GRAFT, a complete system for data fusion

Tomàs Aluja-Banet\textsuperscript{a,}\textsuperscript{*}, Josep Daunis-i-Estadella\textsuperscript{b}, David Pellicer\textsuperscript{c}

\textsuperscript{a}Universitat Politècnica de Catalunya, Jordi Girona Salgado 1-3, CN C5204, E-08034 Barcelona, Spain
\textsuperscript{b}Universitat de Girona, Campus de Montilivi, Edifici P4, E-17071 Girona, Spain
\textsuperscript{c}TNS Audiencia de Medios, Camí de Can Callers núm. 4, E-08173 St. Cugat del Vallès, Spain

Available online 13 December 2006

Abstract

Data fusion concerns the problem of merging information coming from independent sources. Also known as statistical matching, file grafting or microdata merging, it is a challenging problem for statisticians. The increasing growth of collected data makes combining different sources of information an attractive alternative to single source data. The interest in data fusion derives, in certain cases, from the impossibility of attaining specific information from one source of data and the reduction of the cost entailed by this operation and, in all cases, from taking greater advantage of the available collected information. The GRAFT system is presented. It is a multipurpose data fusion system based on the \textit{k}-nearest neighbor (\textit{k}-nn) hot deck imputation method. The system aim is to cope with many data fusion problems and domains. The \textit{k}-nn is a very demanding algorithm. The solutions envisaged and their cost, which allow this methodology to be used in a wide range of real problems, are presented.

© 2006 Elsevier B.V. All rights reserved.

Keywords: Data fusion; Statistical matching; Imputation; \textit{k}-nn

1. Introduction

Data fusion involves the imputation of a complete block of missing variables in independent files. There are various situations in which data fusion can be envisaged: parallel (or shared) questionnaires; self-administered questionnaires with face-to-face re-interview of a part of them; enrichment of a specific questionnaire from a base survey; panels, etc. (see Santini, 1984 for a more exhaustive listing).

The interest in data fusion is increasing every day. Its main application is in media studies where it is used to combine consumption panels with audience panels (Rius et al., 1999; Lejeune, 1995; Aluja and Thió, 2001), but also in web data, where it is used to combine Internet flow data with standard survey questionnaires, and in national statistical institutes, where reducing the increasing burden caused by official statistics is becoming a difficult problem to solve (D’Orazio et al., 2006).

First, we will introduce the key points of data fusion. In the second section we present the GRAFT system and its architecture, and in the third section we introduce the \textit{k}-nn search module and its cost using simulated data. To validate the performance of the algorithm we present the cost obtained with simulated data and an actual data fusion operation of web data with survey data. Finally, in the fourth section, we present the imputation module, its functionalities and a algorithm for constrained imputation. In this paper we do not compare the \textit{k}-nn classifier with other alternatives,
because we focus on the computational issues of the algorithm and its extension to the conditional search and constrained imputation, which is not available in the other usual statistical k-nn classifiers.

1.1. Concepts of data fusion

We address the problem in its simplest case, called unilateral fusion, in which there are two files, one with complete information, $X$ and $Y$ variables (donor file), and the other with just the $X$ variables (recipient file). The $X$ variables are called common, while the $Y$ variables are the specific ones. The objective of the data fusion is to transfer the specific variables of the donor file to the recipient file at individual level. Data fusion is related with the problem of missing data (Little and Rubin, 1987), with the difference that now the missing values correspond to variables missing by design and both files do not need to be representative samples of the same population. However, in order to perform a valid transfer of information, we need the donor file to be representative of the parent population from which inferences can be taken. Because missingness is induced by design, the typology of missing data patterns established by Rubin (1976) does not make sense. However, since we are estimating the missing data under the conditional independence assumption $f(Y/X, Z) = g(Y/X)$ for any set of variables $Z$, we are implicitly assuming missing completely at random (MCAR) in case donor and recipient files are samples of the same population or missing at random (MAR) in the opposite case. Missing data techniques aim mainly to solve the problem of nonresponse in surveys and of computing global statistics, taking into account the uncertainty of the missing data, whereas in data fusion we also expect to find plausible individual data for the specific variables of recipients. In addition, there is a fair amount of overlapping with the so-called record or exact matching (Winkler, 1995), where the objective is to locate information belonging to the same individual in different sources and merge it, while in data fusion we merge information coming from similar individuals.

