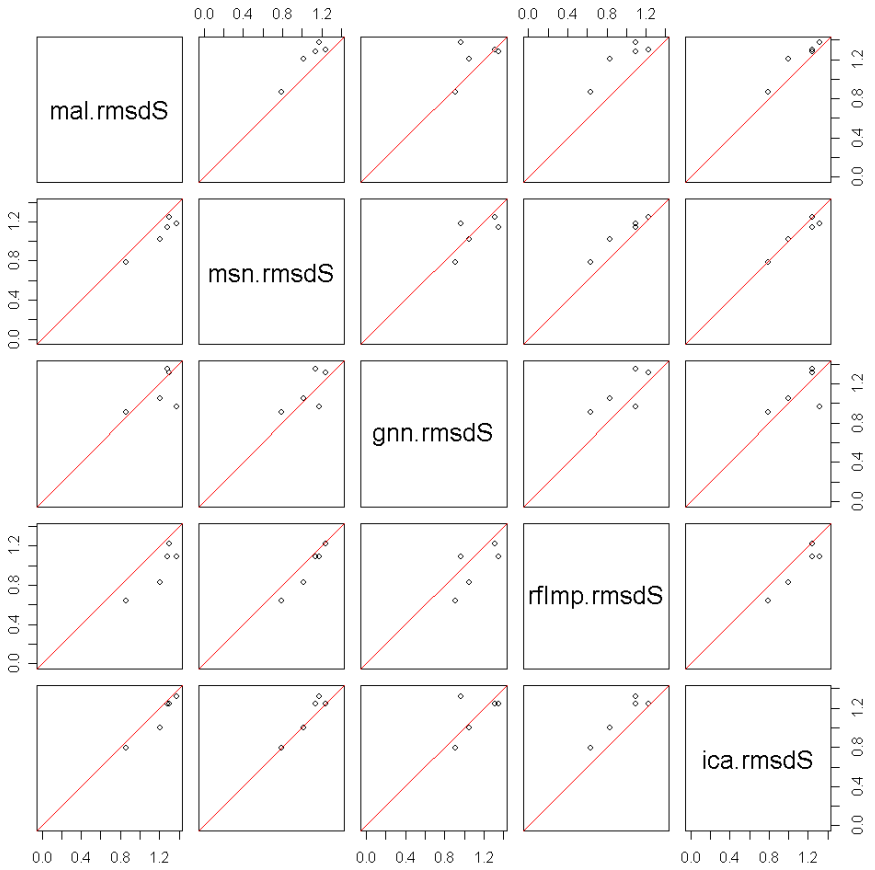

Figure 3: Comparison of the scaled `rmsd` for each method. Most of the values for method `rfImp` (imputed Y-variables build using method `randomForest`) are below the 1:1 line indicating that they are generally lower than those for other methods.

```
R> rfImp <- impute(rf, ancillaryData = y)
R> rmsd <- compare.yai(mal, msn, gnn, rfImp, ica)
R> apply(rmsd, 2, mean, na.rm = TRUE)

  mal.rmsdS   msn.rmsdS   gnn.rmsdS rfImp.rmsdS   ica.rmsdS
   1.205818    1.073877    1.118564    1.051907    1.119871

R> plot(rmsd)
```

The steps presented so far can be repeated using the data available in the package distribution. However, the following steps require that ASCII grid maps of the X-variables be available and they are not part of the distribution. The data are available as supplemental data to the Web page for this article (http://www.jstatsoft.org/v23/i10/).

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 trans-

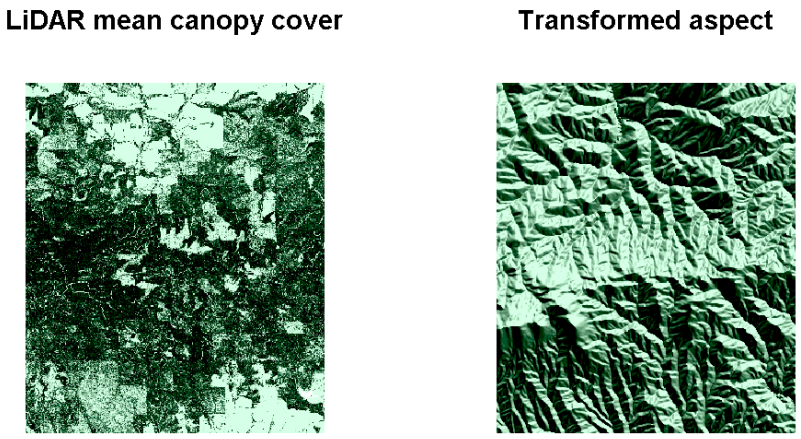

Figure 4: Grid maps of two of the predictor variables.

