Imputation of Missing Values for Unsupervised Data Using the Proximity in Random Forests

Tsunenori Ishioka
Research Division
The National Center for University Entrance Examinations
Tokyo, Japan
Email: tunenori@rd.dnc.ac.jp

***Abstract*—This paper presents a new procedure that imputes missing values by random forests for unsupervised data. We found that it works pretty well compared with k-nearest neighbor (kNN) and rough imputations replacing the median of the variables. Moreover, this procedure can be expanded to semi-supervised data sets. The rate of the correct classification is higher than that of other conventional methods. The imputation by random forests for unsupervised or semi-supervised cases was not implemented.**

Keywords-Ensemble learning; k-nearest neighbor; R; rfImpute; impute.knn.

I. INTRODUCTION

A method of random forests [1] is a substantial modification of bagging techniques that builds a large collection of de-correlated trees and then averages them. Therefore, it is mainly used as an accurate classifier or regression tree. The latest Fortran77 code programmed by Breiman [2] is Version 5.1, dated June 15, 2004. Since Version 4 contains modifications and major additions to Version 3.3, replacement of missing predictor values has been enabled [3]. Breiman offers two options. One is the "missquick" (Ver. 4), which replaces all missing values by the median of the non-missing values in their column, if real, and by the most numerous value in their column if categorical. Another is "missright" (Ver. 5). This option starts with "missquick" but then iterates by using proximities and does an effective replacement even with a large amount of missing data. Missing values are presented by a proximity weighted sum over the non-missing values.

On the basis of these ideas, Andy Liaw implemented their varieties in statistical environment R [4], calling them "na.roughfix" and "rfImpute" [5]. The advantage is that these R functions work for both regression and classification, but unfortunately cannot be applied for unsupervised (unlabeled) cases as a training data set [6]. Only predictive variables in supervised learning are allowed missing.

However, Breiman's ideas could be extended to unsupervised data if we could obtain the proximity of the unsupervised data. The new proximities can be obtained by starting the rough imputation for missing data ("na.roughfix") and repeating to run random forests. The artificial occurences of response variable are given by the method described later (Section 2). In the case of supervised data, Breiman [3] found that an estimate error of a bootstrap train sample (called "out-of-bag", or oob) tends to be optimistic when run on a data matrix with imputed values.

Related works are follows: Pantanowitz and Marwala [7] evaluated the impact of missing data imputation by using human immunodeficiency virus (HIV) seroprevalence data. Rieger et al. [8] provided an implementation of random forests with missing vales in the covariates. Nicholas [9] extended the random forest to handle multi-response variables, and presented another imputation method called "yaImpute." But all the methods described above are not allowed for unsupervised or semi-supervised data.

In this paper, we present a new procedure for proper missing values imputation, which can avoid the overfitting of the estimated model for unsupervised data. In Section 2, we summarize the elements of a technique that imputes the missing values for unsupervised data. In Section 3, we show a new procedure for imputing the missing values. In Section 4, two examples, iris and spam data sets, are illustrated. We assume these data to be unsupervised by dropping the response variables; nevertheless, both are supervised. Section 5 shows the expansion of our method to semi-supervised data sets. Section 6 is the summary.

II. RFIMPUTE

A. Proximity measure

Breiman [3] defines the data proximity as follows: The (i, j) element of the proximity matrix produced by a random forest is the fraction of trees in which elements i and j fall in the same terminal node. The intuition is that "similar" observations should be in the same terminal nodes more often than dissimilar ones. The proximity matrix can be used to identify structures in the data, and for unsupervised learning with random forests.

B. An unsupervised learning example [10]

Because random forests are collections of classification or regression trees, it is not immediately apparent how they can be used for unsupervised learning. The "trick" is to call

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-253-0

the data "class 1" and construct "class 2" synthetic data, and then try to classify the combined data with a random forest. There are two ways to simulate the "class 2" data:

1) The "class 2" data are sampled from the product of the marginal distributions of the variables (by an independent bootstrap of each variable separately).
2) The "class 2" data are sampled uniformly from a hypercube containing the data (by sampling uniformly within the range of the variables).