1.2. The data fusion methodologies

There are three basic approaches for data fusion. The first one consists of embedding the common and specific variables within a parametric multivariate distribution $f(X, Y/\theta)$, assuming donors and receptors independently and randomly draw from this distribution. This distribution can be factored into $f(X, Y/\theta) = f(Y/X, \theta_{Y/X}) f(X, \theta_X)$; hence, it is possible to estimate its parameters $\theta_X$ and $\theta_{Y/X}$ from the available information and use them to impute the missing block of data. The second approach consists of directly modelling the relationship between the $Y$ variables and the $X$ variables in the donor file: $f(Y/X) = r(X) + \epsilon$ and applying this model in the recipient file (explicit modelling). The last approach consists of finding for each individual of the recipient file one or more donor individuals as similar as possible, and then in some way, transferring the values of the $Y$ variables to the recipient individual (implicit modelling). This method is known as hot deck, a term borrowed from data editing in data bases.

The parametric imputation is bound to the missing data problem. It assumes a common distribution $f(X, Y/\theta)$ from which the donor and recipient file are random samples. Then it maximizes the observed likelihood or, more commonly, uses the expectation maximization (EM) (Dempster et al., 1977) or data augmentation (DA) algorithm (Schafer and Olsen, 1998). The EM algorithm is intended to estimate the parameters of the distribution, without incorporating the variability of the estimated parameters like the DA algorithm, which intends to simulate instances of the distributions allowing variability of the parameters themselves. The EM or DA algorithm can be easily performed for a multinormal distribution in cases of continuous specific variables or a multinomial distribution in cases of categorical variables.

Explicit modelling can be performed by simultaneous multiple regression, principal component regression, partial least squares (PLS) regression, neural networks, multivariate adaptive regression splines, etc. It skips the problem of formulating a parametric model, although it is easy to show that in the case of multivariate normality both approaches are equivalent.

Hot deck is the simplest method; it requires no assumption about the probabilistic distribution or about the formal relation between the specific and the common variables. It is a data-based method, and in this sense it is distribution free. It can be a random hot deck when the donor (or donors) are selected at random within a specific group sharing some characteristics with the recipient, or a distance hot deck, better known as the k-nn method, where the donor and recipients are placed in a common subspace defined from the common variables and then for each recipient its $k$-nearest donor neighbors are found and listed. The assignment is finally done on the basis of this list. Clearly, hot deck methodology implies performing random draws from the empirical conditional distribution $f(Y/X, \theta_{Y/X})$. 
Whatever the method chosen, the quality of the results depends mainly on the predictive power of the common variables with respect to the specific ones.

2. The GRAFT data fusion system

The GRAFT system is a multipurpose data fusion system that attempts to cope with many data fusion problems and domains. It has been developed within the ESIS project (http://www.esisproject.com, it runs on a Windows platform in a PC environment, coupled with the SPAD software, http://www.spadsoft.com). It uses the k-nn hot deck imputation method as global framework for the fusion. Imputation from a neighbor is a very simple idea, which implies finding a proxy for each recipient. The k-nn can also be presented as a nonparametric regression or classification, where the neighborhood consists of k points.

The k-nn fusion is founded on statistical grounds. Assuming smoothness of the joint probability distribution \( P(X,Y/\theta) \), the regression function \( r(x) = E[Y/X] \) in a point \( x_i \) can be approximated by

\[
r(x_i) \simeq \text{ave}[r(x), x \in L(x_i)],
\]

where \( L(x_i) \) stands for a neighborhood of the point \( x_i \), since the points in the neighborhood are likely to be close to \( x_i \). Asymptotically it tends to its true value when the number of neighbors, \( k \), tends to infinity whereas the ratio \( k/n \) tends to zero, \( n \) being the total number of instances. The local mean and local variance of continuous variables in a point \( i \), then, tend to the true conditional expectation and variance in this point:

\[
\hat{y}_i \to E[y|x_i] \quad \text{and} \quad s_i^2 \to V[y|x_i] \quad \text{for a continuous variable},
\]

where \( \hat{y}_i \) is the local mean of a specific variable in point \( i \) and \( x_i \) are the values of common variables in this point. For categorical specific variables, the proportion of cases in a class \( j \) tends to the density of this class in that point \( i \):

\[
\hat{p}(y=j) = \frac{l_j}{n_j V_i} \to p(y=j) \quad \text{for a categorical variable},
\]

where \( l_j \) stands for the number of neighbors of class \( j \), \( n_j \) stands for the number of donors of class \( j \) and \( V_i \) stand for the volume of the k-nn around the point \( i \) (see Fix and Hodges, 1951). This result has also been revised by Friedman (1994) and Hastie et al. (2001), allowing the k-nn to be considered as a universal estimator. Another asymptotic result was attained by Cover and Hart (1967), setting an upper bound to the error rate of the 1-nearest neighbor by twice the Bayes error rate (see also Ripley, 1996).

