Information Sciences 259 (2014) 596–610

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Incomplete-case nearest neighbor imputation in software measurement data ☆

Jason Van Hulse, Taghi M. Khoshgoftaar *

Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431, USA

ARTICLE INFO

Article history:
Available online 9 January 2011

Keywords:
Nearest neighbor imputation
Complete-case
Incomplete-case
Software measurement data

ABSTRACT

k nearest neighbor imputation (kNNI) is one of the most popular methods in empirical software engineering for imputing missing values. kNNI typically uses only complete cases as possible donors for imputation (called complete case kNNI or CCkNNI). Though it often produces reasonable results, CCkNNI is severely limited when the amount of missing data is large (and hence the number of complete cases is small). In response, a variant of CCkNNI called incomplete case *k* nearest neighbor imputation (ICkNNI) has been proposed as an attractive alternative. This work presents a detailed simulation comparing CCkNNI and ICkNNI using two different software measurement datasets. The empirical results show that using incomplete cases often increases the effectiveness of nearest neighbor imputation (especially at higher missingness levels), regardless of the type of missingness (i.e., the distribution of missing values in the data).

1. Introduction

The occurrence of missing values is a pervasive problem in software measurement datasets. Unfortunately, it is often impossible to obtain the actual values, and hence the data analyst must contend with the reality that some data is unavailable. To complicate matters further, many algorithms used to analyze measurement data require complete (i.e., non-missing) data in order to execute. For example, suppose the objective is to construct a regression model using past project data which predicts the number of program faults given a set of software measurements. In general, regression techniques are unable to directly deal with missing values, and hence it is necessary to handle the missing data before the regression model can be built. One common solution used by many empirical software engineering (SE) practitioners is called listwise deletion (LD). LD eliminates instances (program modules) with missing values from the data before analysis, resulting in a smaller, complete dataset. Though it is simple to implement, LD can cause significant problems, as mentioned both by researchers in empirical SE [4,11] and in general statistical literature [1]. Further, as SE datasets can be small in size, the information loss caused by LD can be intolerable.

Numerous *imputation* procedures, which fill-in or impute the missing values with one (or more) alternative values, have been proposed in an attempt to alleviate the problems caused by missing data. Imputation techniques have the advantage that no data is discarded, and that normal complete-case methods and algorithms can be used once the imputation is complete. One of the most popular imputation methods in empirical SE is *k* nearest neighbor imputation or kNNI. kNNI is often used because it is simple to implement and often provides good imputation performance. kNNI requires a case library from

☆ This is an expanded version of the work accepted and presented at the 2007 IEEE International Conference on Information Reuse and Integration (IRI) [19].

* Corresponding author. Tel.: +1 561 297 3994; fax: +1 561 297 2800.
E-mail addresses: jvanhulse@gmail.com (J. Van Hulse), khoshgof@fau.edu (T.M. Khoshgoftaar).

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2010.12.017

which the imputations are drawn, and the case library typically consists of only complete instances (we abbreviate this version of kNNI as CCkNNI). This restriction can be a severe problem, however, especially when the amount of missing data is high and there may be few (if any) complete instances. An alternate version of CCkNNI (called incomplete case kNNI or ICkNNI [4]) relaxes the restriction that all examples in the case library be complete, allowing some incomplete cases to also act as donors. The potential drawback of ICkNNI is that it is somewhat more complex and software may not be readily available.

The contribution of this work is to provide a detailed experimental comparison of CCkNNI and ICkNNI in the context of imputing missing software measurement data under various missingness scenarios. To our knowledge, ICkNNI has been the subject of only preliminary experimentation [4]. To achieve our objectives, we implemented software tools in the SAS programming language [12] to execute both CCkNNI and ICkNNI. Detailed simulations using two real-world software measurement datasets called JM1-2445 and CCCS were performed, as described in detail in Section 4. Statistical analysis of the results is provided, and they demonstrate that in almost all cases, ICkNNI is better or significantly better than CCkNNI, especially at higher levels of missingness. We therefore conclude that ICkNNI is an attractive alternative to CCkNNI for imputing missing measurement data, especially when the level of missingness is high.

The remainder of this work is organized as follows. Section 2 provides an overview of the CCkNNI algorithm, and ICkNNI is described in Section 3. The experimental design is presented in Section 4, and related work in the domain of missing values in SE datasets is presented in Section 5. Section 6 presents the experimental results. Conclusions and directions for future work are provided in Section 7. The appendices (Appendices A and B) provide detailed information on each of the datasets used in the experiments and the process of injecting missing values in those datasets.

2. Complete-case nearest neighbor algorithm

The algorithm for complete-case k nearest neighbor imputation (CCkNNI) is provided in Fig. 1. The case library of possible nearest neighbors for each example $\mathbf{x}_i$ to be imputed is the set of complete examples $\mathcal{C}$. The distance between instance $\mathbf{x}_i \in \mathcal{M}$ with missing values and each complete example $\mathbf{x}_j \in \mathcal{C}$ is computed, and the set of k nearest neighbors $\mathcal{K}_i$ is found (line 3). In other words, $\mathcal{K}_i$ is the set of k complete examples that are closest, as measured by some distance measure, to the example $\mathbf{x}_i$. Each missing attribute value in $\mathbf{x}_i$ is imputed with the average value of the k nearest neighbors (line 4); the imputed attribute values for instance $\mathbf{x}_i$ are denoted $\hat{x}_{im_l}$, $l = 1, \ldots, \alpha(i)$. The finally imputed example is denoted $\hat{\mathbf{x}}_i$ on line 5, where each missing attribute value is replaced by the imputed values. The procedure returns an imputed dataset. Since our dataset only contained numeric attributes, and for simplicity, we assume numeric attributes for imputation, though only simple modifications are needed to handle other data types. There are also a number of different versions of kNN imputation, which employ various attribute standardization algorithms. Simple modifications can be made to Fig. 1 to accommodate these scenarios, and the intention of this study is not to analyze all of the different versions of complete-case kNNI.

3. Incomplete-case nearest neighbor algorithm

The algorithm for incomplete-case k nearest neighbor imputation (ICkNNI) is provided in Fig. 2. The main difference between ICkNNI and CCkNNI is that the set of possible nearest neighbors for an example $\mathbf{x}_i$ being imputed *changes depending on the attribute value being imputed*. In particular, the requirement that all examples in the case-library be complete is relaxed. Instead, if attribute x_j of instance $\mathbf{x}_i$ is being imputed, then eligible nearest neighbors are those examples $\mathbf{x}_l$ which have the same subset of observed attributes as $\mathbf{x}_i$, and x_{lj} is also observed (line 3). The set of *potential* nearest neighbors when imputing

Procedure: Complete-Case kNN imputation

input: Dataset $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^{n}$, where $\mathbf{x}_i = (x_{im_1}, \ldots, x_{im_{\alpha(i)}}, x_{io_1}, \ldots, x_{io_{\beta(i)}})$ has $\alpha(i)$ missing attribute values $x_{im_1}, \ldots, x_{im_{\alpha(i)}}$ and $\beta(i)$ observed values $x_{io_1}, \ldots, x_{io_{\beta(i)}}$. Further, $\mathcal{D} = \mathcal{C} \cup \mathcal{M}$, where $\mathcal{C}$ is the set of complete examples ($\mathcal{C} = \{\mathbf{x}_j \in \mathcal{D} \mid \alpha(j) = 0\}$) and $\mathcal{M} = \mathcal{D} \setminus \mathcal{C}$; number of nearest neighbors k. The w attributes are denoted $x_1, \ldots, x_w$, and we assume $w = \alpha(i) + \beta(i)\ \forall\ \mathbf{x}_i \in \mathcal{D}$.

output: Dataset $\widehat{\mathcal{D}} = \mathcal{C} \cup \widehat{\mathcal{M}}$ where $\widehat{\mathcal{M}}$ is the set of imputed examples.

1. do $\forall\ \mathbf{x}_i \in \mathcal{M}$:
2. $\quad \forall\ \mathbf{x}_j \in \mathcal{C}$, find $d_j = d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{l=1}^{\beta(i)} (x_{io_l} - x_{jo_t})^2}$ where $o_t = o_l$.
3. $\quad$ Let $\mathcal{K}_i = \{\mathbf{x}_j \in \mathcal{C} \mid d_j \leq d_q\ \forall\ \mathbf{x}_q \notin \mathcal{K}_i\}$; $k = \mid \mathcal{K}_i \mid$.
4. $\quad$ For $l = 1, \ldots, \alpha(i)$, $\hat{x}_{im_l} = k^{-1} \sum_{\mathbf{x}_p \in \mathcal{K}_i} x_{po_t}$; $o_t = m_l$.
5. $\quad \hat{\mathbf{x}}_i = (\hat{x}_{im_1}, \ldots, \hat{x}_{im_{\alpha(i)}}, x_{io_1}, \ldots, x_{io_{\beta(i)}})$.
6. end do
7. $\widehat{\mathcal{M}} = \{\hat{\mathbf{x}}_i \mid \mathbf{x}_i \in \mathcal{M}\}$, $\widehat{\mathcal{D}} = \mathcal{C} \cup \widehat{\mathcal{M}}$.
8. Return $\widehat{\mathcal{D}}$.