The idea is that real data points that are similar to one another will frequently end up in the same terminal node of a tree — exactly what is measured by the proximity matrix. Thus, the proximity matrix can be taken as a similarity measure, and clustering or multidimensional scaling that uses this similarity can be used to divide the original data points into groups for visual exploration.

C. R procedure

Missing values are indicated by NAs in R [4]. A function returning a result of random forests is "randomForest" developed by Liaw [5]. The algorithm starts by imputing NAs by using "na.roughfix." Then, "randomForest" is called with the completed data. The proximity matrix from the "randomForest" is used to update the imputation of the NAs. For continuous predictors, the imputed value is the weighted average of the non-missing observations, where the weights are the proximities. For categorical predictors, the imputed value is the category with the largest average proximity. This process is iterated a few times.

A function returning the imputed values by random forests is "rfImpute," coded by Liaw [6]. We should note that Liaw's imputation is only available to supervised data without any missing response values.

III. New Procedure to Impute the Missing Data

A. Missing value replacement on the training set

Our procedure as well as Liaw's "rfImpute," has two ways of replacing missing values. The first way is fast. If the mth variable is not categorical, the method computes the median of all values of this variable in class j, then it uses this value to replace all missing values of the mth variable in class j. If the mth variable is categorical, the replacement is the most frequent non-missing value in class j. These missing values are replaced or filled by "na.roughfix."

The second way for replacing missing values is computationally more expensive but performs better than the first, even with large amounts of missing data. It begins by doing a rough and inaccurate filling in of the missing values. Our key technique is to estimate the missing values on the basis of not all non-missing proximities but k-nearest proximities, which include missing data. Then, it runs a forest procedure and computes proximities.

If $x(n, m)$ is a missing continuous value, we estimate its fill as an average over the k-nearest neighbor values of the mth variables weighted by the proximities between the nth case and the other case. If it is a missing categorical variable, we replace it by the most frequent non-missing value where frequency is weighted by proximity.

In summary, we use, in case of a missing continuous value,

$$\widehat{x}(n,m) = \frac{\sum\limits_{\substack{i \neq n \\ i \in \text{neighbor}}} \text{prox}(i,n) x(i,m)}{\sum\limits_{\substack{i \neq n \\ i \in \text{neighbor}}} \text{prox}(i,n)}, \tag{1}$$

instead of rfImpute's

$$\widehat{x}(n,m) = \frac{\sum\limits_{\substack{i \neq n \\ i \in \text{non-missing}}} \text{prox}(i,n) x(i,m)}{\sum\limits_{\substack{i \neq n \\ i \in \text{non-missing}}} \text{prox}(i,n)},$$

where $\text{prox}(\cdot, \cdot)$ is the proximity.

In case of a missing categorical variable, we use

$$\widehat{x}(n,m) = \underset{C_m}{\text{argmax}} \sum_{i \neq n} \text{prox}(i,n), \tag{2}$$

instead of

$$\widehat{x}(n,m) = \underset{C_m}{\text{argmax}} \sum_{\substack{i \neq n \\ i \in \text{non-missing}}} \text{prox}(i,n),$$

where C_m means the mth categorical variables.

Now, iterate-construct a forest again by using these newly filled in values, find new fills, and iterate again. Our experience is that 4–6 iterations are enough.

The reason we use only k-nearest neighbor data in (1) is that the missing imputation of this method would be robust. Even if proximities to the target are rather small, the other continuous values may be outlying. In this case, some outliers will affect the estimate of the target toward ill direction. Our numerical investigation shows that our procedure, the mixture of kNN and random forests, is better than using only random forests. This technique leads the estimates to avoid overfitting of the random forest model.

In (2), however, all data besides k-nearest neighbor data are treated. Because majority votes were adopted, outlying values of x would be unregarded. While, we should regard the proximity associated with missing data, especially when the missing rate is high.

B. Missing value replacement on the test set

When there is a test set, there are two different methods for replacement depending on whether labels exist for the test set.