formed from that in the original set of Y-variables. Therefore, the original `y` data frame is passed to function `AsciiGridImpute` as ancillary data.

```
R> 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")
R> 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")
R> AsciiGridImpute(rf, xfiles, outfiles, ancillaryData = y)
```

The R package **sp** by 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 example X- and Y-variables (Figures 4 and 5).

```
R> library("sp")
R> canopy <- read.asciigrid("canopy.asc")[100:450, 400:700]
R> TrAsp <- read.asciigrid("trasp.asc")[100:450, 400:700]
R> par(mfcol = c(1, 2), plt = c(0.05, 0.95, 0.05, 0.85))
R> image(canopy, col = hcl(h = 140, l = seq(100, 0, -10)))
R> title("LiDAR mean canopy cover")
R> image(TrAsp, col = hcl(h = 140, l = seq(100, 0, -10)))
R> title("Transformed aspect")
```

```
R> totBA <- read.asciigrid("rf_totBA.asc")[100:450, 400:700]
R> psme <- read.asciigrid("rf_psme.asc")[100:450, 400:700]
R> par(mfcol = c(1, 2), plt = c(0.05, 0.95, 0.05, 0.85))
R> image(totBA, col = hcl(h = 140, l = seq(100, 0, -10)))
```

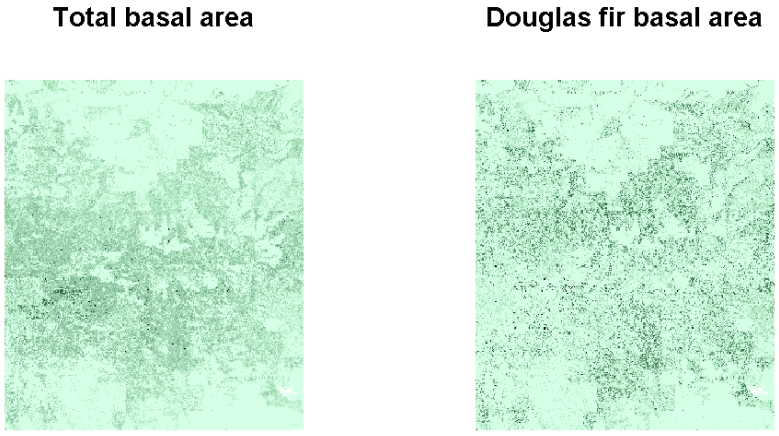

Figure 5: Maps of the total basal area and one of four species. Note that any variable that is known for the reference observations can be imputed.

```
R> title("Total basal area")
R> image(psme, col = hcl(h = 140, l = seq(100, 0, -10)))
R> title("Douglas fir basal area")
```

5. Summary

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 (2007) is also 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.

Acknowledgments

The authors wish to thank Dr. Albert R. Stage for his years of council and encouragement on this and related projects.

References

Anderson E (1935). "The Irises of the Gaspe Peninsula." *Bulletin of the American Iris Society*, **59**, 25.

Arya S, Mount DM, Netanyahu NS, Silverman R, Wu A (1998). "An Optimal Algorithm for Approximate Nearest Neighbor Searching." *Journal of the ACM*, **45**, 891–923.

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

Cheng SM, Lo KT (1996). "Fast Clustering Process for Vector Quantisation Codebook Design." *Electronic Letters*, **32**(4), 311–312.

Crookston NL, Moeur M, Renner D (2002). *Users Guide to the Most Similar Neighbor Imputation Program Version 2*. Gen. Tech. Rep. RMRS-GTR-96., US Department of Agriculture, Forest Service, Rocky Mountain Research Station, Ogden, Utah. URL http://www.fs.fed.us/rm/pubs/rmrs_gtr096.html.

Evans M, Hastings N, Peacock JB (1997). *Statistical Distributions*. John Wiley and Sons, Inc., New York.