In practice, however, these asymptotic conditions are never fulfilled. The number of instances is far below what is required to consider the space dense enough (each dimension implies multiplying by \( n \) the total number of instances to preserve the density, making the sizes needed for 10 dimensions or more completely unrealistic). This is known as the curse of dimensionality and reveals several shortcomings of the k-nn methodology.

The k-nn methodology is one of the most widely used method for data fusion. It allows a general framework of flexible imputation to be built, it can deal with any type of variable (continuous, categorical or textual), it allows the use of the same neighbor table for different grafting operations instead of estimating different models for each operation, and it allows confidentiality to be kept on the specific variables, since the imputation can be done apart from the neighbor search. It obtains imputations conditionally on common variables, thereby reducing bias and preserving the association between specific and common variables. Also, it assures coherent imputations to prevent nonsense values and, finally, it can allow imputation by random draws from a predictive distribution to simulate real data and then allow multiple imputations to be performed to take into account the uncertainty of imputation.

The drawbacks of the method are that the individual accuracy of the imputations is generally lower than other imputation methods (for instance, by explicit modelling or by EM) since it just takes into account local information, which as a result of the curse of dimensionality can be very scattered, biasing the approximation of relation (1). Also, it presents a black hole effect, that is, neighbors tend to be concentrated in the direction of the high-density zone of points, which implies, in cases of correlation of specific variables with the common ones (the interesting case), an imputation bias. We call this a black hole effect because imputations tend to concentrate together in the high-density zone of the cloud of points. Fig. 1 illustrates a typical example of this effect, showing the first factorial display of the PCA of the specific variables of donors (in empty points) together with the imputed values of the recipients as illustrative (in solid
points), when using the local mean with \( k = 10 \). The imputed values tend to be concentrated in the center of the graphic, not only because they are means of 10 points, but also because the majority of neighbors tend to be concentrated toward the high-density zone. This effect is always present but it increases with larger values of \( k \).

2.1. The GRAFT architecture

The data fusion operation consists of a sequence of procedures, which we call generically MULTIV and CLUSTER, followed by the GRAFT system itself, each performing specified operations connected through internal files. The architecture of the system is the following:

2.1.1. Previous procedures

The MULTIV procedure places the donor and the recipient individuals in a common subspace. This common subspace is defined from the common selected variables of the donor file. Several multivariate procedures could be appropriate, depending on the nature and the coding of the common variables: for example, principal component analysis and simple or multiple correspondence analysis. Projection upon components oriented to modelling, such as PLS components (Tenenhaus, 1998), would also be appropriate.

The CLUSTER procedure is intended to perform a hierarchical clustering of donors, since the Fukunaga & Narandra algorithm is a branch-and-bound algorithm and it needs to have the donors structured in a tree fashion. We have used the hierarchical clustering with the Ward criterion, since it usually gives spherical clusters, but any other criteria can be used provided that they deliver a hierarchical tree. Only the upper part of the obtained tree is stored, up to a specified \( nkla \) number of terminal nodes. For each node of the tree, the following elements are stored:

- the centroid of the node as its representative \( m_i \);
- the radius of the node \( r_i \), as the maximum of the distances \( d(m_i, x_{0i}) \) of the representative of the node to its elements;
- the list of pairs of donors belonging to the node with their distance to its representative \( (x_{0i}, d(m_i, x_{0i})) \).

2.1.2. Modules of the GRAFT system

The GRAFT system is divided into two main modules:

- **K-NN module**, whose objective is to produce a neighbor table between recipients and donors, using the factorial coordinates and the tree issued of the previous procedures.
Donors

Recipients

MULTIV procedure

Factorial Coordinates

CLUSTER procedure

Tree

Fig. 2. The data fusion chain.

Imputed data

IMPUTATION module

Neighbor table

k-NN module

GRAFT System

Fig. 3. Searching for neighbors in the F&N algorithm.

- **Imputation module**, whose objective is to fill in the specific variables in the recipient file, using the neighbor table obtained in the K-NN module and the input raw data.

These two modules are assembled in a unique interface forming the GRAFT procedure. The GRAFT procedure runs with the SPAD package, using the functionalities of SPAD to run the first two procedures of the data fusion operation (Fig. 2).

3. **K-NN module**

The objective of the K-NN module is to create a neighbor table, relating each recipient to a list of its $k$-nn donors. We denote in capital letters the actual performed implementation.

Once we have all individuals in the same reference subspace (output of the MULTIV procedure), we search for the $k$-nn donors for each recipient. For that purpose, we use the algorithm proposed by Fukunaga and Narendra (1975). This is a branch-and-bound algorithm using the tree built by the CLUSTER procedure and following two rules of pruning, the first on the nodes of the tree and the second on the individuals of the selected nodes.