If they do, then the fills derived from the training set are used as replacements. If labels do not exist, then each case in the test set is replicated "number of classes" times. The

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-253-0

first replicate of a case is assumed to be class 1 and the class 1 fills used to replace missing values. The second replicate is assumed class 2 and the class 2 fills used on it.

This augmented test set is run down the tree. In each set of replicates, the one receiving the most votes determines the class of the original case.

C. Algorithm

The procedures are summarized as follows.

1) Impute NAs by using "na.roughfix."
2) Repeat following steps for "iter" times. Compute the proximities between all cases by using "randomForest." Then, impute the missing values. If the imputed values are converged, break the loop.
3) Output the data that include estimated (imputed) data.

The procedure will stop when either of the following conditions is satisfied.

1) The number of iterations reaches pre-determined reputation times; the default is 5.
2) The relative differences between the imputed missing values are sufficiently small, less than 1.0e-5.

The R program used in this paper should be referred to Appendix (Fig. 6). Fairly detailed comments are included in the program. The format is in accordance with the tradition of unix or R codings.

IV. Numerical Examples

A. E-mail database indicating spam or non-spam

We use a spam data set [11] collected at Hewlett-Packard Labs, which classifies 4601 e-mails as spam or non-spam. In addition to this class label, there are 57 variables indicating the frequency of certain words and characters in the e-mail. That is, a data frame with 4601 observations and 58 variables. The first 48 variables contain the frequency of the variable name (e.g., business) in the e-mail. If the variable name starts with num (e.g., num650), it indicates the frequency of the corresponding number (e.g., 650). Variables 49–54 indicate the frequency of the characters ";", "(", "[", "!","$", and "#". Variables 55–57 contain the average, longest, and total run-length of capital letters. Variable 58 indicates the type of the mail and is either "nonspam" or "spam," i.e. unsolicited commercial e-mail.

The data set contains 2788 e-mails classified as "non-spam" and 1813 classified as "spam." The "spam" concept is diverse: advertisements for products/web sites, make money fast schemes, chain letters, pornography, and so on. This collection of spam e-mails came from the collectors' postmaster and individuals who had filed spam. The collection of non-spam e-mails came from filed work and personal e-mails, and hence, the word "george" and the area code "650" are indicators of non-spam. We would have to blind spam/non-spam indicator, because we are focusing unsupervised data in this numerical experiment.

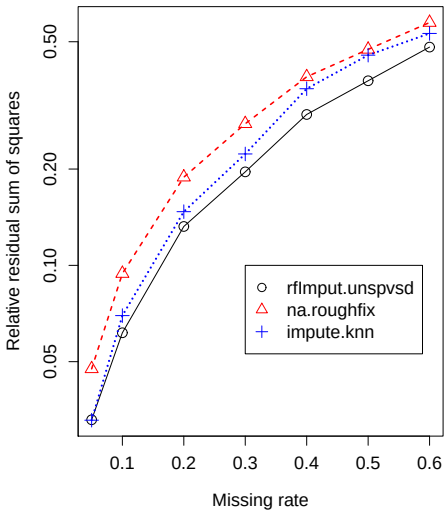

Figure 1. Relative residual sum of squares for unsupervised spam/non-spam data

To illustrate the performance of our method, we compare it with two conventional methods: "na.roughfix" and "impute.knn." The former is used as the baseline of our method. The latter is a typical kNN method [12] stored at biocLite library in R. We set k as the number of neighbors to be 10, the default value of this library. We name our method "rfImput.unspvsd", which means "an imputation method by using random forests for an unsupervised data set."

Missing data for 57 variables are randomly dropped. The missing data rates are 5%, 10%, 20%, 30%, 40%, 50%, and 60%. Fig. 1 shows the relative residual sum of square errors (RSS) between dropped true values and the estimates, depending on missing data rates. Three methods, "na.roughfix", "impute.knn" and "rfImput.unspvsd," are compared with each other. Less RSS shows better performance of their imputations. We found that our method is not inferior to the other two methods irrespective of the missing data rate. Roughly speaking, our method improves the performances 20–30% compared with "na.roughfix" and 5–10% compared with "impute.knn."