Fig. 1. Complete-case kNN imputation.

Procedure: Incomplete-Case kNN imputation

1. do $\forall\, \mathbf{x}_i \in \mathcal{M}$:
2. do $\forall\, x_{im_j}, j = 1, \ldots, \alpha(i)$:
3. Let $O_l = \{x_{o_1}, \ldots, x_{o_{\beta(l)}}\}$. Let $T_{i,m_j} = \{\mathbf{x}_l \in \mathcal{D} \mid \{O_i \cup x_{m_j}\} \subset O_l\}$.
4. $\forall\, \mathbf{x}_l \in T_{i,m_j}$, find $d_l = d(\mathbf{x}_i, \mathbf{x}_l) = \sqrt{\sum_{p=1}^{\beta(i)} \left(x_{io_p} - x_{lo_t}\right)^2}$; $o_t = o_p$
5. Let $\mathcal{K}_{i,m_j} = \{\mathbf{x}_j \in \mathcal{C} \mid d_j \leq d_q\ \forall\, \mathbf{x}_q \notin \mathcal{K}_{i,m_j}\}$; $k = \mid \mathcal{K}_{i,m_j} \mid$.
6. $\hat{x}_{im_j} = k^{-1} \sum_{\mathbf{x}_p \in \mathcal{K}_{i,m_j}} x_{po_t}$; $o_t = m_j$.
7. end do
8. $\hat{\mathbf{x}}_i = (\hat{x}_{im_1}, \ldots, \hat{x}_{im_{\alpha(i)}}, x_{io_1}, \ldots, x_{io_{\beta(i)}})$.
9. end do
10. $\widehat{\mathcal{M}} = \{\hat{\mathbf{x}}_i \mid \mathbf{x}_i \in \mathcal{M}\}$, $\widehat{\mathcal{D}} = \mathcal{C} \cup \widehat{\mathcal{M}}$.
11. Return $\widehat{\mathcal{D}}$.

Fig. 2. Incomplete-case kNN imputation.

Table 1
Sample dataset.

ID	Attr 1	Attr 2	Attr 3	Attr 4	Attr 5
1	4	.	9	.	.
2	1	8	5	.	1
3	2	5	4	8	2
4	3	.	6	6	1
5	4	7	7	.	2
6	6	8	.	1	3

missing attribute $\mathbf{x}_i$ is denoted T_{i,m_j}. For a given example $\mathbf{x}_i$, when imputing each missing attribute, the set of k nearest-neighbors (denoted $\mathcal{K}_{i,m_j}$ since it depends on both the instance and the attribute being imputed) varies for each missing attribute: in general, $\mathcal{K}_{i,m_j} \neq \mathcal{K}_{i,m_p}$ for $j \neq p$. Once the set of nearest neighbors for imputing an attribute value are determined, the remainder of the ICkNNI imputation process is identical to that of CCkNNI. In particular, many of the modifications of CCkNNI such as attribute standardization can also be applied to ICkNNI.

Consider the sample dataset in Table 1, which has six instances, five numeric attributes, and an ID. Missing values are denoted with a '.'. Suppose instance 1, attribute 2 (x_{12}) is being imputed. Using CCkNNI, the only possible nearest neighbor of instance 1 is instance 3, because that is the only complete case. Therefore, $\hat{x}_{12} = 5$ using CCkNNI with $k = 1$. Using ICkNNI, however, the case library $T_{1,2} = \{\mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_5\}$. $\mathbf{x}_4$ is not in $T_{1,2}$ because $x_{42} =$., and $\mathbf{x}_6$ is not in $T_{1,2}$ because $x_{63} =$. but x_{13} is not missing. Continuing with the example of imputing x_{12}, $d(\mathbf{x}_1, \mathbf{x}_2) = 5$, $d(\mathbf{x}_1, \mathbf{x}_3) = 5.39$, and $d(\mathbf{x}_1, \mathbf{x}_5) = 2$, and hence $\mathcal{K}_{1,2} = \{\mathbf{x}_5\}$, and therefore $\hat{x}_{12} = x_{52} = 7$. As another example of the ICkNNI procedure, when imputing the fifth attribute (third missing attribute) of instance $\mathbf{x}_1$, $T_{1,5} = \{\mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}$. $\mathbf{x}_5$ is the nearest neighbor and with $k = 1, \hat{x}_{15} = x_{55} = 2$. With $k = 2, \mathcal{K}_{1,5} = \{\mathbf{x}_4, \mathbf{x}_5\}$, and $\hat{x}_{15} = (x_{45} + x_{55})/2 = 1.5$.

4. Experimental design

This section presents the details of our experimental design. Two different real-world software measurement datasets called JM1-2445 and CCCS were utilized in our experiments. Three missingness mechanisms (MCAR, MAR, and NI – explained in Section 4.1) were used, with missingness injected in 5%, 10%, 20%, 30% of the instances (and 40% and 50% missingness for MCAR).[1] The random selection process was repeated 30 times for the CCCS dataset and 10 times for JM1-2445 (CCCS is a much smaller dataset than JM1-2445, and hence more repetitions were needed to obtain approximately the same level of statistical confidence in the results). The characteristics of the two datasets are significantly different, where the attributes in CCCS are closer to a normal distribution than the attributes of JM1-2445, which are often skewed and far from normal. More details on the JM1-2445 and CCCS datasets, and further information on the injection of missing values, is presented in the Appendix (Appendices A and B).

4.1. Missingness mechanism

Three different missingness mechanisms are commonly used [10]: MCAR (missing completely at random), MAR (missing at random), and NI (non-ignorable). If the input dataset $\mathfrak{D}$ contains n instances and m attributes, let R be an $n \times m$ matrix

[1] The reasons for not using 40% and 50% missingness for MAR and NI missingness, which is related to constraints in the dataset size, are discussed in Appendices A and B.

such that the entry $r_{ij} = 1$ if the jth attribute of instance i is missing, and 0 otherwise. In other words, R identifies the location of the missing values in $\mathfrak{D}$. Let $\mathfrak{D}_{obs}$ denote the observed portion of $\mathfrak{D}$ and $\mathfrak{D}_{mis}$ the missing portion of $\mathfrak{D}$. MCAR missingness occurs if $P(R|\mathfrak{D}_{obs}, \mathfrak{D}_{mis}) = P(R)$. Missingness is MAR if $P(R|\mathfrak{D}_{obs}, \mathfrak{D}_{mis}) = P(R|\mathfrak{D}_{obs})$. In other words, missing data is MAR if the occurrence of missing data depends only on the observed values and not on the missing values. Finally, if the missingness pattern is related to the missing values themselves, then the missingness is called non-ignorable or NI.

4.2. Performance measure

The imputation performance of CCkNNI and ICkNNI was evaluated using the absolute error, i.e., comparing the imputed value for each missing attribute to the actual value (which is known since the original datasets are complete). For each missing attribute value x_{ij}, the imputed value calculated by technique T is $\hat{x}_{ij}^T$. Since the original values for all of the attributes in both datasets are non-negative integers, the imputed values were also rounded to the nearest non-negative integer. The absolute (imputation) error $ae^T(x_{ij})$ for missing attribute value x_{ij} imputed by technique T is calculated as:

$$ae^T(x_{ij}) = |\hat{x}_{ij}^T - x_{ij}|. \tag{1}$$

More specifically, for all instances with a missing value for x_j, imputation technique T has calculated an imputed value denoted by $\hat{x}_{ij}^T$. $ae^T(x_{ij})$ measures how close the imputed and original values are. Note that the absolute errors were calculated separately for each attribute, and in Section 6 the mean ae^T (aae^T) of imputation technique T for each attribute is presented separately:

$$aae^T(x_j) = N^{-1} \sum_{\mathbf{x}_i \text{ with } x_{ij} \text{ missing}} ae^T(x_{ij}). \tag{2}$$

where N is the number of $\mathbf{x}_i$ with missing values for attribute x_j. We choose to use the absolute error in this study because it is commonly used and is reasonable for the experimental datasets. We have also analyzed the results using the relative error, and have reached similar empirical conclusions. Therefore, we omit these results.

When performing kNN imputation, all attributes were normalized according to the following transformation:

$$\forall i,\ j,\ x_{ij} \mapsto \frac{x_{ij} - \min_i x_{ij}}{\max_i x_{ij} - \min_i x_{ij}}. \tag{3}$$

Other variable transformations can also be used, but as noted in Strike et al. [15], the differences may not be significant. Normalization was determined to be appropriate for these datasets, and further, it was not the intent of this study to examine different variable transformations.

5. Related work

Nearest-neighbor imputation has been the subject of much study in the field of empirical software engineering, though most studies concentrate on software effort prediction tasks. To our knowledge, with the exception of Jönsonn and Wohlin [4], none of these works utilize ICkNNI, which our results show severely limits the effectiveness of nearest-neighbor imputation at higher levels of missingness.

