A DECISION TREE-BASED APPROACH FOR MISSING VALUE IMPUTATION OF MIXED-TYPE DATA

by

Heizel M. Rosado Galindo

A Thesis Submitted In Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE
in
INDUSTRIAL ENGINEERING

University of Puerto Rico

Mayagüez Campus

May 2017

Approved by:

Saylisse Dávila, PhD President, Graduate Committee	Date
Wandaliz Torres, PhD Member, Graduate Committee	Date
Noel Artiles, PhD Member, Graduate Committee	Date
Viviana Cesaní, PhD Industrial Engineering Dept. Head	Date
Moraima De Hoyos, PhD Representative of Graduate Studies	Date

ABSTRACT

Researchers and practitioners of many areas of knowledge frequently struggle with missing data. Missing data is a problem because almost all standard statistical methods assume that the information is complete. Missing value imputation offers a solution to this problem. The main contribution of this work lies on the development of a random forest-based imputation method that can handle any type of data, including high-dimensional data with non-linear complex interactions. The premise behind the proposed scheme is that a variable can be imputed taking into account only those variables that are related to it using feature selection. This work compares the performance of the proposed scheme with other two imputation methods commonly used in literature: KNN and missForest. The results suggest that the proposed method can be useful in complex categorical scenarios with high volume of missing values. The proposed method is an approximation of missForest that significantly reduces the amount of variables used in the imputation.

RESUMEN

Investigadores de distintas áreas de conocimiento se enfrentan frecuentemente al problema de datos incompletos. Esto representa un obstáculo, pues la mayoría de los métodos estadísticos disponibles en la literatura asumen que la información está completa. La imputación de datos faltantes representa una opción viable a este problema. Este trabajo tiene como contribución principal el desarrollo de un método de imputación de datos vaciós basado en árboles de decisión, que pueda manejar cualquier tipo de datos, incluyendo datos de alta dimensionalidad con interacciones complejas. La idea detrás del método propuesto es que una variable pueda ser imputada sólo tomando en consideración aquellas que le son significativas; esto a través de un método de selección de variables. Esta tesis compara además, el desempeño del método propuesto con otros dos métodos de imputación utilizados en la literatura (KNN y missForest). Los resultados sugieren que el método propuesto puede ser de gran utilidad en escenarios categóricos complejos con alta cantidad de datos vacíos. El método propuesto es una aproximación de missForest que reduce significativamente la cantidad de variables utilizadas en la imputación.

To God and my parents Marisol and Angel.

For their unconditional love.

ACKNOWLEDGMENTS

I want to thank my advisor Saylisse Dávila for accepting me as her graduate student and believing in me. It was a blessing to have your guidance during this process. It is of great inspiration to see your compromise with all your students and with the university. I also want to thank my graduate committee, Dr. Wandaliz Torres and Dr. Noel Artiles for all their support, disposition and encouragement.

I also want to thank the Industrial Engineering Department for adopting me, helping me through this process and the opportunity of being a TA. Specially, I want to thank Dr. Betzabé Rodríguez for her great support and advice during hard times.

Thanks to my parents for their unconditional love and for supporting me in every decision I make. To my boyfriend Carlos, for always making sure I would eat. To my friends "Los guarracos" Isis, César, Miguel and Jeff, you guys are the best! I'm forever grateful for your friendship. And finally but not least, to my church, my pastors, for all their prayers.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

ACE	Artificial contrast ensemble
AUPRC	Area under precision-recall curve
CART	Classification and regression trees
CFS	Correlation-based feature selection
CPU	Central processing unit
CV	Cross-validation
DMI	Decision tree based missing value imputation
EMI	Expectation maximization imputation
EPR	Endometriosis patient registry
ERP	Endometriosis Research Program
FS	Feature selection
GA	Genetic algorithms
GBT	Gradient boosted trees
HIV	Human immunodeficiency virus
KNN	K-nearest neighbors
MAR	Missing at random
MCAR	Missing completely at random
MDA	Mean decrease in accuracy
MI	Multiple imputation
MICE	Multiple imputation by chain equations
NMAR	Not missing at random
NRMSE	Normalized root mean square error
OOB	Out-of-back error
PRESS	Predicted residual error sum of squares

PSMHS	Ponce School of Medicine and Health Sciences
RF	Random forest
RMSE	Root mean square error
SiMI	Similarity-based missing value imputation
SRI	Stochastic regression imputation
UCI	University of California Irvine
VIS	Variable importance score
VSURF	Variable selection using random forest

1. INTRODUCTION

1.1 Motivation

Researchers and practitioners in many areas of knowledge frequently struggle with missing data. In fact, 89% of the clinical experiments in leading medical journals deal with missing data (Wood et al., 2004). It arises in almost all statistical analyses for reasons such as data collection problems, equipment failures, errors in manual data entry or in cases of non-response items in survey studies with persons (Rogier et al., 2006; Gelman and Hill, 2006). The problem of missing values started to be addressed in the late 70's with few articles in the topic, but received more importance when Little and Rubin published their book *Statistical Analysis with Missing Data* (Little and Rubin, 1986; Rubin, 1976; Kalton and Kasprzyk, 1982).

Missing data is a problem because almost all standard statistical methods assume that the information is complete. As a result, the analysis of the data gets complicated, efficiency is lost, statistical power decreases, and parameter estimates may be biased due to the differences between the complete and missing data (Kaiser, 2014). Researchers often appeal to ad hoc methods such as case deletion or missing value imputation to force an incomplete data set into a complete one (Schafer, 1997). The consequence of case deletion is that potentially valuable data is discarded, which is usually worse than having missing values. *Missing value imputation*, on the other hand, refers to replacing the missing data with acceptable values, by using the data in the recorded variables to unveil the information in the incomplete cases and also make inferences on the population parameters (Andridge and Little, 2010). Note that the terms variables, covariates and features will be used interchangeably throughout this work.

When the missing cases are a small part of the data set (e.g. 5% or less), the case deletion could be a reasonable solution to the missing data problem. But, when dealing with high number of missing data, discarding them will lead to losing large amounts of information. Collecting this data often requires large amounts of time and money. This is the case when conducting studies that involve clinical trials, for instance, a new cancer treatment. First, these trials are only conducted after getting the approval from the Health Regulatory Agency, which in almost all cases involve many years of previous research work and preparation. On top of that, it will also take, on average, five years to collect the necessary data to perform robust analyses, this without mentioning the required monetary investment (Sertkaya et al., 2014). Many things can lead to missing data during the process (e.g. patients dropping out from the study, problems with data collection), thus, knowing the substantial amount of resources it takes to collect it, discarding cases is typically the least attractive option. This is why missing value imputation is a growing area of research, specially among researchers working on experiments that involve a large number of variables (e.g. demographic, environmental, clinical data).

Literature on mixed-type data imputation is limited. Most imputation methods are restricted to only one type of variable. For example, stochastic regression imputation (SRI), is used for categorical data exclusively (Sulis and Porcu, 2008), whereas regression imputation, is only used on continuous data. The options fall even shorter when complex high dimensional mixed-type data comes into play. The first attempt to overcome this gap involved maximum likelihood estimation, combining a multivariate normal model with a Poisson/multinomial model to impute continuous and categorical variables, respectively (Little and Schluchter, 1985). During the last decade, other methods based on decision trees (Stekhoven and Bülmann, 2012) and near neighbors (Kowarik and Templ, 2016) have been proposed. Yet, there still a need for new and enhanced techniques that can satisfy the ongoing necessities of the growing data world.

1.2 Objectives

The main contribution of this work lies on the development of an imputation method that can handle any type of data, including high-dimensional data with non-linear complex interactions. Random forests (Breiman, 2001) are able to handle this type of data without making any restrictive assumptions about the structure of the data. This thesis proposes a random-forest-based missing value imputation scheme that exploits the relationships among variables by means of feature selection. The premise behind the proposed scheme is that a variable can be imputed taking into account only those variables that are related to it, whether this relationship is linear or not. The idea is that when imputing a missing value for a variable, one might not need to carry out a complex optimization routine until convergence in the missing value imputation is obtained, while greatly reducing the amount of preliminary imputations needed. Thus, the *objectives* of this thesis are:

1. Design a tree-based missing value imputation scheme for complex mixed-type and high-dimensional data.

2. Understand which combinations of parameters are more suitable for different types of data sets.

3. Compare and contrast the performance of the proposed method against popular missing schemes in the literature using publicly available and simulated data.