B. Edgar Anderson's iris data

The next example is the famous Fisher's or Anderson's iris data set, which gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of three species of iris. The species are "Iris setosa," "versicolor," and "virginica" [13]. In R, "iris" is a data frame with 150 observations and 5 variables.

Since this data set was treated as an example of discriminant analysis by Fisher, it became a typical test case for

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-253-0

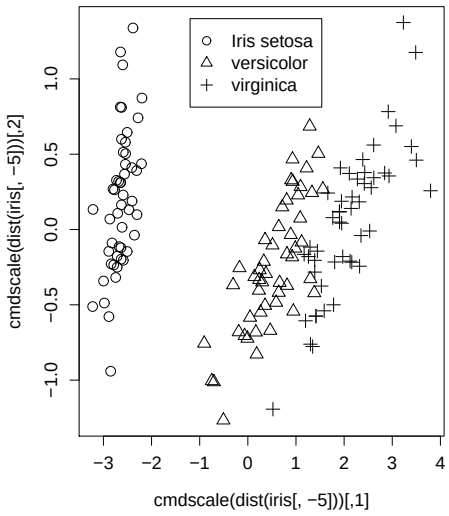

Figure 2. Actual species visualized by MDS

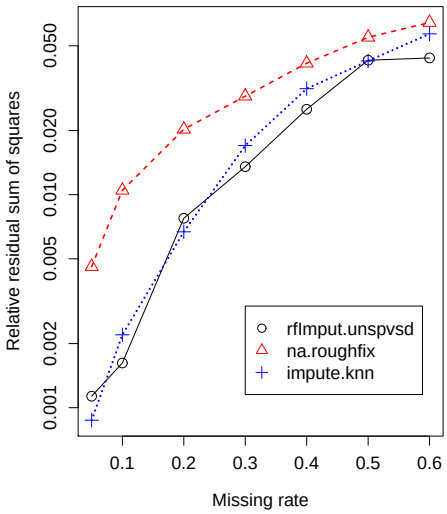

Figure 3. Relative residual sum of squares for unsupervised iris data

many classification techniques in machine learning. Note that the data set only contains two clusters with rather obvious separation. Fig. 2 shows the actual iris species by using multidimensional scaling (MDS), which is used in information visualization for exploring similarities. We assign a location to each observation in 2-dimensional MDS space.

One of the clusters contains Iris setosa, while the other cluster contains both Iris virginica and Iris versicolor and is not separable without the species information Fisher used. This makes the data set a good example to explain the difference between supervised and unsupervised techniques in data mining.

In the same framework of the previous experiment for spam/non-spam, three methods are investigated. Here, we pretend that iris spaces (5th variable) are not measured. Fig. 3 shows the results.

The identical data set corresponding with the missing rate are used to evaluate three methods. Since the missing data structure depends on a seed of the randomization, RSS does not always increase monotonously. It may also be caused by the small sample size of 150. Despite the lack of monotonicity, our method ("rfImput.unspvised") is the best of the three, irrespective of the missing data rate. Rough imputation ("na.roughfix") is worst, naturally enough.

V. Semi-supervised learning

We point out that our method is easy for expanding to a semi-supervised data set, where both predictor (x) and response variables (y) may include missing values. In general, semi-supervised learning, including large amounts of response variables (y), has a potential to cover the real-world data considerably. A good semi-supervised learning method gives us many benefits. Our proposed procedures are as follows.

1) By starting the rough imputation for missing predictor (x), we estimate the missing response variables ($\hat{y}$) by running a random forest.
2) We replace the missing predictor ($\hat{x}$) by using the proximities between cases, and estimate the response variables ($\hat{y}$).
3) If the imputed vales ($\hat{x}$) are converged, we output them ($\hat{x}, \hat{y}$).

We call this procedure "rfImput.smspvsd," which means "an imputation method by using random forests for a semi-supervised data set." We found that the repetitive operation of 2) does not contribute significantly to improvement.

To evaluate the performance of "rfImput.smspvsd," we compare it with the following two methods.