Jönsonn and Wohlin [4] reported an evaluation of k nearest neighbor (kNN) imputation using Likert data in a software engineering context. However, their study considered a very small dataset (54 instances), the data type was very specialized (ordinal-type Likert data), and only MCAR missingness was simulated. They further mention that ICkNNI may be dangerous when the data are not MCAR, though no evaluation was performed (our results contradict this statement). Our work greatly expands on Jönsonn and Wohlin [4] by utilizing two different real-world software measurement datasets, MCAR, MAR, and NI missingness (they only consider MCAR), and statistical analysis of the results. Tamura et al. [16] compare five methods for handling missing data, including kNN. Wasito and Mirkin [21] also analyze kNN imputation.

Myrtveit et al. [11] presented an empirical evaluation of imputation and likelihood-based methods for software cost modeling. Their study utilized listwise deletion (LD), mean imputation (MEI), similar response pattern imputation (SRPI), and full information maximum likelihood (FIML). Their results suggested that FIML was the only appropriate technique when missing data is not MCAR, although the authors state that if the objective is to apply complete-case statistical techniques other than regression, FIML might not be an option [11]. The other imputation techniques generated biased results if the missing data was not strictly MCAR, and in particular, the authors recommend that LD should not be used if the data is not MCAR (this result is supported by many other studies, e.g., [1,10]).

Twala et al. [18] investigated an ensemble strategy in the context of software development effort prediction from incomplete data. They proposed an ensemble comprised of Bayesian multiple imputation [10,13] (BMI) and (complete case) kNNI. The imputation procedures were evaluated using a regression model constructed from the imputed data, and the results show that using both BMI and kNNI in combination resulted in a better performance than using either imputation procedure alone. Twala [17] investigates the effects of missing data on decision tree learners.

Song et al. [14] compared class mean imputation (CMI) and (complete case) kNNI under different missingness mechanisms using a software project effort prediction dataset. Starting with two small datasets (50 and 100 instances,

respectively), missing values were injected into one or two attributes using either MCAR or MAR mechanisms. kNNI and CMI were used to impute the missing values, and mean magnitude relative error (MMRE) was used to compare the imputation performance. CMI was preferred for imputing missing values in small datasets over kNNI due to its higher accuracy. The impact of the missingness mechanism on the accuracy of imputation methods was determined not to be statistically significant for both CMI and kNNI.

Strike et al. [15] evaluated software cost estimation models constructed with incomplete data using LD, MEI, and eight different types of (complete case) hot-deck imputation (kNNI). They conclude that although deleting observations with missing values does not result in a substantial decrease in performance, better results can be obtained by applying imputation techniques. kNN hot-deck imputation obtained the best performance, followed by MEI.

Khoshgoftaar et al. [5] considered the effects of data noise on the effectiveness of five imputation techniques in the context of software quality imputation. Van Hulse et al. [20] compared three imputation techniques, Bayesian multiple imputation (BMI), kNNI, and regression imputation, and found that BMI was the best overall imputation procedure. Khoshgoftaar and Van Hulse [8] compare BMI, kNNI and MEI for handling missing values in software measurement data. None of these works considered ICkNNI.

Cartwright et al. [2] considered two data imputation techniques, MEI and (complete case) kNNI, for handling missing data in software project estimation datasets. The results demonstrate that MEI and kNNI (with $k = 2$) outperformed the regression model constructed without imputation, with kNNI giving the best results. The authors further conclude that "We suggest that it is possible to improve on the strategy of removing incomplete data either through case deletion (more common) or feature deletion" [2].

6. Results

For the JM1-2445 datasets, the results for MCAR missingness are provided in Fig. 3, with MAR missingness in Fig. 4, and NI missingness in Fig. 5. The *x*-axis is the percentage of missing data and the *y*-axis is the *aae*. For each of the three figures, six subfigures show the results for each of the six attributes which were injected with missing values (bcnt, tloc, bloc, uoper, uopan, and ccomp). The *aae* for ICkNNI is shown using the line with the boxes, while the line for CCkNNI is denoted by 'X'. If the *aae* for ICkNNI was significantly lower than that of CCkNNI at a specific missingness percentage, using a *t*-test with 5% significance level, then the box for ICkNNI is filled-in. We evaluated numerous values of *k* from one to 15, and found that $k = 5$ often generated the best results for both techniques when imputing the JM1-2445 dataset, and hence all figures use $k = 5$ for both ICkNNI and CCkNNI. The empirical conclusions are similar or identical given different values of *k* (and further, it was not the intention of this study to evaluate the impact of varying *k* – we leave this to future work).

In almost all cases, ICkNNI performs better or significantly better than CCkNNI. Regardless of the missingness mechanism, ICkNNI and CCkNNI perform similarly at 5% and 10% missingness, and for MCAR at 20% and 30% missingness, ICkNNI outperforms CCkNNI (but not by a statistically significant margin). With MCAR at 40%, and for MAR and NI at 20% and

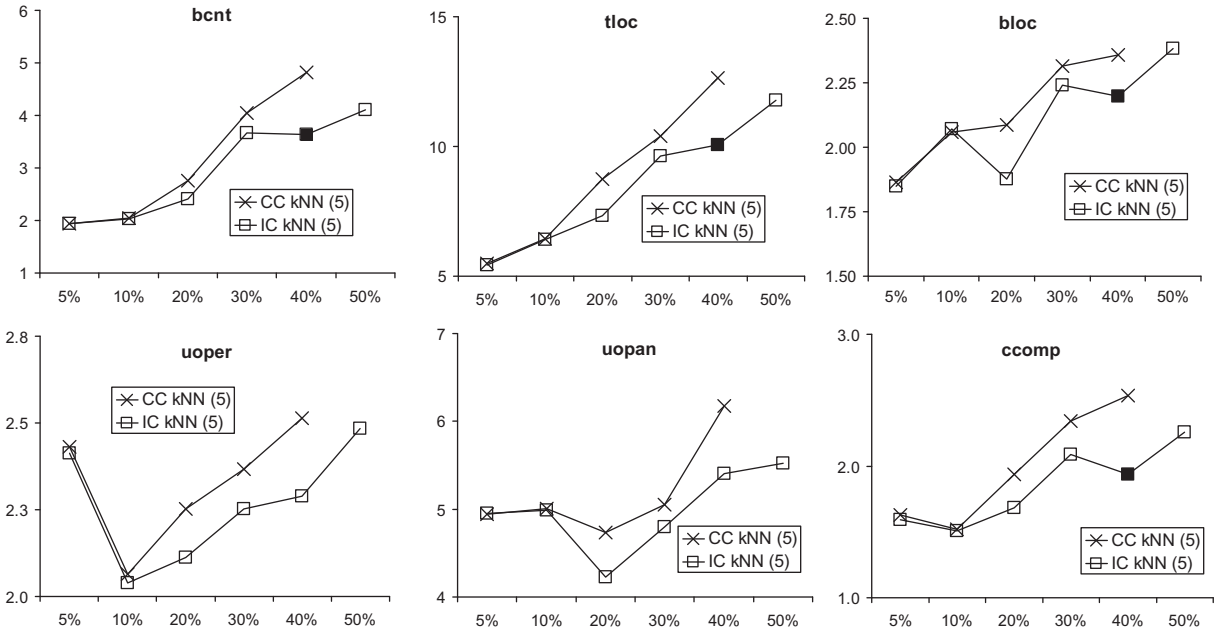

Fig. 3. Imputation comparison, JM1-2445, MCAR.

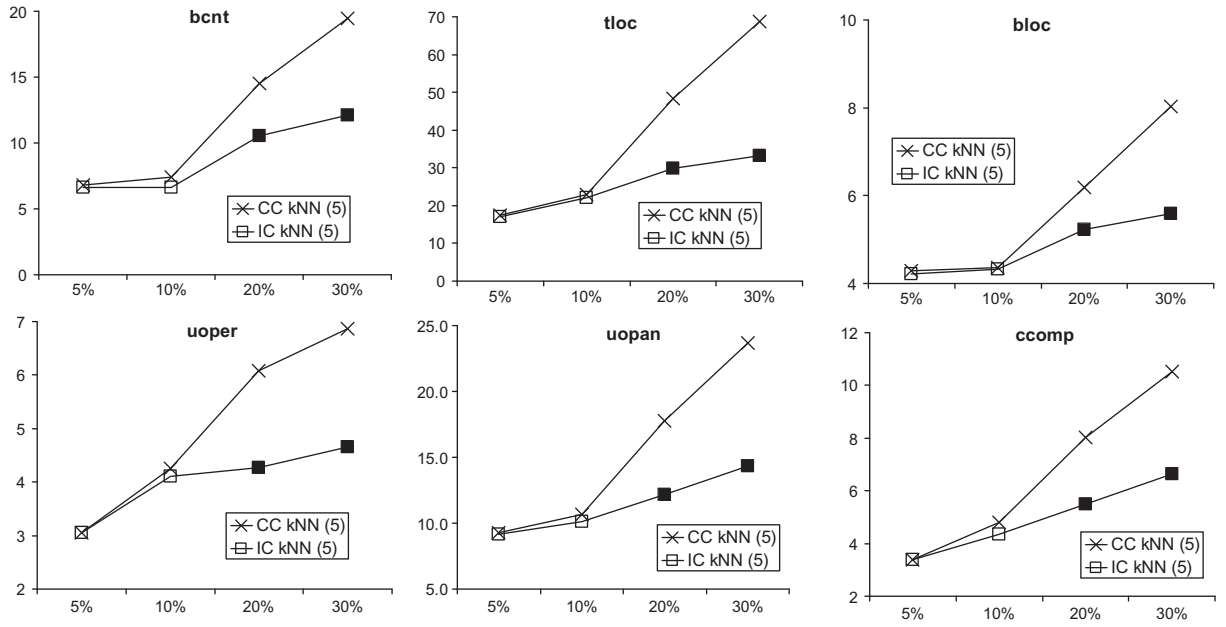

Fig. 4. Imputation comparison, JM1-2445, MAR.