1.3 Scope and General Organization

This thesis describes the development of a random-forest-based imputation for complex high-dimensional mixed-type data. Chapter 2 elaborates on some of the best missing value imputation schemes in the literature and describes how the proposed method compares and contrasts to them in terms of its methodology. Chapter 3 provide the conceptual framework for the proposed imputation scheme. Then, Chapter 4 describes how the feature selection methods were evaluated to select the most appropriate for the proposed imputation scheme. It also explains the experiments performed in order to optimize the proposed random-forest-based imputation. The performance of the proposed scheme was compared to K-nearest neighbors imputation (Kowarik and Templ, 2016) and the missForest algorithm by Stekhoven and Bülmann (2012) in Chapter 5. These evaluations were carried out using the Endometriosis Patient Registry data from the Ponce School of Medicine and Health Sciences (PSMHS), simulated data, and publicly available data, all having between ten to two-hundred variables. Lastly, Chapter 6 will provide a summary of the performance of the proposed scheme and other concluding remarks.

2. LITERATURE REVIEW

2.1 Missing Value Imputation

A wide array of imputation methods have been proposed in literature to deal with the problem of missing data. They encompass anything from simple, like univariate mean/mode imputation to more complex multivariate schemes that look for relationships among covariates. Many studies have compared the performance of imputation methods using benchmark data. But regardless of the simplicity or complexity of an imputation method, its execution will always depend on the fitness between the dataset, imputation method, and characteristics of the missing data (Sim et al., 2015). There are three types of missing data, which are characterized based on the underlying process believed to have led to the missing values. Values *missing at random (MAR)* are missing values whose probability of being unavailable depends on a measurable characteristic of the individual, and not on the missing value itself. The most common example of MAR values is portrayed in the US Census questionnaire, where many people refuse to provide information on their household income. The second mechanism is *missing completely at random (MCAR)*. This type is similar to MAR but the missing data has no systematic cause (e.g., a patient overlooked an item in the questionnaire) (Holmes, 2010). Lastly, when the missingness is related to the missing data itself, the data is said to be *not missing at random (NMAR)* (Penny and Atkinson, 2012).

Some imputation techniques are used to impute numerical data exclusively, whereas others strictly allow for imputing categorical data. However, these options are significantly decreased when mixed-type data comes along. One of the most popular and, by far, the simplest is *mean and mode substitution*. In this method, the missing values of a numerical (quantitative) covariate are replaced by the mean of the observed cases, while missing categorical values are replaced with the covariate's mode (Silva et al., 2011). Mean/mode imputation is easy-to-use but it is depicted as inferior since it distorts the covariance structure of the data, biasing results (Rogier et al., 2006).

Another commonly used method is *regression imputation*. Here, the missing values are predicted from a linear regression equation using the information from the complete cases (Enders, 2010). That is, the variable with missing values becomes the response and the remaining variables are used to predict this missing values. If the relationship between the variable being imputed and the remaining variables is linear, then, the method will work reasonably well. Otherwise, it will fail to understand the relationship among variables. Additionally, regression imputation also produces biased results, overestimating the correlations between covariates, and it only works on numerical data.

Multiple imputation (MI) has also been proven effective in missing value imputation. It is a Monte Carlo approach for estimating missing values in mixed-type scenarios (Rogier et al., 2006). MI generates m imputations for the missing values in a data set. Then, the m imputed data sets are analyzed using standard complete-data procedures and the results are combined to give a final result. A popular approach used to implementing MI is regression modeling, also known as multiple imputation by chain equations (MICE) (Burgette and Reiter, 2010). MICE assumes that the missing data are of MAR type and imputes them, given a conditional model per covariate. The problem with MICE comes when specifying the conditional models for large amounts of covariates with missing values, even more so, when complex interactions exist between them. Identifying these models could be an uneasy task since it is hard to adjust a model that will fit the information of the missing data and simultaneously have convergence with the estimates (López, 2005). MI compares to the proposed method in that they both perform numerous imputations of the missing values in the data. Also, in that they are both conditional approaches, but the proposed method is only conditional on those variables that have a statistically significant relationship with the variable under consideration.

2.1.1 *K*-Nearest Neighbors

Another commonly used missing value imputation scheme is K- *nearest neighbors (KNN)*. It is a non-parametric method that imputes missing data based on the outcome of the K (a user-defined constant) observations closest to the missing value. Missing data are replaced with observed values from donors with similar characteristics (Stekhoven and Bülmann, 2012). Different distance measures are used to determine the similarity between the missing values and the observed data. The most popular distance measure is the Euclidean distance, which is given by the root of squared differences between a pair of observations (x_i and y_i):

$$E(x_i, y_i) = \sqrt{\sum_{i=1}^{k}(x_i - y_i)^2} \tag{2.1}$$

Other distances commonly used are: Manhattan, Minkowski, Supremum (Singh et al., 2013) and Gower (Gower, 1971). In Yeşilova et al. (2010), KNN proved to be more effective when analyzing mixed-type data at different missing ratios. Jonsson and Wohlin (2004) evaluated the performance of the KNN method using Likert scale (ordinal) data. Results showed that it is feasible to use the KNN method with ordinal data as long as an appropriate value of K is used. They suggested K to be the square root of the number of complete cases.

K-nearest neighbors is an attractive approach due to its simplicity and effectiveness in a variety of imputation problems (Liao et al., 2014). But one of the drawbacks of the KNN method is that it only imputes a missing value based on its K-nearest neighbors, which makes it a conditional approach (López, 2005). Also, it is not clear which value of K should be used. Overall, the only resemblance between the proposed method and KNN is the way donors are handled. KNN and the proposed method are both conditional approaches. In KNN, donors represent observations with similar characteristics, whereas in the proposed scheme, a random forest is trained only on those covariates selected to be significant to the covariate being imputed by the feature selection method.

2.1.2 Random Forests

Tree-based missing value imputation techniques are also widely used in mixed-type data sets with complex interactions between variables. Decision trees are non-parametric supervised methods used for classification and regression. They are simple and produced by algorithms that identify various ways of splitting a data set into branch-like segments. Their goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data (Pantanowitz and Marwala, 2009). Decision trees are divided in two main categories: (1) classification and (2) regression. The first is used to predict categorical variables, while the latter is used for continuous variables. *Classification and regression trees (CART)* analysis is a technique that uses either of the tree types for predicting both continuous and categorical variables (Gould, 2000). The CART algorithm builds the tree by recursively partitioning the data set into non-overlapping regions (branches) and, then, use the tree to predict the missing value for the covariate being treated as the dependent variable (Breiman et al., 1984).

A *Random forest (RF)* is an ensemble of decision trees that performs, both, classification and regression by drawing M bootstrap samples from the original training data, using each of these M samples to build M trees within the ensemble. They can be easily adapted to the task of missing value imputation (Breiman, 2001). In fact, random forests can work around missing values without imputing them because they can decide what to do on a split based on the best surrogate for the variable under consideration. Additionally, random forests provide a measure of variable importance scores (VIS), which is a measure of how much the prediction error increases when out-of-bag (OOB) data for that variable is permuted while all others are left unchanged (Liaw and Wiener, 2014). The variable importance score for each variable is then computed as the mean importance over all trees. For a single decision tree, the measure of variable importance proposed by Breiman et al. (1984) is given:

$$VI(x_j, T) = \sum_{t \in T} \Delta I(x_j, t), \tag{2.2}$$

where $\Delta I(x_j, t) = I(t) - p_L I(t_L) - p_R I(t_R)$ is the decrease in impurity due to an actual or potential (surrogate) split on numerical predictor x_i at a node t of the optimally pruned tree, T and $p_L(p_R)$ denotes the proportion of cases assigned to the left (right) child node of t. Note that Equations 2.2 refers to a single decision tree. For ensembles of M trees, the VIS of a predictor is obtained by averaging over all its VIS across the ensemble as in:

$$VI(x_j) = \frac{1}{M} \sum_{m=1}^{M} VI(x_j, T_m), \tag{2.3}$$

where T_m denotes tree m. This process of averaging across all VIS in the ensembles has a stabilizing effect that leads to a more reliable predictor of variable importance.

For classification purposes, a measure of node impurity commonly used is the gini index:

$$Gini(t) = 1 - \sum_{k \neq k'} p_k^t p_{k'}^t \tag{2.4}$$

where p_k^t is the proportion of cases in node t whose response label equals k $(y = k)$. It is zero when t has cases only from one class and is maximized when classes are evenly mixed. On the other hand, Breiman (2001) suggests the use of the mean decrease in accuracy (MDA) for assessing node impurity in regression problems. The MDA is assessed for each variable by removing the association between that variable and the target. This is achieved by randomly permuting the values of the variable and measuring the resulting increase in error.

Various random forest-based algorithms have been evaluated in the literature to assess their performance in missing data imputation. *rfImpute* was used in Pantanowitz and Marwala (2009) to evaluate missing HIV data from a clinic study survey. This algorithm first imputes the missing values using the mean and mode of the observed values. Then, a random forest is trained with the complete data and its proximity matrix is used to update the imputation of the missing values. For continuous covariates, the imputed value is the weighted average of the non-missing observations, using proximities as weights. For categorical predictors, the imputed value is the category with the largest average proximity. In Pantanowitz and Marwala (2009), the rfImpute algorithm resulted to be superior in terms of computation time and accuracy in comparison to other supervised methods such as neural networks.