**Rule 1**: A node $t$ cannot contain a closest neighbor to a given recipient if

$$d(x_{1j}, \text{knn}) + r_t < d(x_{1j}, m_t),$$

where $x_{1j}$ are the coordinates of recipient $j$, $\text{knn}$ is its current $k$ nearest neighbor, $m_t$ the representative of node $t$ and $r_t$ is its radius.

**Rule 2**: For an eligible node $t$, an individual $x_{0ti}$ cannot be closer than the current $\text{knn}$ neighbor if:

$$d(x_{1j}, \text{knn}) + d(m_t, x_{0ti}) < d(x_{1j}, m_t).$$

Fig. 3 illustrates the application of the aforementioned rules.

The parameters of the algorithm will be the number of donors and recipients, $n_0$ and $n_1$, the dimensionality $naxe$ of the common space, the number of terminal classes $nkla$ of the tree and the number of neighbors $k$ to compute.
3.1. Conditional search

Very often it is useful to assure that recipients and donors share a certain list of conditions, to avoid incoherencies, to respect the sampling design of donors or to just improve the imputation stage. (For example, the user may want the donors and the recipient to share the same interval of age, labor condition, etc.) This is achieved by declaring a list of conditions to be fulfilled for the neighbors. This means conditioning the search for neighbors within the cells of a hypercube, defined by the variables to preserve.

This implies distinguishing between the common variables used to define the common subspace of the search, which we try to keep similar between donors and recipients, and the list of variables that should be equal between donors and recipients. This latter group is usually called critical variables.

One problem arising from imposing a long list of conditions is that they define a huge and sparse hypercube, so it might happen that the requested number of \( k \) neighbors is larger than the actual number of donors fulfilling the conditions. To handle these situations, we have adopted two strategies, depending on the relative importance of the conditions. If all the conditions have the same importance, we relax the list of conditions freeing one or more of them, but not necessarily the last ones, in order to find the \( k \) closest neighbors maximizing the number of conditions accomplished per donor. Then we take as the \( k \) closest neighbors the list of donors ordered by number of accomplished conditions and distance. But, if the conditions have different degrees of importance, we rank them according to it, in such a way that the first condition is the most important to be fulfilled and so on; then we evaluate the number of conditions sequentially, starting from the first one.

Consequently, we add to the parameters of the module the list of conditions (ordered or unordered) to be accomplished and in the output we obtain the number of fulfilled conditions for each recipient (see Algorithm 1).

**Algorithm 1.** Algorithm \( k \)-nn with conditional search.

**Input** \{ the Tree \( T \) obtained in the CLUSTER procedure (stored up to \( nkla \) terminal classes), the representative of nodes \( m_t \), their radius \( r_t \), the list of members of the terminal nodes of the tree with their distance to its representative \( (x_{0ti}, d(m_t, x_{0ti})) \), and the coordinates of recipients \( x_{1j} \) in the common subspace), list of conditions (optional) \};

**for** each receptor \( x_{1j} \) **do**

  - Choose at random \( k \) donors, and sort them according their number of conditions fulfilled and distances to \( x_{1j} \);
  - **for** each non-discarded node \( t \) of the tree \( T \), following a preorder sequence **do**
    - **if** \( t \) is an internal node **then**
      - Apply Rule 1 to \( t \) and if it is true discard \( t \) and its descendants;
    - **else** \{ \( t \) is a terminal node \}
      - **for** each individual \( x_{0ti} \) in \( t \) **do**
        - Evaluate the number of conditions fulfilled;
        - **if** this number is less than the actual number of conditions fulfilled by the last \( knn \) neighbor, **then** discard it;
        - **if** this number is equal than the actual number of conditions fulfilled by the last \( knn \) neighbor **then**
          - apply Rule 2 to \( x_{0ti} \);
          - **if** it is true **then**
            - discard it;
          - **else**
            - Compute \( d(x_{1j}, x_{0ti}) \);
            - Compare \( d(x_{1j}, x_{0ti}) \) with the actual last distance, \( d(x_{1j}, knn) \) of the list of neighbors of \( x_{1j} \) and update this list if \( d(x_{1j}, x_{0ti}) \) is less than \( d(x_{1j}, knn) \);
          - **end if**
        - **end if**
      - **end for**
    - **end if**
  - **end for**
**end if**
if this number is greater than the actual number of conditions fulfilled by the last \textit{knn} neighbor, then update the list of the closest neighbors with the distance $d(x_{1j}, x_{0ti})$;
end for
end if
end for
end for
Output \{ Table of Neighbors containing for each receptor $x_{1j}$ the list of $k$ closest donors $x_{0ti}$ with their distance $d(x_{1j}, x_{0ti})$ to the receptor, ordered by number of conditions fulfilled and distance \}.

3.2. Cost

The cost of the K-NN module depends on the structure of the data file, so we must verify it empirically.