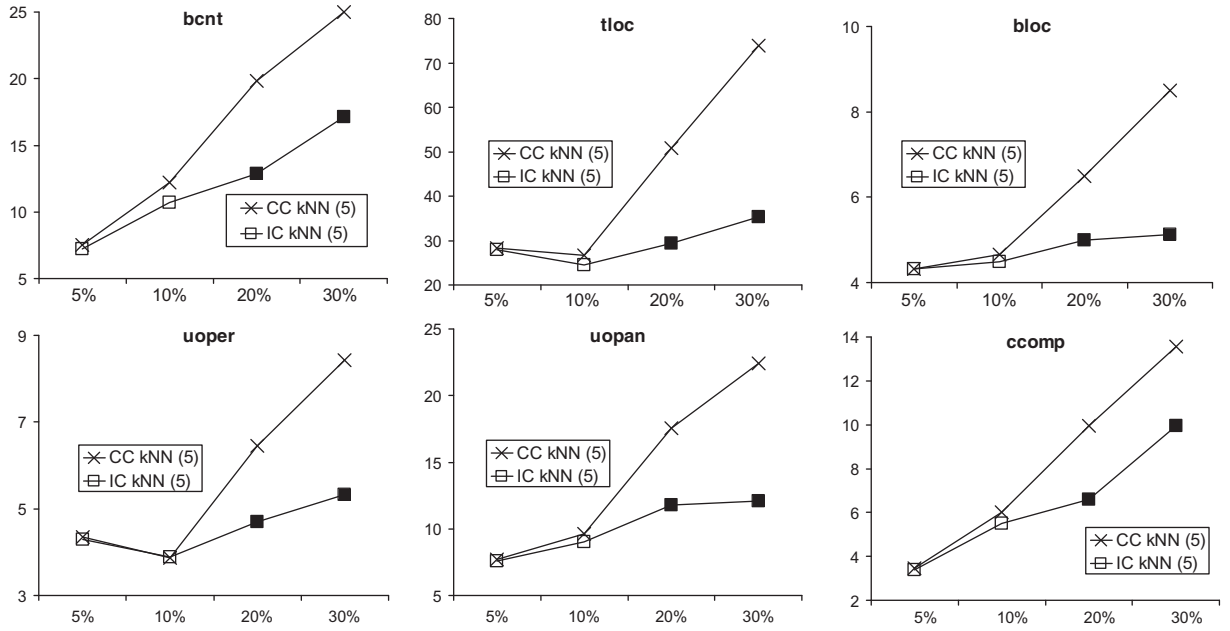

Fig. 5. Imputation comparison, JM1-2445, NI.

30%, ICkNNI vastly outperforms CCkNNI, and in the MAR and NI cases, is significantly better for all six attributes. For MCAR, ICkNNI is significantly better for four of the six attributes at 40% missingness. Further, ICkNNI never performed significantly worse than CCkNNI in any of our experiments. Note that 50% MCAR missingness resulted in very few complete cases. Therefore, CCkNNI could not be executed with all of the particular values of k used in this study, and hence is excluded.

The results for the CCCS dataset with $k = 3$ are presented in Figs. 6–8. For CCCS, $k = 3$ often generated the best results for both techniques, and the results with alternate values of k are similar. There were often no complete cases with 40% or 50% missingness with MCAR, and therefore CCkNNI could not be executed in these situations. For MAR and NI missingness at 30%, the results have been omitted for CCkNNI to make the figures easier to read, as the same trends observed at lower missingness levels continue at 30% missingness.

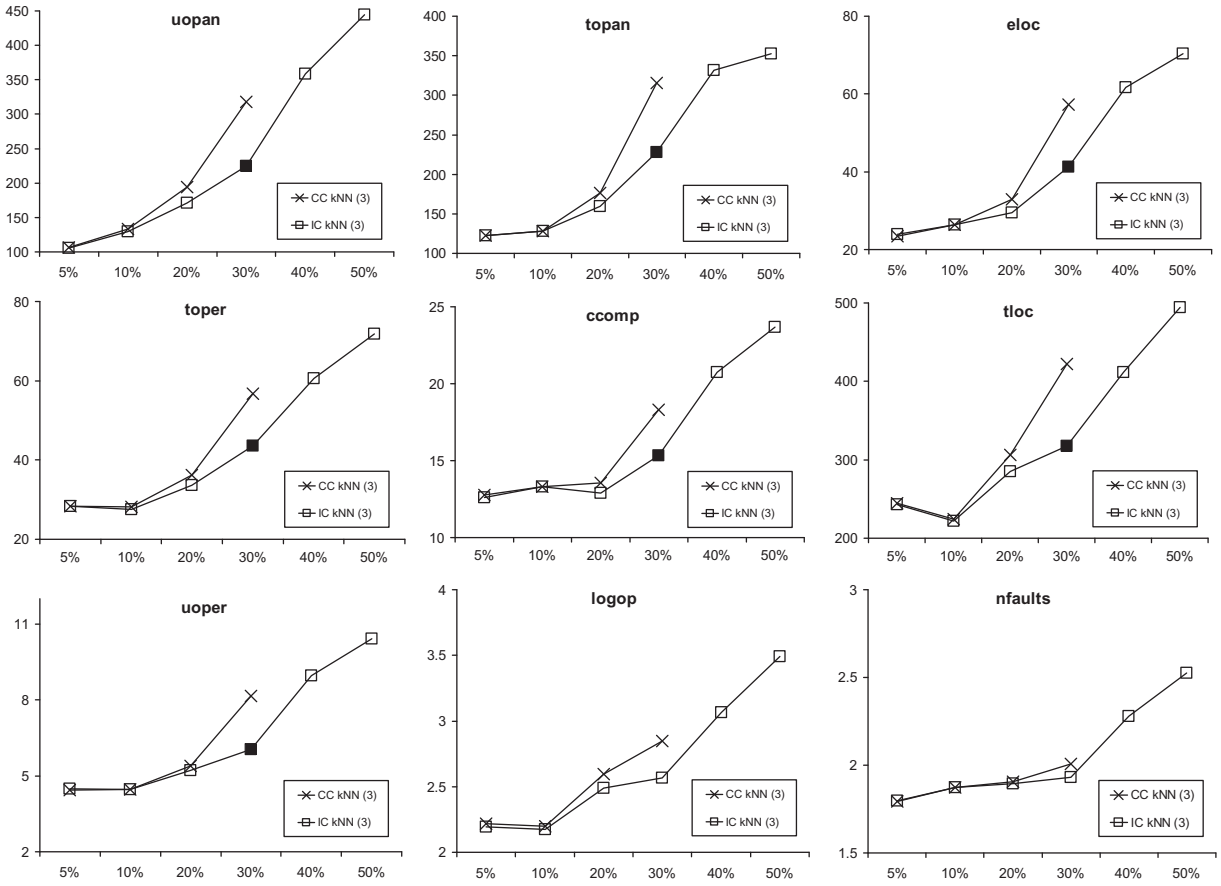

Fig. 6. Imputation comparison, CCCS, MCAR.

For MCAR missingness, the *aae* of ICkNNI and CCkNNI for MCAR at 5%, 10%, and in some cases 20%, is very similar. For seven of nine attributes with 30% missingness, however, ICkNNI is significantly better than CCkNNI. At 40% and 50% missingness, there were not enough complete cases for CCkNNI to execute.

For MAR and NI missingness, ICkNNI is better or significantly better than CCkNNI for all attributes with 10% or more missing values. At 20% missingness with MAR, ICkNNI was significantly better than CCkNNI for five of the eight attributes. For NI missingness, ICkNNI was significantly better than CCkNNI a total of ten times (twice at 10% missingness and eight times at 20% missingness).