Two other decision tree and random forest-based imputation methods were presented in Rahman and Islam (2013): (1) *Decision tree based Missing value Imputation (DMI)* and (2) *SiMI*. DMI uses decision trees to identify horizontal segments with higher similarity among variables. The algorithm applies expectation maximization (EMI) (Schneider, 2001) to impute numerical missing values and mode imputation for categorical missing values (Rahman and Islam, 2011). In a similar way, SiMI uses a decision forest algorithm, such as SysFor (Islam and Giggins, 2011), to build k decision trees over the complete data. The main difference between them is how similarity is assessed among variables. SiMI algorithm finds the intersections of the records belonging to the leaves of the forest and merge the small-sized ones with another intersection, maximizing a similarity metric S_j within the merged intersection. S_j is calculated using the average distance of pairs of records between two intersection. The distance between two records is calculated using the Euclidean distance for numerical attributes and a similarity based distance for categorical attributes. Finally, missing values are imputed the same way as DMI, using EMI and mode imputation for numerical and categorical missing values, respectively.

In Rahman and Islam (2013), SiMI the was top performer in terms of imputation accuracy when compared to other EMI techniques. They used a random forest which identifies even better correlations among the attributes and similarities between the records, therefore, giving better accuracy in the imputation. They also acknowledge this method is more complex and computationally expensive since they use decision trees and forests. DMI and SiMI resemble the proposed method in that they impute the missing values using only a subset of the data that is most highly correlated to the missing value. Another similarity is that they are both based on decision tree methods, such as the proposed scheme.

The method in literature that most closely resembles the proposed approach is the iterative non-parametric imputation scheme known as ***missForest***. In a similar approach to the proposed method, missForest can be used with any type of data: numerical, categorical, or even mixed-type data. First, it makes an initial guess of the missing values of a data set X using mean/mode imputation. Then, it sorts the covariates (X_i) according to their amount of missing values ($y_{mis,i}$) in increasing order. The imputation of each X_i is done by training a RF on the observed values ($x_{obs,i}, y_{obs,i}$) of the data set and then predicting the missing values with the trained RF. This procedure is repeated until a criterion γ is met. The stopping criteria is met when the difference between the newly imputed data matrix and the previous one increases for the first time with respect to both covariate types (categorical and numerical). missForest also provides the user with an estimate of the imputation error, which is based on the out-of-bag (OOB) error estimate of random forest. In Stekhoven and Bülmann (2012), this method proved to have better performance in comparison to other multiple imputation and regression imputation schemes, especially in mixed-typed high-dimensional data with complex interactions.

Overall, the main difference between this approach and the proposed method is how the random forest is built on the observed data. In the proposed approach, only those covariates that are related to the variable being imputed are used to build the random forest. Meanwhile, missForest uses all predictor variables in the data set, regardless of the dimensionality of the problem and regardless of whether these predictors have any relationship with the predictor being imputed. As a result, the proposed method can be used as an approximation to missForest in scenarios where the computational complexity of the imputation problem becomes an issue and the relationships among predictors are known or can be assessed a priori.

2.2 Feature Selection

The selection of relevant features or feature selection (FS) is a topic that has grown in popularity in recent years with the increase of complex, high-dimensional data. The presence of redundant or irrelevant features constitutes a problem since it can degrade the performance of learners in terms of speed and predictive accuracy. As a matter of fact, the amount of training samples needed to reach a given accuracy grows exponentially with the amount of irrelevant features in a data set (Langley and Iba, 1993). But, how to know when a feature is relevant? In Kohavi and John (1997), the authors portrayed three different types of features: those that are strongly relevant to a target, weakly relevant, or irrelevant. They described a feature X_i be strongly relevant when its removal results in the deterioration of prediction accuracy. On the other hand, when X_i is weakly relevant, it implies that the feature can sometimes contribute to prediction accuracy depending on which other features are removed. Finally, an irrelevant feature does not add any information to a model, thus, it never contributes to prediction accuracy. With this in mind, many algorithms have been developed to measure how useful is each variable in a data set. The objective of *feature selection* is to select a small subset of features from the original data that will provide the most significant information from a target (Kira and Rendell, 1992). Therefore, it helps to better understand the data, reduces computational requirements, improves predictor performance, and facilitates to identify which features are relevant to a specific problem (Chandrashekar and Sahin, 2014).

2.2.1 Classification of Feature Selection Methods

Feature selection methods are grouped into three main classes: (1) filter, (2) wrapper, and (3) embedded.

Filter Feature Selection

Filter selection methods are the most common approach of feature selection. They apply variable ranking techniques as the principle criteria for variable selection (Shardlow, 2008). A suitable ranking criterion is used to score the variables and a threshold is used to remove variables below this threshold. Once this ranking/score has been computed, a feature set composing of the best N features is created. An advantage of filter methods is that they are easy to implement and fast to execute; however, most of them do not take into account the interaction with the learner or even among the features (Saeys et al., 2007). Finding a suitable learning algorithm for filters can be a difficult task as the underlying learning algorithm is ignored. Some examples of filter methods are information gain and correlation coefficient scores.

Correlation-based feature selection (CFS) uses a correlation based heuristic to evaluate a subset of features. The rationale behind this heuristic is that an important or good feature is highly correlated with the class but uncorrelated with one another (Hall, 2000). The measure used to evaluate the score of a feature subset S is given by:

$$M_s = \frac{k\overline{r_{cf}}}{\sqrt{k + k(1-k)\overline{r_{ff}}}} \tag{2.5}$$

where k is the amount of features in S, $\overline{r_{cf}}$ is the average feature-class correlation and $\overline{r_{ff}}$ is the average feature-feature correlation. CFS calculates a matrix of feature-class and feature-feature correlations from the training data and then searches the feature subsets using the best first search algorithm (Kohavi and John, 1997). The best subset is the one with highest M_s after five consecutive subsets not showing improvement.

CFS uses symmetrical uncertainty to measure correlation between discrete features in discrete class problems as in:

$$SU(x,y) = 2\left[\frac{IG(x|y)}{H(x) + H(y)}\right] \tag{2.6}$$

Information gain (IG), is the amount by which the entropy, $(H(x))$ of a variable x is decreased after observing values of another variable y. $H(x)$ is the entropy of a variable x and $H(x|y)$ is the entropy of x after observing the values of y.

$$IG(x|y) = H(x) - H(x|y) \tag{2.7}$$

$$H(x) = -\sum_i P(x_i)\log_2(P(x_i)) \tag{2.8}$$

$$H(x|y) = -\sum_j P(y_j)\sum P(x_i|y_j)\log_2(P(x_i|y_j)) \tag{2.9}$$

On the other hand, CFS uses the Pearson correlation coefficient, as described in 2.10, to measure association between continuous variables in Equation 2.5.

$$r(x,y) = \frac{\sum_i(x_i - \overline{x_i})(y_i - \overline{y_i})}{\sqrt{\sum_i(x_i - \overline{x_i})^2}\sqrt{\sum_i(y_i - \overline{y_i})^2}} \tag{2.10}$$

where $\overline{x_i}$ and $\overline{y_i}$ are the mean of variables x and y, respectively.

Correlation-based algorithms are significantly faster than other selection methods and have high prediction accuracy when analyzing high dimensional data (Yu and Liu, 2007; Doshi and Chaturvedi, 2014). In summary, CFS algorithms can highly reduce the dimensionality of the data while maintaining the performance of learning algorithms.

ReliefF is another commonly used filter method, extended from the original Relief (Kira and Rendell, 1992), that can deal with multi-class problems and incomplete data. It weights the features based how well their values distinguish between instances that are near to each other (Robnik-Sikonja and Kononenko, 2003).

ReliefF randomly chooses an instance R_i and searches for its k nearest neighbors from the same class (near-hits, H_j) and also its k nearest neighbors from each of the different classes (near-miss, M_j) using Euclidean distance. In regression problems the predicted value is continuous, therefore H_j and M_j cannot be used and a kind of probability that the predicted values of two instances are different is introduced.

This probability can be modeled with the relative distance between the predicted values of two instances. ReliefF updates a weight estimation $W[A]$ for all attributes A, estimated as the average squared difference between R_i and H_j and M_j:

$$W[A] = \frac{-(R_i - M_j)^2 + (R_i - H_j)^2}{m} \tag{2.11}$$

Then, the algorithm selects those features whose average weight is greater than or equal to a given threshold τ. Kira and Rendell (1992) showed that τ can be determined by Chebyshev's inequality (Taylor, 2016) as $\tau = \frac{1}{\sqrt{\alpha * m}}$ for a confidence level (α). They also showed that there is a clear contrast between relevant and irrelevant features, allowing τ to be determined by inspection as well.