1) Liaw's "rfImpute" [6]: Since "randomForest" does not work for y that includes missing responses, "rfImpute" functions as well. Therefore, we configure the forest model for non-missing response cases (y) by obtaining imputed predictor ($\hat{x}$) by using "rfImpute." Then, using this model, we estimate the response values ($\hat{y}$) for their missing y.
2) kNN [14]: We start the rough imputation of $\hat{x}$ for non-missing y, and a training kNN model is configured. Then, using this model, we estimate the response values ($\hat{y}$) for their missing y.

In semi-supervised as well as supervised learning, the prediction or estimation of y based on x is accomplished.

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-253-0

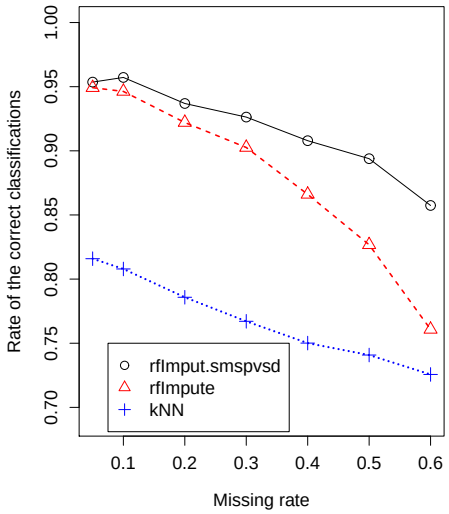

Figure 4. Correct classification for semi-supervised spam/non-spam data

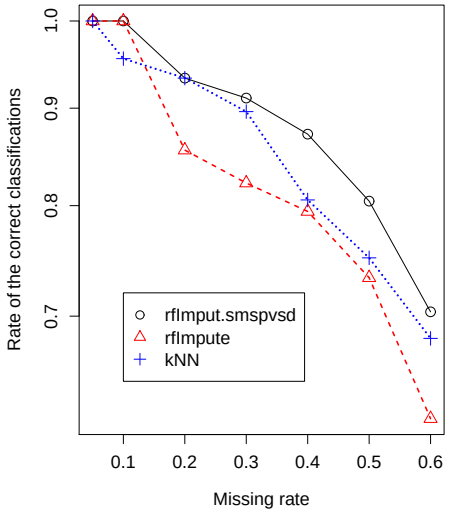

Figure 5. Correct classification for semi-unsupervised iris data

Therefore, as a criterion for evaluating the performance of learners, we use the precision, that is, the rate of the correct classifications.

The values of three methods are shown in Fig. 4 (spam/non-spam data) and Fig. 5 (iris data). A larger value on the vertical axis indicates a better performance. A value of 1 means that all missing y are completely predicted.

In general, the larger the missing data rate on the horizontal axis, the smaller the value on the vertical axis becomes. Due to the randomization of the missing data, the lines on the graph do not always decrease monotonously. Nevertheless, our method ("rfImput.smspvsd") is always the best of the three, irrespective of the missing data rate. In particular, in the case of the high missing data rate, e.g., 60%, the advantage of our method is remarkable.

Whereas spam data is alternative, iris data is a threefold choice. Therefore, the slopes of decreasing lines in the latter (Fig. 5) are sharper than those in the former (Fig. 4).

VI. Summary

For unsupervised data sets, the proposed method ("rfImput.unspvsd") works pretty well compared with the other conventional method: k-nearest neighbor imputation ("impute.knn") as well as the replacement by column median ("na.roughfix"). For semi-supervised data sets, our method ("rfImput.smspvsd") is also superior to the other two methods ("rfImpute" and "knn").

Since data imputation enables us to handle missing data the same as complete data, even statistical beginners can use this type of data easily. Speaking from a statistical point of view, our method makes an assumption called "missing at random (MAR)"[15], wherein the missing depends on only observations and not non-observations. The MAR is a more general assumption than "missing completely at random" wherein the probability of missingness is the same for all cases.