7. Conclusions

As can be seen from the works discussed in Section 5, CCkNNI is an extremely popular imputation procedure in empirical SE. This work evaluates an attractive alternative to complete-case nearest neighbor imputation called ICkNNI. Instead of being restricted to utilizing only complete instances, ICkNNI allows incomplete instances (i.e., instances where some of the attribute values are missing) to also be considered as possible donors. This expansion of the potential donor set is important especially as the amount of missingness increases (and hence the number of complete cases decreases) or when the missingness is moderate and the missing values are more heavily concentrated in certain examples (i.e., under MAR and NI missingness). Intuitively, CCkNNI improves upon MEI by restricting the set of donors to those that are most similar to the example being imputed. However, as level of missingness increases, and the number of complete cases decreases, the performance of CCkNNI converges to that of MEI. By restricting the donors to only complete cases, the nearest neighbors may not really be that close to the example being imputed – in other words, a local mean converges to a global mean, and hence the performance MEI and CCkNNI converges. ICkNNI, however, broadens the potential donor population, allowing the set of neighbors to in fact be closer to the instance being imputed.

Similar reasoning explains why ICkNNI outperforms CCkNNI at slightly lower missingness levels when the missing values are biased (MAR and NI) instead of randomly distributed throughout the feature space (MCAR). Under biased missingness, instances in the subspaces of the feature space which contain the missing values will have very large errors relative to

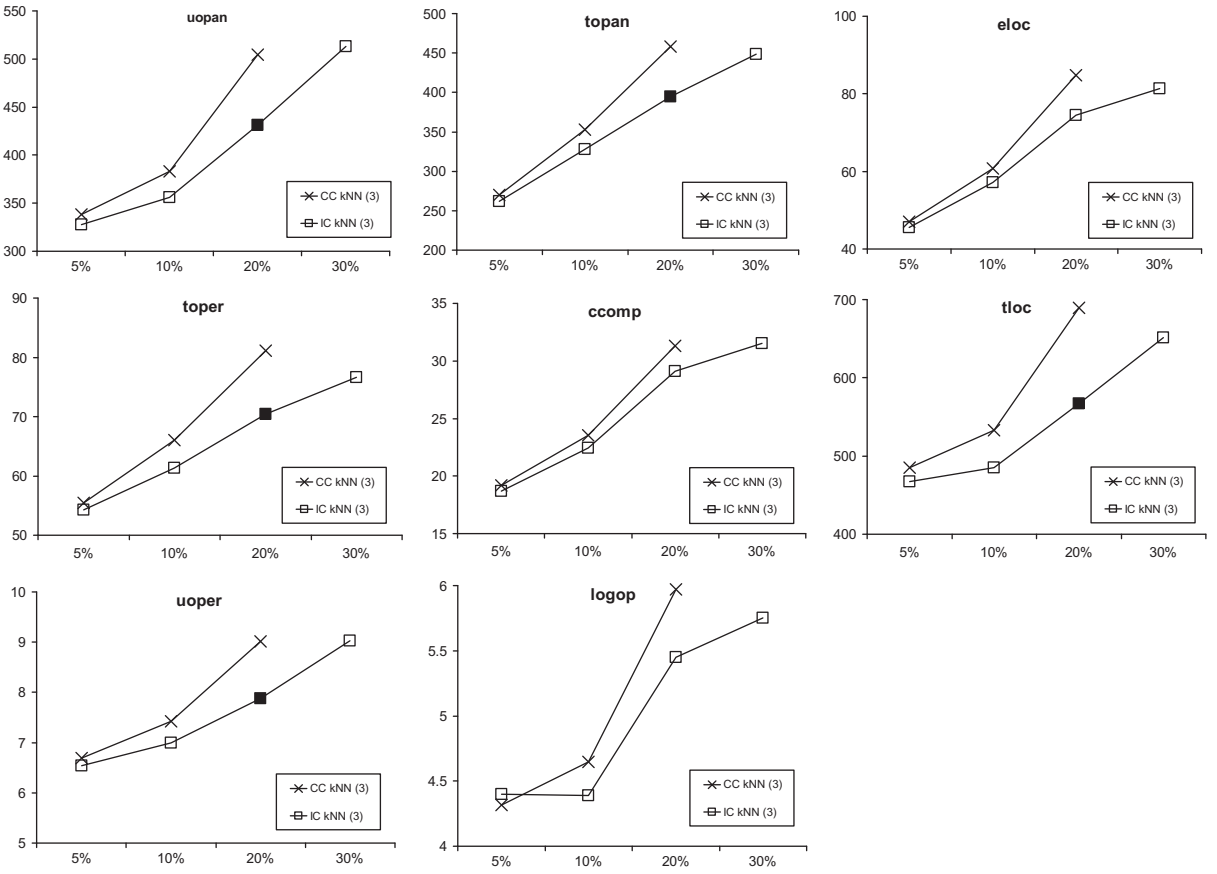

Fig. 7. Imputation comparison, CCCS, MAR.

instances in the remainder of the space. Allowing some incomplete cases to act as donors for those instances with missing values located in the subspaces with a high concentration of missingness improves performance by keeping the donors closer to the instance being imputed.

Despite these advantages, ICkNNI has received almost no attention in the domain of empirical software engineering. This study presents comprehensive experiments comparing ICkNNI and CCkNNI using two measurement datasets with simulated missingness. At higher levels of missingness, ICkNNI outperforms CCkNNI, in many cases significantly, regardless of the missingness mechanism. These empirical results strongly support the use of ICkNNI as an alternative to traditional nearest neighbor imputation, especially as missing values become more pervasive in the dataset. Future work should include a more detailed examination of the effect of varying k on the effectiveness of ICkNNI and CCkNNI. We also advocate additional experimentation on other datasets to further validate the utility of ICkNNI.

Appendix A. Missingness simulation in JM-2445 dataset

A.1. JM1-2445 dataset

The JM1-2445 dataset was derived from a NASA software project written in C called JM1 [9], which was obtained through the NASA Metrics Data Program (publically available at http://mdp.ivv.nasa.gov). Each instance (program module) in JM1-2445 represents a function or subroutine. Thirteen software metrics (attributes, independent variables or features) were measured at the function or subroutine level, as listed in Table 2 (JM1 originally had 21 metrics, but the eight derived Halstead metrics were not used in this study). In Table 2, LOC represents the lines of code in a program module. Additional information about the software metrics used in this study can be found in Fenton et al. [3]. The intent of this study is not to evaluate the utility of these specific software metrics, and other studies may choose to utilize additional or alternative metrics. The JM1-2445 dataset was also characterized by a dependent variable, Class, with values *nfp* (not fault prone) and *fp* (fault prone). An instance was labeled *fp* if it contained at least one fault and *nfp* otherwise. JM1 originally consisted of 10,883 program modules. JM1-2445, which contains 2,445 program modules, was derived from JM1 in a related study [22], the details of which we omit for space considerations and because it is not directly related to the objectives of this

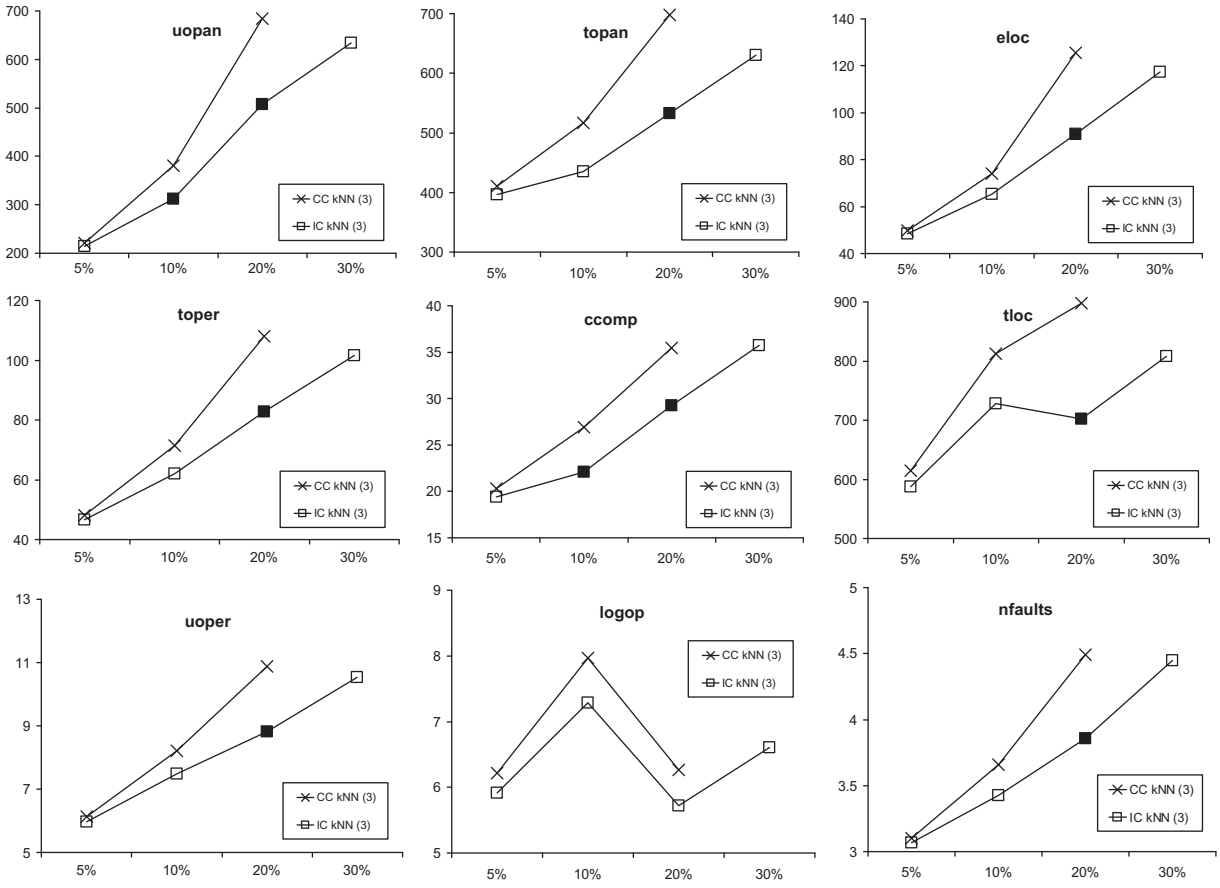

Fig. 8. Imputation comparison, CCCS, NI.