Finley AO, McRoberts RE (2008). "Efficient k-Nearest Neighbor Searches for Multi-Source Forest Attribute Mapping." *Remote Sensing of Environment*. In press.

Finley AO, McRoberts RE, Ek AR (2006). "Applying An Efficient k-Nearest Neighbor Search to Forest Attribute Imputation." *Remote Sensing of Environment*, **52**(2), 130–135.

Franco-Lopez H, Ek AR, Bauer ME (2001). "Estimation and Mapping of Forest Stand Density, Volume, and Cover Type Using the k-Nearest Neighbor Method." *Remote Sensing of Environment*, **77**, 251–274.

Friedman JH, Bentley JL, Finkel RA (1977). "An Algorithm for Finding Best Matches in Logarithmic Expected Time." *ACM Transactions on Mathematical Software*, **3**(3), 209–226.

Gittins R (1985). *Canonical Analysis: A Review with Applications in Ecology*. Springer-Verlag, New York.

Holmström H, Fransson JES (2003). "Combining Remotely Sensed Optical and Radar Data in kNN-Estimation of Forest Variables." *Forest Science*, **49**(3), 409–418.

Hudak AT, Crookston NL, Evans JS, Falkowski MJ, Smith AMS, Gessler PE, Morgan P (2006). "Regression Modeling and Mapping of Coniferous Forest Basal Area and Tree Density from Discrete-Return Lidar and Multispectral Satellite Data." *Canadian Journal of Remote Sensing*, **32**(2), 126–138.

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

Marchini JL, Heaton C, Ripley BD (2007). "**fastICA**: FastICA Algorithms to Perform ICA and Projection Pursuit." R package version 1.1-9, URL http://CRAN.R-project.org/.

McRoberts RE, Nelson MD, Wendt DG (2002). "Stratified Estimation of Forest Area using Satellite Imagery, Inventory Data, and the K-Nearest Neighbor Technique." *Remote Sensing of Environment*, **82**, 457–468.

Moeur M, Stage AR (1995). "Most Similar Neighbor: An Improved Sampling Inference Procedure for Natural Resources Planning." *Forest Science*, **41**(2), 337–359.

Mount DM (1998). "**ANN** Programming Manual." *Technical report*, Department of Computer Science, University of Maryland. URL http://citeseer.ist.psu.edu/333325.html.

Ohmann JL, Gregory MJ (2002). "Predictive Mapping of Forest Composition and Structure with Direct Gradient Analysis and Nearest Neighbor Imputation in Coastal Oregon, USA." *Canadian Journal of Forest Research*, **32**, 725–741.

Pebesma EJ, Bivand RS (2005). "Classes and Methods for Spatial Data in R." *R News*, **5**(2), 9–13. URL http://CRAN.R-project.org/doc/Rnews/.

Ra SW, Kim JK (1993). "A Fast Mean-Distance-Ordered Partial Codebook Search Algorithm for Image Vector Quantization." *IEEE Tranactions on Circuits and Systems*, **40**(9), 576–579.

Rao CR (1973). *Linear Statistical Inference*. John Wiley and Sons, Inc., New York.

R Development Core Team (2007). *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/.

Sproull RF (1991). "Refinements to Nearest-Neighbor Searching in K-Dimensional Trees." *Algorithmica*, **6**, 579–589.

Stage AR, Crookston NL (2007). "Partitioning Error Components for Accuracy-Assessment of Near-Neighbor Methods of Imputation." *Forest Science*, **53**, 62–72.

Tomppo E (1990). "Designing a Satellite Image-Aided National Forest Survey in Finland." In "SNS/IUFRO Workshop on the Usability of Remote Sensing for Forest Inventory and Planning," pp. 43–47. Umea, Sweden.

Warren WG (1971). "Correlation or Regression: Bias or Precision." *Applied Statistics*, **20**(2), 148–164.

Affiliation:

Nicholas L. Crookston
USDA Forest Service
Rocky Mountain Research Station
1221 South Main Street
Moscow, Idaho 83843, United States of America
E-mail: ncrookston@fs.fed.us

Andrew O. Finley
Department of Forestry and Department of Geography
Michigan State University
East Lansing, Michigan 48824-1222, United States of America
E-mail: finleya@msu.edu

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 23, Issue 10 *Submitted:* 2007-06-20
January 2008 *Accepted:* 2007-10-10