Overall, ReliefF algorithms are good in detecting conditional dependencies, thus, they are robust and noise-tolerant (Robnik-Sikonja and Kononenko, 2003).

Wrapper Feature Selection

The *wrapper approach* uses the prediction performance of a given induction algorithm/learner (a classifier) to assess the usefulness of subsets of features in the data (Guyon and Elisseeff, 2003). Here, the feature selection is "wrapped" around the learner and does an exhaustive search for variables in the data. The subset with highest performance is then chosen (Kohavi and John, 1997). Wrappers have the ability of taking into account variable dependencies by considering the induction algorithm as a black box. They can be computationally intensive and prone to overfitting, which, in turn, introduces bias and increasing the classification error. A common example of a wrapper method is the recursive feature elimination algorithm, which employs backward selection. Here, the model is fitted using all the features first and each one is ranked based on their importance to the model. The model is then refitted with subsets of size S that vary according to the amount of features. The subset S_i, which contains the top ranked features based on the performance of the model, is selected.

A popular wrapper approach is feature selection using ***Genetic Algorithms (GA)***. This method mimic properties of biological evolution (e.g crossover, inheritance, mutation, and selection) applying heuristic search methods to optimize the amount of variables in a data set (Pantanowitz and Marwala, 2009). An initial set of candidate feature subsets (S_i) are created and their corresponding performance is calculated. The fitness values are some measure of model performance, such as the root mean square error (RMSE) or classification accuracy. The subsets S_i with the best fitness values are combined randomly to produce another subsets which make up the next population. This process is repeated many times until the best solution is found (Chandrashekar and Sahin, 2014). GA have demonstrated to be effective handling both small and high-dimensional data, however, they can be more computational intensive than other feature selection methods (Mohamad et al., 2004).

Embedded Feature Selection

Embedded methods are somewhat similar to wrappers but they perform the feature selection as part of the learning process. Hence, they are specific to a given algorithm that learns which features best contribute to the performance of a model. The most common examples of embedded methods are regularization methods (e.g. Lasso and Ridge regressions) (Brownlee, 2014). Embedded FS approaches also include decision tree-based methods. They carry out a greedy search through the space of decision trees using an evaluation function to select the attribute that has the best ability to discriminate among the classes. They partition the training data based on this attribute and repeat the process on each subset, extending the tree downward until no further discrimination is possible (Blum and Langley, 1997).

Embedded methods have the advantage that they include the interaction with the classifier, while at the same time being less computationally intensive than wrapper methods. Embedded methods have the advantage that they include the interaction with the classifier and are less computationally intensive than wrapper methods.

Artificial Contrast with Ensembles (ACE) is an embedded FS method that uses parallel ensembles of decision trees or random forest to select the best features in a data set. ACE creates a traditional statistical inference setting by building N times a random forest of M trees. It generates N random forests and calculates N VIS for $2J$ covariates in the training data set. That is, the J predictor covariates and J additional artificial covariates. An artificial covariate x_j^* is simply a random permutation of the observed values of predictor x_j. This process is repeated for each feature until the set of J artificial covariates has been generated. Since these artificial predictors are random permutations of the original, they share the same marginal distributions, but they are by no means related to the response. The idea is that their variable importance scores must be low since they are not related to the response, and, hence, they can be used to create a threshold to better understand when the magnitude of a VIS is indeed large. From each of the N random forests, a VIS is recorded for each predictor as well as a large quantile (q), often $q_{0.8}$ or higher, for the artificial covariate VIS's. At the end of this iterative process, a paired t-test is used to determine whether each of the predictors has a VIS that is larger than the large quantile from the artificial VIS. All predictors that show a statistically significant improvement over the artificial variables are selected as important; the remaining predictors are discarded (Tuv et al., 2009).

The R package ***VSURF*** (Genuer et al., 2015a), is also an embedded method that uses random forests as a mean to select important features. It is based in a two-strategy approach: (1) preliminary elimination and ranking, and (2) variable selection. In the first step, the objective is to find important variables highly related to the response using random forest VIS. Here, variables are sorted according to their mean VIS in decreasing order. A threshold value is given by the minimum predicted value of a pruned CART tree model fitted to the curve of the standard deviations of the VIS. Only the variables with an average VIS greater than this threshold are retained for the next step.

In the second step, a series of embedded random forest are modeled starting with a random forest build with only the most important variable and ending with a model having all the variables selected in the first step. Then, the minimum mean OOB error, $min(\overline{X_{oob}})$, of these models and its associated standard deviation (s_{oob}) are computed. Finally, the smallest model (and hence its corresponding variables), having a mean OOB error less than the minimum mean OOB error plus its standard deviation is selected ($\overline{X_{oob}} < min(\overline{X_{oob}}) + s_{oob}$).

3. METHODOLOGY

This chapter presents the detailed description of the proposed imputation scheme and the data used. The premise behind the proposed approach is that a variable can be imputed taking into account only those other variables that are related to it. When a missing value in a specific variable must be imputed, the imputation algorithm might not need to make a large amount of preliminary imputations in all other covariates with missing values or carry out a computationally-intensive optimization routine until convergence in the missing value imputation is obtained. The method is currently implemented in two phases, the first one being the feature selection and the second, the missing value imputation. Figure 3.1 depicts an overview of the evaluation done throughout this work.

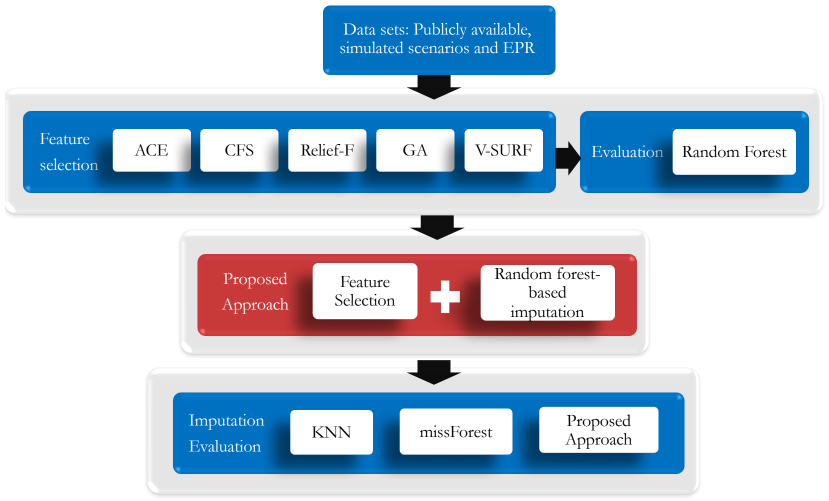


Fig. 3.1.: Flow diagram of proposed approach

3.1 Proposed Imputation Approach

The main steps of our proposed imputation method are described in Algorithm 1. Let D_{mis} be a $n \times p$-dimensional data set having missing values. By default, the algorithm uses a forest of 125 trees (T) unless specified by the user. The maximum amount of imputations, k, per missing record is also set to 30, unless the user specifies otherwise. This value k was set to 30 to ensure convergence of the imputation (Comulada, 2015). The categorical and numerical impute change thresholds, Δ_c and Δ_n, are also require in the algorithm. Δ_c is define to 4, meaning that the imputation of record r in a categorical missing variable X_i stops after 4 unchanging consecutive imputations. In a similar way, the numerical impute change threshold (Δ_n) is set to 2.5%, meaning that the imputation for a record $r \in X_i$ stops when the difference between the actual imputation (P_{new}) and an old one (P_{old}) is 2.5% or less.

Variables with missing values are first identified in the incomplete data set D_{mis}. A vector `M` is created with the amount of missing values in each missing variable X_i. Afterwards, an initial guess of the missing values in D_{miss} is carried out using `mean/mode` imputation, prior the feature selection. The statistically significant variables for each $X_i \in D_{mis}$ are then determined using genetic algorithms feature selection (GA). An important variables incidence matrix `F` is created as a result. The columns in `F` refer to the variables with missing values, and the rows of the matrix refer to all variables in the data set. In this incidence matrix, a value of zero in element i, j implies predictor i was not detected as to have a significant relationship with predictor j. Otherwise, element i, j would have a value of one, portraying a significant relationship between variables i and j. The imputation of $X_i's$ is carried out in increasing order of missing values. Therefore, `O` is the vector of indices of columns in D_{mis}, sorted in increasing order of missing values. `O` indicates the order in which missing variables X_i are imputed in the data set.