It is clear that the algorithm is scalable with respect to the number of recipients, so we will evaluate its cost per recipient with respect to the parameters of the algorithm: the dimensionality of the search space $naxe$, the number of terminal nodes of the tree $nkla$, the number of neighbors $k$, and the number of donors $n_0$.

We have performed a simulation of 10 000 cases of 30 uniform variables, which we have split randomly into two sets of 5000 each, one forming the donors and the other forming the recipients. The uniform distribution represents a neutral situation where all the points are distributed randomly in the search space. In practice, the distribution of points will very rarely fit the uniform assumption, but this is the most natural choice for evaluating the cost of the algorithm, although with real data the patterns showed here must be taken as an approximation.

3.2.1. Cost by dimensionality

We have executed the K-NN module for an increasing number of dimensions, from 1–30, with three different values of $nkla$ terminal nodes of the tree: one in 50 terminal classes, another in 500 classes and a third in 5000, which means using the complete tree for the search. Fig. 4 presents the number of distances computed in the search of the closest neighbor ($k = 1$) within 5000 donors.

We can observe that the number of distances per recipient increases with dimensionality following a sigmoidal pattern. First, they increase exponentially and then stabilize at a certain maximum, which depends on the total number of donors and the nodes of the tree ($n_0 + 2*nkla - 2$); this is the maximum number of distances necessary to explore the whole tree. Comparing the curves with the cost of the raw algorithm (of cost $n_0$), we see that for low dimensionalities the $k$-nn algorithm obtains a better result, but for larger dimensionalities it is more costly than the simple raw algorithm.
Thus, dimensionality is a crucial aspect of the $k$-nn algorithm, which only performs well in low-dimensional spaces and is rather inefficient in a higher dimensionality. This deterioration comes from the fact that in high dimensionality the search space becomes empty, causing the radius of nodes to overlap between classes and leading to the exploration of all terminal nodes, that is, all the space. In such a situation there is no gain with respect to the raw algorithm. This is the well-known problem of the curse of dimensionality which strengthens the importance of incorporating a previous reduction of dimensionality to regularize the data and to push the algorithm forward. Also, comparing the three curves for a different number of terminal classes, we see that, for low dimensionalities, performing the branch and bound in the complete tree ($nkla = n_0$) gives better results, although they are very similar to those obtained with a large tree (for instance, with 500 terminal classes). For high dimensionalities, however, taking the complete tree is clearly the most costly situation, because of the extra exploration of the nodes of the tree.

3.2.2. Cost by the number of terminal nodes of the tree

As we have seen, the number of terminal nodes of the tree has an influence on the cost of the $k$-nn search. In order to explore this influence further we have performed a simulation taking at random 5000 donors and recipients, with 5 dimensions, which is a nice dimensionality for the $k$-nn algorithm. Then we executed the $k$-nn algorithm for the closest neighbor, taking a tree with an increasing number of terminal nodes, from 3 to 500, obtaining the results of Fig. 5.

We can observe that the cost decreases with the number of terminal classes, obtaining a sharp decrease in the beginning, when we start to increase the number of terminal classes, and stabilizing at a minimum, a function of the depth of the tree and the dimensionality. An exponential decrease with the number of terminal classes can be appreciated, and we are therefore interested in performing a search in trees with a fairly large number of terminal classes (in this case 250), which is almost equivalent to using the complete tree. With a large number of $nkla$ terminal classes, their radius will be less, we will have less overlap between nodes and we will avoid exploring useless nodes. This situation reverses with large dimensionality: for $naxe = 30$, the complete dimensionality, the radius increases (see Friedman, 1994) and forces exploration of the whole search space.

3.2.3. Cost by the number of neighbors

To evaluate the cost of finding a list of $k$ neighbors per each recipient, we have split our data set into 5000 donors and 5000 recipients taken at random, with 5 dimensions and 250 terminal classes of the tree. We have then performed several simulations with an increasing value of $k$, from 1 to 50.

The results are displayed in Fig. 6.
We can observe an increasing cost, that is, the more neighbors we are interested in, the more costly is the algorithm, following a logarithmic pattern $O(\log_2(k))$. In this case we can see that finding 50 neighbors implies an increase of approximately three times in the cost of finding the closest neighbor.

3.2.4. Cost by the number of donors

This is a critical parameter in a data fusion operation. To evaluate its influence in the cost, we have taken 5 dimensions and 250 terminal classes and we have performed several simulations, splitting our data file and changing the number of donors from 250 to 9000. The obtained results of the distances per recipient are displayed in Fig. 7.