We should note that, even at a low missing data rate, e.g., 5% for spam/non-spam data, a complete case is rare. The occurrence probability is only $(0.95)^{57} \approx 0.0537$. The missing data rate of 10% in turn, yields an occurrence of 0.00246. If we use only complete cases by removing missing data, almost all cases should be avoided. Our method works effectively under the condition that the number of variables is rather large.

Moreover, our method does not take account of the effects on the data selection biases, because all cases can be available as they were. The situation or condition under which the complete data are obtained is often restricted. We hope that our method can be widely used in the future.

Indeed, the limitations of this method should be investigated. Especialy, the influence of cases in which MAR assumption is not satisfied, as well as the dependency of missing ratio and the number of variables, are significant. Because our method is based on the MAR assumption.

References

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

[2] L. Breiman and A. Cutler, Random Forests, http://www.stat.berkeley.edu/~breiman/RandomForests/ updated March 3, 2004.

[3] L. Breiman, *Manual for Setting Up, Using, and Understanding Random Forest V4.0*, http://oz.berkeley.edu/users/breiman/Using_random_forests_v4.0.pdf, 2003.

[4] The R Project for Statistical Computing, http://www.r-project.org/

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-253-0

[5] A. Liaw, Missing Value Imputations by randomForest, *R Documentation*, http://www.stat.ucl.ac.be/ISdidactique/Rhelp/library/randomForest/html/rfImpute.html

[6] CRAN, *Package randomForest*, http://cran.r-project.org/web/packages/randomForest/randomForest.pdf

[7] A. Pantanowitz and T. Marwala, Evaluating the Impact of Missing Data Imputation, *ADMA '09 Proceedings of the 5th International Conference on Advanced Data Mining and Applications*, pp.577–586, 2009.

[8] A. Rieger, H. Torsten and S. Carolin, Technische Reports **79**, Random Forests with Missing Values in the Covariates, Department of Statistics, Univ. of Munich, 2010.

[9] C. L. Nicholas and F. O. Andrew, yaImpute: An R Package for kNN Imputation, *Journal of Statistical Software*, **23** (10), Jan 2008.

[10] A. Liaw and M. Wiener, Classification and Regression by randomForest R News Vol. 2/3, 18–22, ISSN 1609-3631, 2002.

[11] UCI Repository of Machine Learning Databases, http://www.ics.uci.edu/~mlearn/MLRepository.html

[12] T. Hastie, R. Tibshirani, B. Narasimhan and G. Chu, impute: impute: Imputation for microarray data, R package version 1.14.0

[13] R. A. Becker, J. M. Chambers and A. R. Wilks, The New S Language. *Wadsworth & Brooks/Cole*, 1988.

[14] k-Nearest Neighbour Classification, R Documentation, knn {class}, http://stat.ethz.ch/R-manual/R-patched/library/class/html/knn.html

[15] A. Gelman and J. Hill, *Data Analysis Using Regression and Multilevel/Hierarchical Models*, Cambridge University Press, 2007.