Table 2
Software metrics for JM1-2445.

Attribute	Description	Abbreviation
Branch	Branch Count	bcnt
Line Count	Executable LOC	eloc
	Comments LOC	cloc
	Blank LOC	bloc
	Code and comments LOC	ccloc
	Total Lines of Code	tloc
Basic Halstead	Total Operators	toper
	Total Operands	topan
	Unique Operands	uopan
	Unique Operators	uoper
McCabe's	Cyclomatic Complexity	ccomp
	Essential Complexity	ecomp
	Design Complexity	dcomp

study. The class distribution in the JM1-2445 dataset is 2,210 *nfp* instances and 235 *fp* instances. Table 3 provides some additional information on each of the software metrics in JM1-2445. For each of the 13 independent variables, the mean, standard deviation, and percentiles are provided.

A.2. Missing data injection

Since JM1-2445 did not contain missing values, it was necessary to inject missing data into the dataset. This section describes the injection process under each of the three missingness mechanisms MCAR, MAR, or NI. Recall that given a missingness level and mechanism, the random selection of instances was repeated ten times for JM1-2445 (called *runs*). The

Table 3
Attribute distribution for the JM1-2445 dataset.

Attribute	Mean	Std Dev	Percentiles								
			Max	95%	90%	75%	Median	25%	10%	5%	Min
bcnt	9.92	38.00	826	53	7	3	1	1	1	1	1
tloc	36.21	131.36	3442	200	17	12	9	6	4	4	1
eloc	27.11	107.86	2824	151	12	8	5	3	2	2	0
cloc	2.75	13.40	344	16	3	0	0	0	0	0	0
bloc	3.97	15.43	447	23	4	2	1	0	0	0	0
toper	70.79	258.46	5420	449	26	17	12	8	5	5	1
topan	47.17	166.76	3021	295	20	12	8	5	3	2	0
uoper	9.33	15.49	411	24	13	9	7	5	4	4	1
uopan	15.21	45.39	1026	74	15	9	6	4	3	2	0
ccomp	5.87	23.11	470	28	4	2	1	1	1	1	1
ecomp	2.80	8.81	140	11	3	1	1	1	1	1	1
dcomp	3.93	16.45	402	16	3	2	1	1	1	1	1
ccloc	0.29	1.85	38	1	0	0	0	0	0	0	0

Table 4
Distribution of missing values: JM1-2445, MCAR.

# Miss Attr	5%		10%		20%		30%		40%		50%	
	#	%	#	%	#	%	#	%	#	%	#	%
0	1788.5	73.10	1296.7	53.00	636.6	26.00	286.0	11.70	105.9	4.30	34.0	1.40
1	581.5	23.80	869.3	35.60	965.1	39.50	736.6	30.10	452.5	18.50	213.8	8.70
2	72.7	3.00	241.5	9.90	608.9	24.90	805.0	32.90	772.6	31.60	574.8	23.50
3	2.1	0.10	35.3	1.40	190.3	7.80	444.2	18.20	686.3	28.10	795.1	32.50
4	0.2	0.00	2.2	0.10	40.3	1.60	146.1	6.00	336.6	13.80	583.1	23.80
5	0.0	0.00	0.0	0.00	3.8	0.20	25.2	1.00	81.6	3.30	211.3	8.60
6	0.0	0.00	0.0	0.00	0.0	0.00	1.9	0.10	9.5	0.40	32.9	1.30

tables presented in this section represent an average over all *runs* for the given dataset. In each scenario, specific parameters were established to generate the missing data. These parameter settings were chosen to be reasonable for the datasets and the experiments done here; different parameter choices could have also been selected.

JM1-2445 contains 13 independent variables and a class label. When injecting missing values into JM1-2445, missingness was restricted to six of the 14 attributes. Therefore each instance had a value for at least eight of the 14 attributes. The attributes used for missing value injection were bcnt, tloc, bloc, uoper, uopan, and ccomp.[2]

A.2.1. JM1-2445 dataset – MCAR missingness

For the JM1-2445 dataset with MCAR missingness, attribute values were chosen completely randomly from among the six specified attributes for missing value injection. As there are 2445 instances in JM1-2445 and six attributes were considered, a grid of $2445 \times 6 = 14670$ cells (or attributes values) are available for missing value injection. For 5% missingness, for example, $14670 \times 5\% = 734$ cells were set to missing. Note that in the overall dataset, which has $2445 \times 14 = 34230$ cells, 5% missingness in the six chosen attributes is equivalent to 2.14% (=734/34230) missingness. We will refer to this level of missingness as 5% throughout this work, even though the percentage of missing data in the larger dataset is less than 5%. 10% missingness equates to 1476 cells or attribute values, while 20%, 30%, 40%, and 50% are 2934,4401, 5868, and 7335, respectively. Table 4 provides the distribution of the number of missing attributes for MCAR missingness. With 5% missingness, the dataset had on average 1788.5 complete examples (no missing values), 581.5 examples with one missing value, and 72.7 with two missing values. At 50% missingness, on the other hand, only 34 instances on average had no missing values, while over 66% of the examples had three or more missing attribute values.

A.2.2. JM1-2445 dataset – MAR missingness

MAR missingness with JM1-2445 was generated by making the distribution of missing values dependent on the value of the class, which was fully observed. 2210 instances or 90.39% of the dataset are class *nfp*, and 235 instances or 9.61% of the dataset are class *fp*. If simple random selection of attribute values were used, then it would be expected that approximately 9.61% of the missing values are from instances in class *fp*. MAR missingness was implemented using a biased selection

[2] The reason for choosing these six attributes is due to a related study [7], which shows these are the three cleanest and three noisiest attributes, in a relative sense, in the JM1-2445 dataset. We do not further discuss the added dimension of data quality and its relation to imputation here due to space limitations, but will consider this relationship in future work. Other attributes could have also been selected, and our experiments have shown that the empirical conclusions presented in this study do not significantly change.

process where 25% of the missingness was located in the class *fp* instances. Therefore, class *fp*, which is only 9.61% of the dataset, contains 25% of the missingness. 25% missingness in the *fp* modules was used because higher percentages (e.g., 50%) could not be accomodated with 30% overall missingness. In other words, there are some constraints among the parameters used, and we acknowledge that other parameters could have been chosen.

As mentioned previously, missingness in JM1-2445 was injected only in the grid of 14760 cells, comprised of 2445 instances with six attributes. Further, 2210 instances are class *nfp*, for a grid size of 13260 cells (=2210 $\times$ 6). With 235 *fp* instances, the grid size is 235 $\times$ 6 = 1410 cells. Consider the case of 5% missing value injection, which equates to 734 missing cells in the total grid of 14760 cells. Since our missing value injection algorithm requires 25% of the missingness to be located in the *fp* class, 183 of the 1410 cells from instances in this class will be randomly chosen. The remaining 551 cells for missing value injection will be chosen from the *nfp* class. Therefore, the effective missingness from the six attributes is 13.01% (=183/1410) for class *fp* and 4.16% (=551/13260) for class *nfp*. For 30% missingness, the effective information loss in the six chosen attributes is 78.03% for class *fp* and 24.89% for class *nfp*. These statistics demonstrate the relatively higher amount of information loss in class *fp* instances compared to class *nfp* instances. The distribution of the number of missing attribute values using MAR is provided in Table 5.

A.2.3. JM1-2445 dataset – NI missingness

The missing value injection algorithm used for non-ignorable missingness in JM1-2445 had missingness restricted to six chosen attributes, and the threshold t_A was chosen for each attribute A such that approximately 75% of the instances had a value of attribute A less than t_A. After determining the threshold values, 40% of the missingness in each attribute was injected into instances with a value for attribute A less than t_A, while 60% of the injected missingness was located in instances with a value for attribue A greater than t_A. As with MAR missingness, the parameters used for NI missingness were chosen based on data and parameter constraints, and because they are reasonable for the JM1-2445 dataset – other parameters can also be chosen. The information loss in attribute A was significantly greater for instances with a large value for A. Table 6 provides the distribution of the number of missing attributes for the non-ignorable scenario.