Clearly, we can appreciate that the cost of the algorithm is logarithmic with the number of donors $O(\log_2(n_0))$. The observed variability is due to the fact that we are changing the set of donors in each run. Thus, a high increase in the
number of donors does not imply a high increase of the cost. We have added, for the sake of comparison, another figure, including the cost of the raw algorithm which is $O(n_0)$. This nice behavior is probably the most outstanding feature of the $k$-nn algorithm, but it fades when we increase the dimensionality.

### 3.3. Costs of the K-NN module applied to real data

To evaluate the performance of the $k$-nn algorithm with real data we have applied it to the fusion of the web data with socio-economic information within the framework of the DIASTASIS project (“Digital era statistical indicators”, IST-2000-31083, [http://www.eurodyn.com/diastasis](http://www.eurodyn.com/diastasis)). The common variables correspond to categorical attributes collected in a survey of university personnel, who provided information about personal characteristics and their awareness of Internet use. The specific variables correspond to actual navigation on the Internet collected by a sniffer from a sample of volunteers; this information has been aggregated into 22 main Yahoo categories.

To see the behavior of the $k$-nn algorithm, we present the number of distances computed per recipient when computing the neighbor table with the K-NN module, varying the parameters of the algorithm according to the levels of the Table 1.

By crossing these levels we define 588 experimental cells and for each cell we have performed 5 runs choosing the donors at random. On the other hand, the average number of donors per cell is 4770. Thus, this would be the number of distances computed (on average) with the raw algorithm, and hence it provides a baseline with which to compare the achieved results.

Since we have seen the importance of the dimensionality in the cost of the algorithm, we present the distances computed per recipient according to the different number of axes used in the algorithm. The total number of axes available is 84, since the common variables are categorical, but with 10 axes we kept up to 73% of the information from the global cloud of points and with 25 axes we arrived at 98%.

Looking at the influence of the number of axes and the number of terminal nodes, we confirm the increasing pattern of cost according the number of dimensions, showing in this case an exponential increase for the dimensions selected (Fig. 8); and a decreasing pattern of cost with the size of the tree (parameter $nkla$) up to a certain size, where the cost stabilizes or even increases a little bit. Hence, in this case, from a computational point of view, we could be interested in working with 10 dimensions and a large tree with 250 terminal nodes.

Looking at the influence of the number of neighbors and the number of donors (see, Fig. 9), we just appreciate a logarithmic increase when working with 25 dimensions, whereas in the other cases we observe a linear trend, but far away of the cost of the raw algorithm.

### 4. Imputation module

The imputation stage means imputing the specific variables of recipients from the values of their neighbors.

The GRAFT system implements different imputation possibilities in order to cope with different objectives of the data fusion procedure. These possibilities refer to different choices of parameters for the imputation module. These parameters are:

- **Value of $k$ for imputation** ($k = 1$ or $k > 1$): That is, how many neighbors from the available list of neighbors should be taken into account? Taking $k = 1$, the user specifies that he/she wants the direct transfer of values from one donor to the recipient; this implies that the imputed values would be real observed ones. The second choice corresponds to cases in which the user wants to take into account the information in the $k$ neighbors to make the imputation. Obviously, the value of $k$ should be less or equal to the available list of donors contained in the table of neighbors.

- **Value of $p$**: Probability to repeat a donor when $k = 1$. With $p = 1$ it performs an unconstrained imputation. With $p = 0$ it performs a constrained imputation per each recipient (fastRN algorithm explained later). With $p$ between both bounds, it indicates the probability of repeating a donor.
Deterministic or stochastic imputation: The first option leads to a unique determination of the imputed values, whereas in the second option we add some random fluctuation in order to simulate real data.

Combining these options we obtain different possibilities for imputation. In any case, the imputation is multivariate, in order to preserve the local correlations that may exist.

$1D (k=1, \text{deterministic})$: We transfer the specific values from the nearest neighbor. The imputation can be constrained, unconstrained or somewhere in between, depending on the value of parameter $p$ of the algorithm fastRN. The constrained imputation will allow the marginals and the correlations to be preserved, since we maximize the number of donors used, but the unconstrained option gives better accuracy values, since unconstrained imputation minimizes the distances between recipients and their donor. In any case, this option is the simplest one, it gives reasonable results and assures the coherence of the imputed values.
1S \((k = 1, \text{stochastic})\): We can add some random fluctuation to the previous method, by simply selecting, not the closest donor, but a donor selected by roulette wheel from the \(k\) list of donors, with probabilities being the inverse function of the distances to the recipient. This option performs a local resampling procedure and is therefore interesting when we are mainly interested in simulating real data. This implies worse accuracy with respect to 1D since the closest neighbor is a better estimator, but it preserves the marginals and correlations.