APPENDIX

```
# Description:
#   Unsupervised data imputation using the
#   proximity from random forests.
# Usage:
#   rfImpute.unsupvsd(x, iter=10)
#
# Arguments:
#   x: An unsupervised data frame or matrix,
#      some containing 'NA's. Response vector
#      is not needed.
#   iter: Number of iterations needed to run
#      the imputation.
# Details:
#   The algorithm starts by imputing 'NA's
#   by using 'na.roughfix'. Then, 'randomForest'
#   is called with the completed data. The
#   proximity matrix from the randomForest is
#   used to update the imputation of the 'NA's.
#   Note that rfImpute(), developed by Andy Liaw,
#   has not (yet) been implemented for the
#   unsupervised case.
#
# Value:
#   A data frame or matrix containing the
#   completed data matrix, where 'NA's are
#   imputed by using the proximity from .
#   randomForest
#
# See Also:
#   'rfImpute', 'na.roughfix'
#
# Example:
#
# library(randomForest)
# data(iris)
# iris.na <- iris
# set.seed(111)
# ## artificially drop some data values.
# for (i in 1:4)
#   iris.na[sample(150, sample(20)), i] <- NA
# x <- iris.na[,-5] # Remove the 'Species'
# set.seed(222)
# irisImpute.unsupvsd <- rfImpute.unsupvsd(x)

rfImput.unsupvsd <- function (x, iter=5){
  x.roughfix <- na.roughfix(x)
  rf.impute <- x

  while (iter){
    x.rf <- randomForest(x.roughfix, ntree=100)
    x.prox <- x.rf$proximity

    for (i in 1:ncol(x)){
      rf.impute[,i] <- nafix.prox(x[,i],
          x.roughfix[,i], x.prox)
    }
    diff.rel <- dist.rel(rf.impute, x.roughfix)
    if (diff.rel < 1e-5){
      break
    }else{
      x.roughfix <- rf.impute
      iter <- iter -1
    }
  }
  return(rf.impute)
}

# Return relative distance between 'x.impute'
#   and 'x.org'
# Arguments:
#   x.impute: imputed data
#   x.org: original data
dist.rel <- function (x.impute, x.org){
  max.x <- lapply(abs(x.org), max) # normalize
  if (FALSE){ # available for only numeric
    diff.x <- (x.impute - x.org) / max.x
    diff.rel <- sum(diff.x^2) /
        sum((x.org / max.x)^2)
  }else{
    ncol.x <- length(max.x)
```

```
    mat.x.impute <- matrix(as.numeric
        (unlist(x.impute)), ncol=ncol.x)
    mat.x.org <- matrix(as.numeric
        (unlist(x.org)), ncol=ncol.x)
    max.numx <- as.numeric(unlist(max.x))

    diff.x <- sweep((mat.x.impute - mat.x.org),
        2, max.numx, FUN="/")
    size.org <- sweep(mat.x.org, 2, max.numx,
        FUN="/")
    diff.rel <- sum(diff.x^2) / sum(size.org^2)
  }
  cat ("diff.rel =", sum(diff.x^2), "/",
        sum(size.org^2), "=", diff.rel, "\n")
  return(diff.rel)
}

# Impute or revise NA elements by using the
#   data proximity.
# Arguments:
#   na.vales: data vector that includes NA;
#     unchanged.
#   rough.vales: rough data vector to be
#     replaced; NAs cannot be included.
#   x.prox: data proximity matrix; each
#      element is positive and <= 1.
nafix.prox <- function (na.vales,
        rough.vales, x.prox){
  if (length(na.vales) != length(rough.vales)){
    stop( "'na.vales' and 'rough.vales'
        must have the same length")
  }else if(length(rough.vales) != ncol(x.prox)){
    stop("'rough.vales' and 'x.prox' size
        incorrect")
  }
  # NA imputation ONLY for NA data
  na.list <- which(is.na(na.vales))
  replaced.vales <- rough.vales
  for (i in 1:length(na.list)){
    j <- na.list[i]
    x.prox[j,j] <- 0 # imputed datum itself
    replaced.vales[j] <- kWeighted.mean
        (rough.vales, x.prox[,j])
  }
  return(replaced.vales)
}

# Return k-neighbor weighted mean for numeric
#   variables or most weighted frequent factor
#   element for factor variables.
# Arguments:
#   value: vector; numeric or factor variables.
#   weight: vector; numeric.
#   k: the number of neighbors.
kWeighted.mean <- function(value, weight, k=10){
  if (missing(weight))
    w <- rep.int(1, length(value))
  else if (length(weight) != length(value)){
    stop("'value' and 'weight' must have the
        same length")
  }
  k <- min(k, length(value))
  if (is.numeric(value)){ # weighted mean
    order.weight <- order(weight, decreasing=T)
    ww <- weight[order.weight]
    vv <- value[order.weight]
    ret <- sum(ww[1:k] * vv[1:k]) / sum(ww[1:k])
  }else if (is.factor(value)){
    wgt.sum <- tapply(weight, value, sum)
    # most weighted frequent factor element
    ret <- names(subset (wgt.sum,
        wgt.sum == max(wgt.sum)))
  }else{
    stop("'value' is neither numeric nor
        factor")
  }
  return(ret)
}
```

Figure 6. R program to impute the missing unsupervised data