A random forest of T trees is built for each variable with missing values X_i, treating the vector of its observed values $D_{train}[X_i]$ as the response. The data set

```
Input    : A data set $D_{mis}$ having missing values
Output   : A data set $D_{imp}$, with all missing values imputed
Require: $T \leftarrow 125$; /*Number of trees in the random forest.*/;
         $k \leftarrow 30$; /*Maximum amount of imputations.*/;
         $\Delta_c \leftarrow 4$; /*Categorical impute change.*/;
         $\Delta_n \leftarrow 0.025$; /*Numerical impute change.*/;
foreach $X_i$ do
    $m_i \leftarrow$ CountMissVal($X_i$) /*$M$ is a vector of the frequency of missing
    values in each $X_i$.*/;
end
$D_0^{imp} \leftarrow$ MeanMode($D_{mis}$) /*Initial imputation using mean/mode.*/;
foreach $x_i$ do
    $f_i \leftarrow$ GA($D^i_{imp,0} \sim D^{-i}_{imp,0}$ /*Run feature selection.*/;
end
$F \leftarrow$ CreateIncidenceMatrix($F$) /*Create Important variables incidence
matrix.*/;
$O \leftarrow$ Sort($M$) /*Vector of indices of columns in $D_{mis}$ sorted in increasing
order of missing values.*/;
foreach $X_i \in D_{mis}$ do
    $cols \leftarrow$ which($f_i$ == 1) $\cup$ which(ColNames($D_{mis}$) == "$Y$") /*Important
    variables and overall $Y$ of data.*/;
    $x_i^{obs} \leftarrow$ NamesCompleteCases($X_i$) /*Row names of observed values in $X_i$.*/;
    $x_i^{mis} \leftarrow$ NamesInCompleteCases($X_i$) /*Row names of missing values in
    $X_i$.*/;
    $D_{train} \leftarrow D_0^{imp}[x_i^{obs}, cols]$ /*Training sample.*/;
    $D_{test} \leftarrow D_0^{imp}[x_i^{mis}, cols]$ /*Testing sample.*/;
    foreach record $r$ in $X_i$ do
        while $j$ in $1 : k$ or $\delta_n > \Delta_n$ or $\delta_c \neq \Delta_c$ do
            $j = 1$;
            $rF \leftarrow$ randomForest($D_{train}[X_i] \sim D_{train}$, T) /*Fit random forest.*/;
            $P \leftarrow$ Predict($X_i[x_i^{mis}]$);
            $\delta_n \leftarrow$ ChangeInImpNum($P_{new}$, $P_{old}$);
            $\delta_c \leftarrow$ ChangeInImpCat($P_{new}$, $P_{old}$) ;
            $j = j + 1$;
        end
    end
    if is.numeric($X_i$)==TRUE then
        $X_i^{imp}$= mean($P_{new}, P_{old}$);
    end
    else
        $X_i^{imp} = P_{new}$
    end
end
Return $D_{imp}$ /*Imputed data set.*/
```

Algorithm 1: Proposed Imputation Scheme

used to train the random forest (D_{train}) includes the response of the overall supervised learning scenario, Y, as well as all other predictors that have been selected by the feature selection algorithm, as indicated by the elements that are equal to 1 in column j of the incidence matrix `F`. The random forest draws T bootstrap samples of D_{train} to build the T decision trees in the ensemble. The trained random forest `rF` is then used to predict the missing values in X_i. The testing sample D_{test}, consists of Y and the important variables of X_i, and is used to predict X_i's missing values.

This process is repeated k times or until a stopping criterion Δ is met. The change in imputation of numerical variables δ_n, is the percentage of difference between the new imputation of a missing value, P_{new} and its previous, as given by P_{old}. When δ_n is less than or equal to Δ_n, the algorithm stops imputing values for that record r of X_i and moves on to the next one. In the case of categorical variables, if the new imputed value has not change in the last Δ_c iterations, then, the algorithm stops the imputation process for that record and moves to the next r in X_i. Finally, the overall imputation of numerical missing values, X_i^{imp}, is given by the average of the j imputations used in δ_n. Additionally, the final imputation for a categorical missing value is given by its last imputation.

3.2 Data

Different data sets were used throughout this work to evaluate the feature selection, the parameter tunning of the random forest imputation, and the final comparison of the proposed imputation scheme with KNN and missForest. These data sets are divided into three groups mainly: publicly available data, simulated scenarios, and a special case study data set called Endometriosis Patient Registry (EPR).

3.2.1 Publicly Available Data

Three data sets available at the UCI machine learning repository (Lichman, 2013) were used: Cleveland Heart Disease (Janosi et al., 1988), Breast Cancer Wisconsin (William, 1992), and Sylva Ecology (Service, 2006). Table 3.1 gives a general description of all three data sets.

The Cleveland Heart Disease data has a total of 14 mixed-type variables, including demographical information, symptoms, and results on clinical analyses. The response variable of this classification data set is an integer valued from 0 to 4, denoting the presence of heart disease in the patient. Heart disease generally refers to conditions that involve narrowed or blocked blood vessels that can lead to a heart attack, chest pain (angina) or stroke (Staff, 2014). A value of zero represents the absence of heart disease, whereas values from 1-4 represent the presence of any of the conditions previously described.

Figure 3.2 shows the missing patterns in the data set, which contains approximately 2% missing values. The rows in the grid display the different patterns found within the missing variables, in order of occurrence. Out of the 13 variables in the data set, 2 of them have missing values as portrayed by the number of vertical bars in the bar plot (Figure 3.2). Over 98% of the rows represent complete cases, while the remaining cases are missing one value in either predictor *ca* or *thal*.

Table 3.1: General description of publicly available data sets.

Data set	***Records***	***Num. Attr.***	***Cat. Attr.***	***Variables with MV's***	***Missing***	***Response***
Heart disease	303	5	8	2	2%	Class
Breast Cancer	699	0	10	1	2%	Class
Sylva	13,085	51	177	0	0%	Class

The second, Breast Cancer Wisconsin, is also a small classification data set that includes characteristics of breast cells from biopsy studies in patients. All of the variables in this set are categorical with the response being the diagnosis of the breast tissue being malignant or benign (Wolberg and Mangasarian, 1990). Figure 3.3 shows

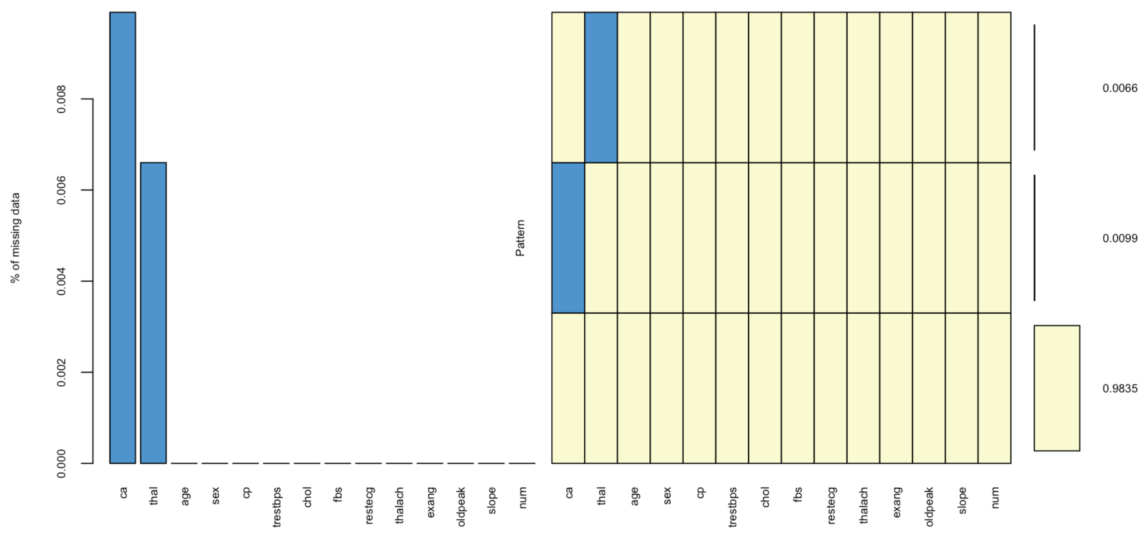


Fig. 3.2.: Visualization of missing data in Heart Disease data set. The yellow cells show available information and the blue cells indicates missing values. The different rows in the grid shown in the plot reflect the most common patterns in the data set.

the missing patterns in the data set. It has approximately 2% missing values, which correspond to variable *Bare.nuclei*.

Finally, the Sylva Ecology data set, is the biggest data set used in this study. It is a mixed-type data set with over 200 variables and over 10,000 records. The set is a binary-classification of forest cover in the United States. It classifies Ponderosa Pine trees vs everything else. An interesting aspect of this data set is that half of the variables are noise variables.

3.2.2 Simulated Scenarios

Six different simulated data sets were also evaluated throughout this work. These data sets can be divided into three main groups as well: (1) data sets with linear relationships (portrayed by rows two and three in Table 3.2), (2) nonlinear relationships (rows four to six), and (3) an isolated simulated data set (row one).