Appendix B. Missingness simulation in CCCS dataset

B.1. CCCS dataset

The CCCS dataset used in our experiments is a large military command, control and communications system written in Ada [6]. CCCS contains 282 instances (program modules), where each instance is an Ada package consisting of one or more procedures. CCCS contains eight software metrics which are used as independent variables or attributes. An additional attribute, *nfaults* (the dependent variable), indicates the number of faults attributed to a module during the system integration and test phases, and during the first year of deployment. Table 7 lists the software metrics in the CCCS dataset.

Table 5
Distribution of MAR missing values.

# Miss	5%		10%		20%		30%	
	#	%	#	%	#	%	#	%
0	1811	74.1	1323	54.1	735	30.1	383	15.7
1	543	22.2	838	34.3	918	37.5	807	33.0
2	83	3.4	230	9.4	498	20.4	663	27.1
3	8	0.3	46	1.9	189	7.7	304	12.4
4	0	0.0	6	0.3	74	3.0	137	5.6
5	0	0.0	1	0.0	26	1.1	97	4.0
6	0	0.0	0	0.0	4	0.2	54	2.2

Table 6
Distribution of NI missing values.

# Miss	5%		10%		20%		30%	
	#	%	#	%	#	%	#	%
0	1821	74.5	1376	56.3	823	33.7	533	21.8
1	525	21.5	756	30.9	831	34.0	725	29.6
2	88	3.6	241	9.9	448	18.3	521	21.3
3	10	0.4	60	2.5	207	8.5	291	11.9
4	0	0.0	11	0.4	98	4.0	178	7.3
5	0	0.0	1	0.0	33	1.4	131	5.4
6	0	0.0	0	0.0	5	0.2	66	2.7

Table 7
Software metrics for the CCCS dataset.

Independent variables	Abbreviation
Unique operators	uoper
Total operators	toper
Unique operands	uopan
Total operands	topan
Cyclomatic complexity	ccomp
Logical operators	logop
Total lines of code	tloc
Executable LOC	eloc
Dependent variable	**Abbreviation**
Number of faults	nfaults

Table 8
Attribute distribution for the CCCS dataset.

Attribute	Mean	Std Dev	Percentiles								
			Max	95%	90%	75%	Median	25%	10%	5%	Min
uoper	23.05	15.75	88	54	42	31	21	9	5	4	1
toper	107.61	143.72	1124	409	293	138	62.5	13	5	4	1
uopan	573.14	993.30	8606	2364	1480	634	188.5	71	19	15	4
topan	483.26	874.51	7736	1980	1308	527	164	45	10	6	1
ccomp	27.98	58.86	614	109	69	30	8	3	1	1	1
logop	2.87	7.33	58	20	9	2	0	0	0	0	0
tloc	785.84	1038.99	9163	2680	1775	899	405.5	240	147	118	19
eloc	103.74	163.75	1412	429	246	116	44.5	21	9	6	2
nfaults	2.37	5.08	42	12	7	2	0	0	0	0	0

Table 8 provides additional information on each of the software measurements in the CCCS dataset. In particular, for each of the nine attributes in CCCS, Table 8 provides the mean value, the standard deviation and nine different percentiles, including the maximum, minimum and median.

B.2. Missing data injection

B.2.1. CCCS dataset – MCAR missingness

For MCAR in the CCCS dataset, missing values were randomly injected into any of the nine attributes in the dataset with equal probability, subject to the constraint that each instance (program module) must have at least 2 non-missing attribute values. Missingness was injected at 5%, 10%, 20%, 30%, 40%, and 50% levels. In other words, the CCCS dataset contains a total of 2538 (=282 $\times$ 9) attribute values. 5% missingness corresponds to 5% $\times$ 2538 = 127 missing attribute values chosen at random from the dataset, subject to the constraint that each instance must have at least 2 non-missing attributes. Table 9 presents the distribution of the average number of instances with n missing values, $n = 0,\ldots,7$, for each of the missingness levels. For example, 10% missingness injected according to MCAR results in datasets with an average of 100.33 instances with no missing values, 119.87 instances with 1 missing value, and so on. The average is taken over the 30 different random selections, hence the reason why the count of instances (in the column titled '#') is not a whole number.

B.2.2. CCCS dataset – MAR missingness

MAR missingness was implemented in CCCS by injecting missing values into eight of the nine attributes (excluding the dependent variable *nfaults*) such that 75% of the missing values occurred for program modules with *nfaults* > 1, while 25% of

Table 9
Distribution of missing values: CCCS, MCAR.

# Miss Attr	5%		10%		20%		30%		40%		50%	
	#	%	#	%	#	%	#	%	#	%	#	%
0	171.13	60.70	100.33	35.60	29.67	10.50	8.47	3.00	1.20	0.40	0.10	0.00
1	95.47	33.90	119.87	42.50	86.80	30.80	38.60	13.70	12.33	4.40	2.10	0.70
2	14.67	5.20	51.97	18.40	95.83	34.00	78.63	27.90	42.10	14.90	13.77	4.90
3	0.73	0.30	9.13	3.20	52.00	18.40	85.23	30.20	76.40	27.10	44.30	15.70
4	0.00	0.00	0.70	0.20	15.17	5.40	49.43	17.50	81.80	29.00	77.47	27.50
5	0.00	0.00	0.00	0.00	2.33	0.80	18.30	6.50	49.60	17.60	83.47	29.60
6	0.00	0.00	0.00	0.00	0.20	0.10	3.13	1.10	15.90	5.60	46.33	16.40
7	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.10	2.67	0.90	14.47	5.10

the missingness occurred for program modules with *nfaults* = 0 or 1. Therefore, the missing values injected under MAR for CCCS were dependent only upon the response variable *nfaults*, which was fully observed. Note that 199 instances, or 70.57% of the dataset, have *nfaults* = 0 or 1, while only 83 instances or 29.43% of the dataset has *nfaults* > 1. Therefore with an unbiased random selection, approximately 70.57% of the missing would be from instances with *nfaults* = 0 or 1, while 29.43% would come from instances with *nfaults* > 1. Under the MAR missingness mechanism, however, our sampling scheme required that 75% of the missing values to be located in instances with *nfaults* > 1 and 25% to be located in instances with *nfaults* = 0 or 1. The parameters 75% and 25% were chosen because they place a significant emphasis on missing values in program modules with a large number of faults. Further, there is a trade-off between how much missingness can be injected (if 75% were increased to 90%, then 30% missingness could not be accomodated) and how different MAR missingness is from MCAR (i.e., lowering 75% makes the missingness more similar to MCAR). The constraint that no instance can have more than seven missing attributes was still utilized for MAR missingness. Since *nfaults* is fully observed, the distribution of missing values is partially dependent on a fully observed variable, satisfying the MAR condition. Given these parameters, it was not possible to generate 40% and 50% missingness. Different parameters could also be chosen at the descretion of the experimenter, however these were deemed reasonable for this dataset.

Table 10 presents the distribution of the number of missing attributes for the CCCS dataset under the MAR missingness mechanism at various missingness levels. At 5% missingness, there are on average (over 30 *runs*) 199.30 complete instances. The number of complete instances drops to 156.27 instances with 10% missingness, 104.70 with 20% missingness and 74.87 instances with 30% missingness. As the level of missing data increases, the percentage of instances with a large number of missing attributes also increases. At 20% missingness, almost 20% of the instances have four or more missing attributes, while at 30% missingness, almost 30% of the instances have four or more missing attributes.

Note that a significant amount of the missingness is confined to those instances with *nfaults* > 1, so with higher missingness levels the amount of information loss in the instances with more faults is very high. A complete description of these statistics cannot be presented for space considerations, but are summarized briefly here. At 30% missingness, there are no complete instances with *nfaults* > 1. Further, on average, 37.17 instances (which is 44.8% of the instances with *nfaults* > 1), have seven missing attributes, and 24.43 instances (30.60%) have six missing attributes. On the other hand, for instances with *nfaults* $\leqslant$ 1,74.87 instances or 37.6% are complete, while 85.27 instances have only one missing attribute value. This data demonstrates the significant bias in missing values towards instances with two or more faults.

B.2.3. CCCS dataset – NI missingness

Non-ignorable missingness was implemented in CCCS as follows. A threshold t_A was determined for each attribute A (including *nfaults*) such that the number of instances with values for attribute A less than t_A was approximately 75% of the dataset, or roughly 212 instances. Within each attribute, missingness was injected such that 40% of the missingness occurred in the 75% of the dataset with attribute $A \leqslant t_A$, while 60% of the missingness occurred in the instances with attribute $A > t_A$. This was done independently for each attribute in the dataset. In other words, 60% of the missing values in each attribute were confined to the largest quartile of the distribution of that attribute. Unlike MAR, the distribution of missingness for NI is dependent on the missing values themselves (because the original attribute values were used to determine the threshold t_A), making the missingness non-ignorable. As in the cases of MCAR and MAR, the constraint that no instance can have more than seven attributes missing was adhered to for NI. Different methodologies and parameter values could also have been used for NI missingness, but these were deemed reasonable for these experiments.