\(kD\) \((k > 1, \text{deterministic})\): Imputing from one neighbor implies not taking advantage of the information collected from the remaining ones. Thus, it makes sense to impute from \(k\) neighbors by the local weighted mean for continuous variables and by the local weighted mode for categorical variables. The weighting depends inversely on the distance of donors to their recipients. This is equivalent to performing a conditional local imputation. Hence, we improve the accuracy with respect to the previous options, but we do not generate real data, since it increases the correlations of imputed data and decreases the variance of imputed values, since we are imputing by a conditional mean. Moreover, we do not preserve the marginals, since in this case the black hole effect is more evident, producing an imputation bias.

\(kS\) \((k > 1, \text{stochastic})\): To soothe the shortcomings of the preceding method, we can perform imputation by random draw from the multivariate distribution of specific variables. We assume multivariate normality for continuous variables or a multinomial distribution for categorical variables. The multivariate distribution is computed from the global distribution adjusting for the local marginals. In this way, we preserve the marginals and the correlations among variables but we get worse accuracy than with the previous method. An interesting feature of \(kS\) is that we can use it to simulate real data, which can be used to perform multiple imputations, although the imputed data may take unrealistic values for some variables.

### 4.1. Cost of the imputation stage

The cost of the imputation module depends on the type of imputation and its parameters. These parameters are:

- \(n\): number of individuals to impute. Although in general they should only be the recipients, we impute the donors as well in order to evaluate the accuracy of the method;
- \(k\): number of neighbors used for the imputation;
- \(v\): number of specific variables imputed.

**Cost of the 1D method** (unconstrained): The cost of imputation is the cost of going round all the individuals \(O(n)\).

**Cost of the 1S method**: This case is similar to the previous one, but selects one neighbor at random instead of taking the nearest one; the cost will be \(O(n)\).

**Cost of the kD method**: In this case, apart from the loop upon the individuals, we need to go round all the selected neighbors and for each one of them to the specific variables. Thus, the cost will be \(O(n \cdot k \cdot v)\).

**Cost of the kS method**: In this case we must differentiate between imputing continuous variables or imputing categorical variables.

For the first case, its cost is similar to the previous one \((kD)\) plus the cost of diagonalizing the global correlation matrix, which is \(O(v^3)\), so the final cost is \(O(n \cdot v \cdot k + v^3)\).

In cases in which we are imputing categorical variables, for every individual we need to make random draws from the hypercube defined by its \(k\) neighbors. Hence, for every individual we need to compute its corresponding hypercube, whose cost is \(O(k \cdot \log(k))\), so the cost will be \(O(n \cdot k \cdot \log(k))\).

Thus, considering the cost of imputation per individual and in the usual case of having \(k \ll n\) and \(v \ll n\), we can see that the cost of the imputation module is very low.

### 4.2. Constrained imputation in 1D

Imputation from the closest neighbor may run into trouble because the same donor can be repeated for several recipients. This causes a progressive deterioration of the mean and variance of imputed variables, faking their marginal values. As a result, the correlations with the imputed variables do not reflect the true ones due to the regularity (the repetitions) introduced in the imputed variables, decreasing the reliability of the imputed data. This problem can be solved by constrained imputation, which implies not repeating the same donor for different recipients or repeating it a minimum number of times.
For that purpose we have implemented a constrained version of the closest neighbor imputation. We have addressed this problem by searching the list of donor and recipient pairs, minimizing the repetitions of donors and minimizing the total sum of distances

$$\min \sum_{i=1}^{n_1} d(x_{1j}, x_{0,1nn}).$$

It is clear that the donor and recipient pair with minimum distance will be an eligible couple. The problem arises when the closest donor for a subsequent recipient has been used previously. Hence, we have designed a fast algorithm to locate, in such cases, the unused donors assuring minimum distance, using the neighbor table issued from the K-NN module. We call this algorithm FastRN.

The idea is to initialize the list of pairs, taking for each recipient the first element of the table of neighbors obtained by the K-NN module, then locating the pair with minimum distance, eliminating it from the search space, and continuing the search for the pair with minimum distance. If one pair has a donor already used, this donor is simply replaced by the next one listed in the table of neighbors. We call the donor obtained for a given recipient its reciprocal neighbor. Since the search is not done in the entire space of \( n_0 \) donors, but only within the list of the \( k \) closest ones for each recipient in the neighbor table, this algorithm does not guarantee that the obtained donor will be optimal. In practice, for a large enough value of \( k \), the results would be very similar if not equal. (The results obtained would be undistinguishable from a practical point of view.)

In this algorithm the donors are repeated when necessary: when they are less than the recipients and also when the list of \( k \) donors for a given recipient has been exhausted. This algorithm does the matching of two sets, the donors and the recipients, not necessary of the same size, assigning one recipient to one donor (or more donors in case of necessity).