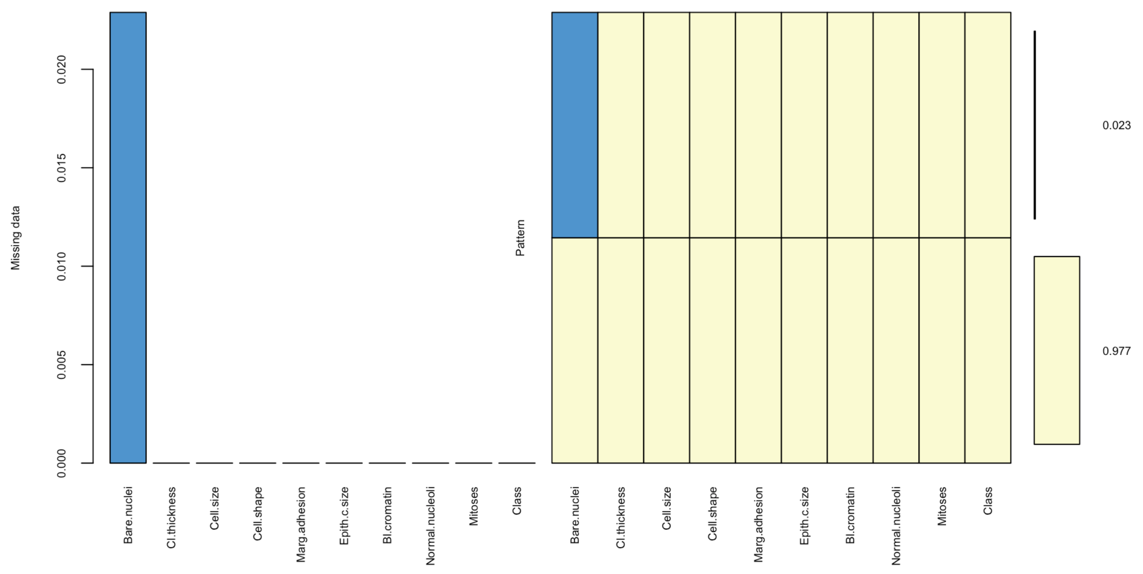


Fig. 3.3.: Visualization of missing data in Breast Cancer Wisconsin data set. The yellow cells show available information and the blue cells indicates missing values. The different rows in the grid shown in the plot reflect the most common patterns in the data set.

Table 3.2: General description of simulated data scenarios.

Data set	***Records***	***Num. Attr.***	***Cat. Attr.***	***Variables with MV's***	***Missing***	***Response***
SimOriginal	500	4	6	0	0%	Num
LinReg203	500	151	52	0	0%	Num
LinClass145	500	86	59	0	0%	Class
NonLinReg70	500	70	0	0	0%	Num
NonLinReg38	500	27	11	0	0%	Num
NonLinReg125	500	77	48	0	0%	Num

Generated Linear Data

The linear relationship data sets were generated using Tuv et al. (2009) linear data generator. The *LinReg203* is a regression data set with 203 mixed-type variables. The categorical predictors have cardinality ranging from 2 to 10 levels. It has 4 statistically important variables, 99 correlated redundant variables (regression $R^2 \sim 0.5$ and $R^2 \sim 0.1$) and 100 noise variables, following a standard normal distribution $N(0, 1)$. The response was generated based on the additive model $y = x_1 + x_2 + x_3 + x_4 + \varepsilon$, where $\epsilon \sim N(0, 1)$ and $x_1 ... x_4$ are the important variables.

The *LinClass145* is also a mixed-type linear data set with a two-class response variable. The categorical variables were also generated with the same cardinality as *LinReg203*. It has 15 important variables with random linear dependency between them and the response variable. A random weight, following an uniform distribution $U(0, 1)$, is given to each of them when generating Y. It also has 30 redundant which are random combinations of one important numerical variable plus random noise, $N(0, 1)$. In addition, this data set has one hundred noise variables with distribution $N(0, 1)$.

Generated Nonlinear Data

The nonlinear data sets were generated using the nonlinear data generator described in Friedman (2001). The *NonLinReg70* data set is a replica of the nonlinear data set used in Tuv et al. (2009). This data set has 10 important variables from which the target is generated. It also has 20 redundant variables, that are a random linear combination of 3 important variables plus a random noise, based on an uniform distribution. Finallu, there are 40 noise variables, $N(0, 1)$, in the data set. The numeric response variable is a weighted sum of 10 multidimensional Gaussians, each Gaussian involving about 4 variables randomly drawn from the 10 important variables.

The *NonLinReg38* is also a regression data set with mixed-type variables. It has 8 important, 20 noise, and 10 redundant variables that were generated using the same structure as *NonLinReg70*. *NonLinReg125* follows the same pattern as well but it has 20 important, 75 noise and 30 redundant variables. The categorical variables in both

NonLinReg38 and *NonLinReg125* were generated with cardinality ranging from 2 to 10 levels.

SimOriginal Data

The *SimOriginal* data set was simulated using R (R Core Team, 2016a). It has a numerical response variable and 10 mixed-type predictors (four numerical ($X_1 - X_4$) and six categorical ($X_5 - X_{10}$)). The categorical predictors consider variables, which range from low (binary) to high cardinality (10 levels). The data was simulated so that half of the predictors are related ($X_1 - X_5$), while the remaining half are independent ($X_6 - X_{10}$). The structure of the simulated data is shown in Table 3.3. The value of predictor X_2 depends on the value of predictor X_1 plus random noise based on the standard normal distribution, $N(0, 1)$. Predictors $X_3 - X_5$ also depend on the value of X_1 and so on. In addition, predictors X_3 and X_4 further depend on X_2. For X_3, the relationship with X_2 is additive, whereas its relationship with X_4 is multiplicative. Finally, the response was generated based on an additive model using X_1, X_2 and a random uniform noise.

Table 3.3: SimOriginal data structure. Simulated data includes 500 observations in 10 mixed-type data predictors and one numerical response variable.

$X_{00} \sim UNIF(-0.25, 0.25)$	$X_{01} \sim N(0, 1)$
$X_1 = X_{00} + X_{01}$	$X_7 \sim DUNIF(1, 3)$
$X_2 = 2 * X_1 + X_{01}$	$X_8 \sim DUNIF(1, 5)$
$X_3 = X_1 + X_2 + X_{01}$	$X_9 \sim DUNIF(1, 7)$
$X_4 = X_1 * X_2 + X_{01}$	$X_{10} \sim DUNIF(1, 10)$
$X_5 = \begin{cases} DUNIF(2, 4) & X_1 \geq 0 \\ 1 & elsewhere \end{cases}$	$Y \sim 3 * X_1 + X_2 + X_{00}$
$X_6 \sim DUNIF(1, 2)$	

3.2.3 Endometriosis Patient Registry (EPR)

The Endometriosis Patient Registry data set, from the Endometriosis Research Program (ERP) at the PSMHS, is relatively large with a moderate number of missing values. This registry, described in Table 3.4, gathers information of women with endometriosis-related symptoms, some of which chose to be diagnosed via an invasive surgical procedure (e.g. laparoscopy, laparotomy). It includes data on demographical information, endometriosis-related symptoms, pre-existing conditions, lifestyle choices, and family and medical history for a total of 99 different variables and 2,763 records.

The EPRs main challenge is the fact that it has more than 37,000 (14%) missing values. If any record with missing values were to be discarded, there would be zero records left. This issue comes up from the fact that this data was collected using a survey that was subject to changes over a ten-year period (e.g. some questions were added, some questions were removed). Figure 3.4 shows a visual representation of missing values in the EPR data set. The grid displays the different patterns found within the variables with missing values in order of occurrence. Out of the 99 variables in the data set, 25 of them have missing values as portrayed by the number of vertical bars in Figure 3.4. The most common pattern is records with missing values in 22 variables as shown in the first line in the grid. In fact, this pattern is present in 21% of the records in the data set. Table 3.5 also breaks down the proportion of incomplete values in the missing variables of the data set, with some of them having up to 99% missing records (*Pap test class* variable). It is important to note that this specific variable was removed from the data set since it is nearly non-existent. This clearly describes the challenging nature of the EPR. Whenever any given record needs to be imputed, there is a need for up to 21 preliminary imputations per record. For this specific pattern, there would be 21 imputations.

Thus, EPR was the motivation behind the proposed imputation method, for its high dimensional mixed-type nature and large amount of missing values. Also, it is the most complete data repository for endometriosis patients in the island and it took a significant amount of time and effort to gather it. This data is key to further analysis for the treatment of endometriosis.

Only complete records of this data set were used in order to perform the final evaluation of the imputation methods. Constant variables were also removed, hence, giving a final data set size of 91 variables and 421 records.