Table 11 shows the distribution of missing values within each of the nine attributes specifically for 30% missingness. The second column is the threshold t_A, chosen so that approximately 75% of the instances have a value for the attribute $\leqslant t_A$. The column '% miss' is the percentage of missing data in the attribute, which in this case is approximately 30% (each attribute has approximately the same level of missingness). The next three columns, with the heading 'Below Attribute Threshold', display the number of non-missing and missing values and the percentage of missing values below the threshold for that attribute. The next three columns present the same information for instances with a value for the attribute greater than the threshold. Finally the last column is the percentage of instances in the dataset with a value for the attribute greater than the threshold (typically around 25%). For attribute *uoper*, for example, the threshold was chosen to be 30. 26.24% of the instances in the

Table 10
Distribution of missing values: CCCS, MAR.

# Miss Attr	5%		10%		20%		30%	
	#	%	#	%	#	%	#	%
0	199.30	70.70	156.27	55.40	104.70	37.10	74.87	26.50
1	59.47	21.10	67.57	24.00	80.10	28.40	85.27	30.20
2	16.97	6.00	29.83	10.60	24.47	8.70	33.33	11.80
3	5.53	2.00	17.60	6.20	17.73	6.30	6.33	2.20
4	0.70	0.20	8.03	2.80	22.83	8.10	4.70	1.70
5	0.00	0.00	2.37	0.80	18.60	6.60	14.90	5.30
6	0.03	0.00	0.33	0.10	10.53	3.70	25.43	9.00
7	0.00	0.00	0.00	0.00	3.03	1.10	37.17	13.20

Table 11
Distribution of missing values by attribute: CCCS, NI, 30% missingness.

Attribute	Threshold	% miss	Below attribute threshold			Above attribute threshold			
			# non-miss	# miss	% miss	# non-miss	# miss	% miss	%
uoper	⩽30	30.25	170.6	37.4	17.96	26.1	47.9	64.77	26.24
toper	⩽137	29.00	177.5	33.5	15.88	22.7	48.3	67.98%	25.18
uopan	⩽632	30.57	173.4	37.6	17.80	22.4	48.6	68.50	25.18
topan	⩽510	29.68	175.1	35.9	17.01	23.2	47.8	67.32	25.18
ccomp	⩽29	30.15	174.0	36.0	17.16	23.0	49.0	68.06	25.53
logop	⩽1	30.09	175.0	36.0	17.08	22.2	48.8	68.78	25.18
tloc	⩽896	30.28	173.9	37.1	17.58	22.7	48.3	68.03	25.18
eloc	⩽115	29.52	177.1	33.9	16.07	21.7	49.3	69.48	25.18
nfaults	⩽2	30.32	179.1	36.9	17.08	17.4	48.6	73.64	23.40

Table 12
Distribution of missing values: CCCS, NI.

# Miss Attr	5%		10%		20%		30%	
	#	%	#	%	#	%	#	%
0	187.13	66.36	132.53	47.00	62.53	22.17	29.87	10.59
1	70.37	24.95	85.87	30.45	87.47	31.02	68.03	24.13
2	18.23	6.47	36.50	12.94	56.93	20.19	63.97	22.68
3	4.97	1.76	16.47	5.84	31.13	11.04	40.53	14.37
4	1.23	0.44	8.00	2.84	19.80	7.02	21.63	7.67
5	0.07	0.02	2.13	0.76	13.63	4.83	16.23	5.76
6	0.00	0.00	0.43	0.15	7.60	2.70	16.40	5.82
7	0.00	0.00	0.07	0.02	2.90	1.03	25.33	8.98

dataset have a value for *uoper* greater than 30. The percentage of missing data in instances with *uoper* $\leqslant$ 30 is 17.96% (=37.4/(170.6 + 37.4)). 74 instances have *uoper* > 30, and on average 47.9 have missing values, for a missing data percentage of 64.77%. In other words, *uoper* has suffered almost 65% information loss from the largest quartile of its distribution, while from the remaining 75%, the information loss is slightly less than 18%.

The set of instances larger than the threshold changes depending on the attribute used, however since some of the attributes in CCCS are correlated, there is often a significant amount of overlap. Of the 74 instances with *uoper* > 30, 59 also had *toper* > 137 and 60 had *uopan* > 632. As another example, 71 instances had *toper* > 137, and 66 of those also had *topan* > 510.

The distribution of the number of missing attributes given the NI missingness mechanism in the CCCS dataset is presented in Table 12. At 5% missingness, the dataset has, on average, about 187 complete instances. At 10% missingness, the percentage of complete instances drops to 47%. Finally at 30% missingness, there are on average less than 30 complete instances (approximately 10.59%). Further over 20% of the instances have at least five missing attributes. The injection algorithm employed for NI missingness has resulted in significant information loss at higher missingness levels, especially at the tail of the attribute distribution.

References

[1] P.D. Allison, Missing Data. 07-136. Sage University Papers Series on Quantitative Applications in the Social Sciences, Thousand Oaks, CA, 2000.
[2] M.H. Cartwright, M.J. Shepperd, Q. Song, Dealing with issuing software project data, in: 9th IEEE International Software Metrics Symposium, 2003, pp. 154–165.
[3] N.E. Fenton, S.L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, second ed., PWS Publishing Company, ITP, Boston, MA, 1997.
[4] P. Jönsson, C. Wohlin, An evaluation of *k*-nearest neighbour imputation using likert data, in: 10th IEEE International Symposium on Software Metrics (METRICS'04), 2004, pp. 108–118.
[5] T.M. Khoshgoftaar, A. Folleco, L. Bullard, J. Van Hulse, Software quality imputation in the presence of noisy data, in: Proceedings of the IEEE International Conference on Information Reuse and Integration (IRI 2006). Hawaii, 2006, pp. 484–489.
[6] T.M. Khoshgoftaar, N. Seliya, Comparative assessment of software quality classification techniques: an empirical case study, Empirical Software Engineering Journal 9 (2) (2004) 229–257.
[7] T.M. Khoshgoftaar, J. Van Hulse, Identifying noisy features with the pairwise attribute noise detection algorithm, Intelligent Data Analysis: An International Journal 9 (6) (2005) 589–602.
[8] T.M. Khoshgoftaar, J. Van Hulse, Imputation techniques for multivariate missingness in software measurement data, Software Quality Journal 16 (4) (2008) 563–600.
[9] T.M. Khoshgoftaar, S. Zhong, V. Joshi, Enhancing software quality estimation using ensemble-classifier based noise filtering, Intelligent Data Analysis: An International Journal 9 (1) (2005) 3–27.
[10] R.J.A. Little, D.B. Rubin, Statistical Analysis with Missing Data, second ed., John Wiley and Sons, Hoboken, NJ, 2002.
[11] I. Myrtveit, E. Stensrud, U. Olsson, Analyzing data sets with missing data: an empirical evaluation of imputation methods and likelihood-based methods, IEEE Transactions on Software Engineering 27 (11) (2001) 999–1013.
[12] SAS Institute, SAS/STAT User's Guide, SAS Institute Inc., 2004.
[13] J.L. Schafer, Analysis of Incomplete Multivariate Data, Chapman and Hall/CRC, 2000.

[14] Q. Song, M.J. Shepperd, M.H. Cartwright, A short note on safest default missingness mechanism assumptions, Empirical Software Engineering 10 (2) (2005) 235–243.
[15] K. Strike, K.E. Emam, N. Madhavji, Software cost estimation with incomplete data, IEEE Transactions on Software Engineering 27 (10) (2001) 890–908.
[16] K. Tamura, T. Kakimoto, K. Toda, M. Tsunoda, A. Monden, K. Matsumoto, Empirical evaluation of missing data techniques for effort estimation (2009). doi: 10.1.1.145.780 <http://citeseerx.ist.psu.edu/viewdoc/summary?>.
[17] B. Twala, An empirical comparison of techniques for handling incomplete data using decision trees, Applied Artificial Intelligence: An International Journal 23 (5) (2009) 373–405.
[18] B. Twala, M.H. Cartwright, Ensemble imputation methods for missing software engineering data, in: Proceedings of 11th IEEE International Software Metrics Symposium, 2005, pp. 30–40.
[19] J. Van Hulse, T.M. Khoshgoftaar, Incomplete-case nearest neighbor imputation in software measurement data, in: Proceedings of the IEEE International Conference on Information Reuse and Integration (IRI 2007), Las Vegas, NV, 2007, pp. 630–637.
[20] J. Van Hulse, T.M. Khoshgoftaar, C. Seiffert, A comparison of software fault imputation procedures, in: Proceedings of the IEEE International Conference on Machine Learning and Applications (ICMLA 2006), Orlando, FL, 2006, pp. 135–142.
[21] I. Wasito, B. Mirkin, Nearest neighbour approach in the least-squares data imputation algorithms, Information Sciences 169 (1–2) (2005) 1–25.
[22] S. Zhong, T.M. Khoshgoftaar, N. Seliya, Analyzing software measurement data with clustering techniques, IEEE Intelligent Systems (2004) 22–29.