**Algorithm 2.** FastRN algorithm.

**Input** {The list of recipients \( x_1 \), The Table of Neighbors \( N \) containing for each recipient \( x_{1j} \) the list of the \( k \) closest donors \( x_{0ti} \) with their distance \( d(x_{1j}, x_{0ti}) \) to the recipient, ordered by number of conditions fulfilled and distance};

**Compute** \( \text{maxDonations} \), the maximum number of repetitions allowed in case that the number of recipients exceed the number of used donors in the Neighbor Table \( N \);

**Form** the list of pairs, joining every recipient with its first neighbor of the table \( N \),

\( (\text{recipient } (x_{1j}) = r_j, \text{first donor } (\text{Inn} ) = d_{j,1}); \)

**Insert** this list of pairs \((r_j, d_{j,1})\) into the \( \text{Heap} \) with a key equal to the distance between the recipient and the donor \( d(r_j, d_{j,1}); \)

1. **for** each donor initialise their number of donations equal to 0

   **while** \( \text{Heap} \) is not empty **do**

   Get the pair \((r_j, d_{j,i})\) of the top of the heap and its key;

   Set the recipient \( r_j \) of the pair;

   **if** the recipient \( r_j \) of the pair does not have reciprocal neighbor, **then**

   **if** the number of donations of the donor \( d_{j,i} \) is less than \( \text{maxDonations} \)

   then

   Reciprocal of recipient equal to its donor of the pair \( \text{RN}(r_j) = d_{j,i}; \)

   Update the number of donations of donor \( d_{j,i}; \)

   **else**

   Insert in the \( \text{Heap} \) the pair formed by the recipient with the following neighbor of the list of the table \( N \); \((r_j, d_{j,i+1}); \)

   **if** we have exhausted the available donors of the recipient in its list of neighbors of table \( N \), **then**

   insert the pair formed by the actual recipient with its first donor in the table \( N \) \((r_j, d_{j,1})\) into the \( \text{recycleHeap} \);

   **end if**

   Remove the top of the \( \text{Heap} \);

   **end if**

   **end while**
if recycleHeap is not empty then
    Heap = recycleHeap;
    goto (1)
end if

Output {The list RN giving for each recipient its selected neighbor}.

To add randomness to the repetition of donors, we have included in the algorithm a parameter $p$ specifying the probability of repetition for a given donor. If we set $p = 0$, we obtain the aforementioned constrained imputation, whereas with $p > 0$, we allow some donors to repeat with probability $p$, and with $p = 1$ we take the closest donor for every recipient, irrespective of the number of repetitions that may have occurred (unconstrained imputation).

4.2.1. Cost of the FastRN algorithm

The list of pairs is held in a heap structure, thus the cost of this algorithm is the cost of the insertions and deletions in the heap structure, which is logarithmic (Ahuja et al., 1993). It consists of first introducing all the pairs in the list of the heap, which is $O(n \cdot \log(n))$, and then (if we have enough donors) extracting one by one each pair recipient-heap neighbor has a cost of $O(n \cdot \log(n))$. The total cost is thus $O(2n \cdot \log(n))$. Hence, to pass from the unconstrained imputation to constrained one implies to multiply by $\log(n)$ the cost of the algorithm, whereas the classical implementation of the constrained imputation consists of running the assignment problem (assuming equal number of donors and recipients, a special case of the transportation problem) with cost $O(n^3)$ (Papadimitriou and Steiglitz, 1982).

5. Conclusions

We have presented a complete system for data fusion, which we call the GRAFT system, based on $k$-nn methodology, including conditional search of neighbors. We have analyzed the computational cost of the Fukunaga–Narendra implementation with respect to the parameters of the algorithm—dimensionality, size of the tree, number of neighbors and number of donors; using simulated and actual data. The results obtained highlight the importance of the dimensionality and the number of terminal nodes of the tree. We have seen that for low dimensionality the results are far superior to those obtained with the raw algorithm; also, working with larger trees decreases the cost, up to a certain size where the cost stabilizes making useless to work with the complete tree. In our case using trees of about 250 terminal nodes gives almost the same cost of using the complete tree. On the other hand, with a convenient low dimensionality, we have seen that the numbers of neighbors to compute or the number of donors are not very important regarding the cost. We have presented the FastRN algorithm to perform a constrained imputation; although this algorithm cannot guarantee the global optimum, it is very fast since it performs the search within the neighbor table issued by the K-NN module.

Thus, we can conclude that the GRAFT system is highly efficient in a reduced space, using a large tree, irrespective of the number of donors and the number of neighbors we are trying to find.

References


Santini, G., 1984. Le méthode de fusion sur référentiel factoriel. IREP.