Table 3.4: General description of the EPR data set.

Data set	***Records***	***Num. Attr.***	***Cat. Attr.***	***Variables with MV's***	***Missing***	***Response***
EPR	2,763	5	94	25	14%	Class

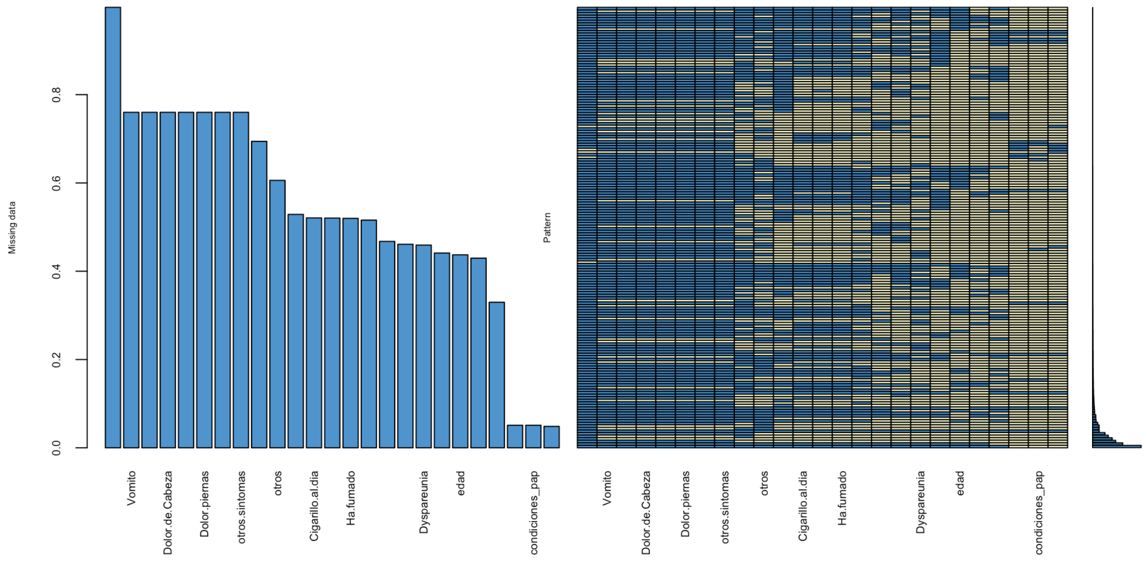


Fig. 3.4.: Visualization of missing data in the EPR data set. The yellow cells show available information and the blue cells indicates missing values. The rows in the grid reflect the most common missing patterns in the data set.

Table 3.5: Proportion of incomplete values in the 25 missing variables of the EPR data set.

Variable	***% Missing***	***Variable***	***% Missing***
1. Pap test class	99.8%	14. Smoke	52.0%
2. Vomiting	76.0%	15. Cramps	51.6%
3. Irritated stomach	76.0%	16. Problems getting pregnant	46.8%
4. Headache	76.0%	17. Amount of pregnancies	46.1%
5. Chronic Pelvic Pain	76.0%	18. Dyspareunia	45.9%
6. Leg numbness	76.0%	19. Age at menarche	44.1%
7. Bloating	76.0%	20. Age	43.7%
8. Other symptoms	76.0%	21. Dysmenorrhea	43.0%
9. Age symptoms started	69.4%	22. Period length	33.0%
10. Other conditions	60.6%	23. Constipation	5.1%
11. Number of days between period	52.9%	24. PAP test	5.1%
12. Amount cigarettes the patient smokes per day	52.1%	25. Ovarian Cysts	4.8%
13. Years smoking	52.0%		

4. ANALYSIS

The proposed approach is a combination of feature selection along with random forest-based imputation. In order to develop our proposed scheme, five different feature selection methods (ACE, CFS, GA, ReliefF, and VSURF) were considered and evaluated in a cross-validation setting. Extensive parameter tuning was carried out to determine the most suitable combination of parameters for the proposed imputation method. Finally, the performance of the proposed scheme was assessed and compared to two other well known imputation methods, KNN, and missForest. This chapter describes the framework of the analysis followed through the design of the proposed missing imputation approach (Section 4.1) and its performance evaluation (Section 4.2).

4.1 Development of Proposed Imputation Approach

The proposed imputation approach consists of two phases: (1) selecting the important features of each missing variable in the data set, and (2) imputing the missing variables based on the significant variables chosen by the feature selection. Phase 1 will includes an extensive evaluation of feature selection methods for mixed-type, high-dimensional data. Phase 2 involves a thorough parameter tuning of the random-forest-based imputation.

4.1.1 Evaluation of Feature Selection Methods

ACE, CFS, ReliefF, GA, and VSURF were evaluated in the selection of the feature selection method to be used in the proposed imputation approach. Figure 4.1 shows a general description of the undergone evaluation. The performance of the feature selection methods was assessed using five-fold cross-validation (CV) on a random forest model. The performance of the feature selection methods was also assessed in more detail using the simulated data sets since their structure was known. Seven out of the ten data sets discussed in Section 3.2 were employed in this evaluation, as depicted in Figure 4.1. We included regression and classification data sets of various sizes. All of the experiments in this phase were performed using the R statistical software (R Core Team, 2016b).

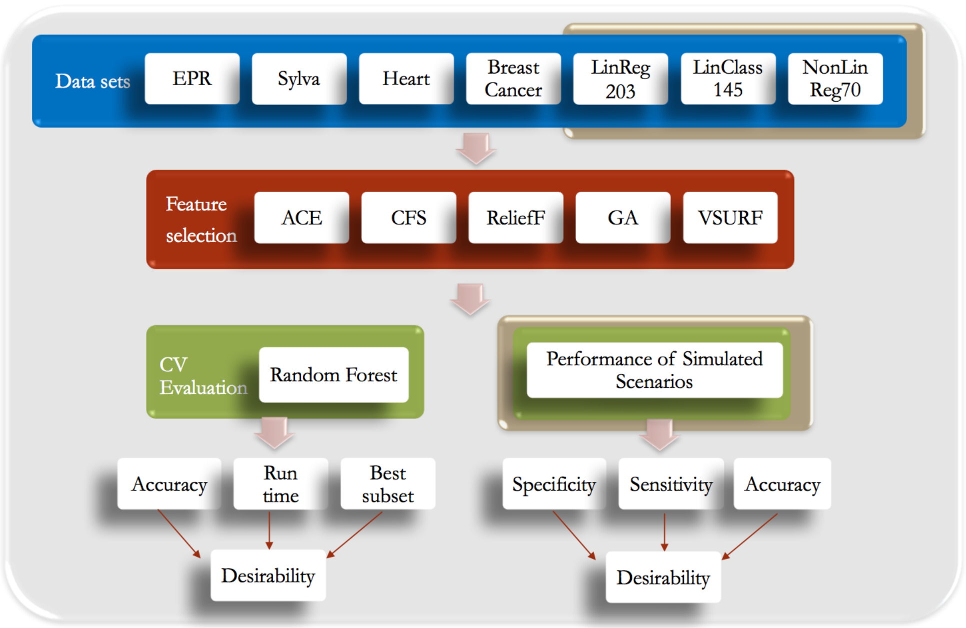


Fig. 4.1.: Flow chart of steps carried out for the selection of FS method.

Thirty bootstrap samples of each of the seven data sets were created, therefore, the five-fold cross-validation was carried out 30 times for each FS method and each data set.

The *ACE* method used in this evaluation is a modified version of the original ACE by Tuv et al. (2009). This Tuv et al. (2009) version uses gradient boosted trees (GBT) to obtain the variable scores, whereas the one used in this analysis uses random forests. Preliminary results on the *SimOriginal* data were used to select the ACE parameters. Multiple runs suggested the selection $M = 150$ trees for each ensemble as well as a $q = q_{0.90}$ for the artificial predictors VIS. Figure 4.2 shows how the error rate in the ensemble stabilizes after including approximately 150 trees, which is what motivated the use of ensembles of 150 trees. The results also suggested that the importance scores were not normally distributed, thus, the non-parametric hypothesis Wilcox test (Wild, 2011) and a Bonferroni approach (Bland and Altman, 1995) was used to assess significance $\alpha = 0.05/(\textit{number of predictors})$.

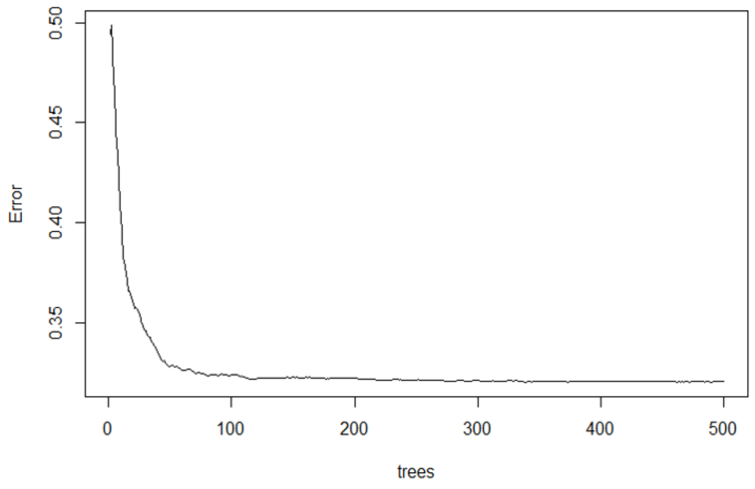


Fig. 4.2.: Preliminary evaluation of the number of trees in a parallel ensemble or random forest.

The *CFS and ReliefF* feature selection methods were implemented using the R package `FSelector` (Romanski and Kotthoff, 2016). The ReliefF method requires that the user chooses a threshold to select the top important variables. Variables were sorted in order of attribute importance and the percentage between them was calculated. The top subset was composed of all the variables whose difference was below 35%. The following parameters were used: neighbours.count = 10 and sample.size= 0.05(DatasetSize). CFS was run using its default parameters.

The *VSURF* method was implemented using the R package `VSURF` (Genuer et al., 2015b). The amount a of trees for the random forest was changed to $ntree = 125$. Finally, the *GA* algorithm was executed using the R package `caret` (Kuhn, 2016), with number of search iterations selected as $iter = 2$.

Performance Measures

Four performance measures were evaluated in the five-fold cross-validation: (1) *accuracy*, (2) *best subset size*, (3) *run time*, and (4) *overall desirability.* In classification models, the ***accuracy*** is the fraction of instances that are correctly predicted. It is given by:

$$Accuracy = 1 - \frac{f}{n} \tag{4.1}$$

where f/n is the classification error given by the amount samples incorrectly classified out of the total samples n. On the other hand, the accuracy of a regression model is also given by the prediction error. Thus, the predicted residual error sum of squares (PRESS) was calculated for those data sets with numeric response. The ***PRESS*** is given by:

$$PRESS = \sum_{i=1}^{n} (y_i - \hat{y})^2 \tag{4.2}$$

where y_i and $\hat{y}$ are the observed and predicted values, respectively. PRESS values were scaled to a range between 0 and 1 and its complement was calculated to convert them into accuracy values. Consequently, values close to one are preferred for this scaled measure.

Run time denotes the CPU time, in seconds, taken to run the algorithm. A faster FS method is desired; hence, lower run time values are preferred. The ***Best subset*** measure depicts the number of important variables selected by the FS method. Smaller subsets will lead to less complex models and, thus, smaller subsets are preferred.

Since various performance measures may favor different methods, a *desirability function* was used to determine the top performer. The overall desirability function (D) combines the previous measures and gives each one of them different weights based on their relative importance: accuracy (60%) > run time (30%) > best subset (10%).

$$D_{cv} = 0.6 \times (Accuracy) + 0.3 \times (1 - Runtime') + 0.1 \times (1 - BestSubset') \tag{4.3}$$

Note that all the performance measures were scaled between 0 and 1 before calculating the desirability function. Values closer to one are preferred, since the desirability function is scaled as well. The desirability function was scaled to values between 0 and 1 as well as:

$$D'_{cv} = \frac{D_{cv} - min(D_{cv})}{max(D_{cv}) - min(D_{cv})} \tag{4.4}$$

Furthermore, the *Sensitivity*, *Specificity* and *Accuracy* of the FS methods were also calculated for the simulated data sets since their important variables were known. The overall *Desirability function* was evaluated once more. Based on the feature selection analysis, ***sensitivity*** is the proportion of important variables for a predictor j correctly identified by the feature selection method:

$$Sensitivity = \frac{TP}{TP + FN} \tag{4.5}$$

where TP is the number of variables accurately detected as important and $TP + FN$ is the total number of important variables.

The ***specificity*** is the proportion of variables accurately not detected as important by the feature selection method:

$$Specificity = 1 - \frac{FP}{FP + TN} \tag{4.6}$$

where FP is the number of excess variables detected by the feature selection method and $FP + TN$ is the number of variables that should not have been detected as important, which are the redundant and noise variables.

The ***accuracy*** is the proportion of variables correctly identified by the feature selection method, given by:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.7}$$

where TP is the amount of important variables detected and TN are the number of non-important variables not detected out of the total number of variables in the data set.

Once again, the overall ***desirability*** combines the previous measures in Equation 4.8 and gives each one of them different weights based on their relative importance: sensitivity (50%) > specificity (25%) > accuracy (25%).

$$D_f = 0.5 \times Sensitivity + 0.25 \times Specificity + 0.25 \times Accuracy \tag{4.8}$$

The desirability function was scaled to values between 0 and 1 as well:

$$D'_f = \frac{D_f - min(D_f)}{max(D_f) - min(D_f)} \tag{4.9}$$

4.1.2 Evaluation of Random Forest Imputation

Additional experiments were carried in order to improve the random forest-based imputation performance. Specifically, various factors were taken into account for the evaluation of the stopping criteria in the imputation of the missing values in X_i. This algorithm was implemented using R statistical software as well as the random forest function available in the R package `randomForest` (Liaw and Wiener, 2014).

This analysis was performed using the *SimOriginal* data set. The performance of an imputation technique depends on the amount of missing values present in the data set. Thus, missing ratios of 5%, 10%, 15% and 20% were simulated for each of the *SimOriginal* bootstrap samples. These missing values are randomly distributed in the data sets, meaning that the x% of the total variable values are missing.

Figure 4.3 portrays the multiple factors evaluated for the imputation stopping criteria and the overall imputation. Five different factors were considered in this evaluation: missing ratio, stage, impute change, stop rule, and imputation. The stopping criteria δ is calculated for both numerical and categorical variables. In the case of numerical variables, δ_n is the percentage of difference between the new imputation of a missing value and an old imputation. This difference is compared against a threshold called *impute change* (Δ_n). The algorithm will keep imputing the missing value as long as $\delta_n > \Delta_n$.

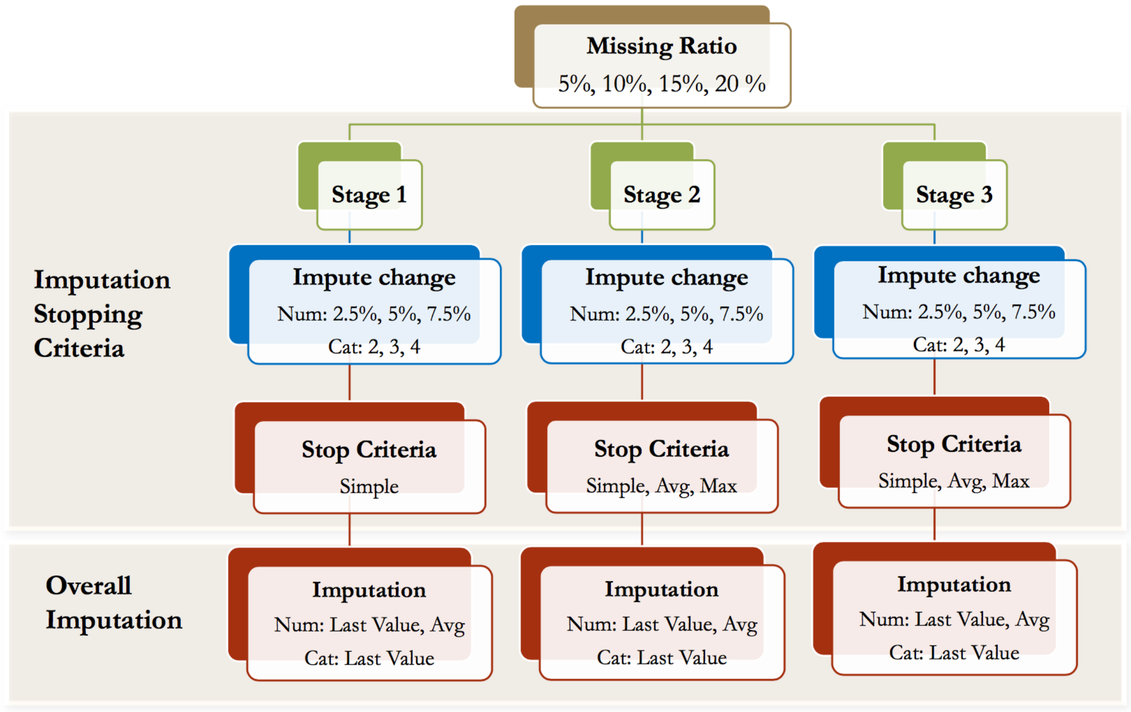


Fig. 4.3.: Flow chart of evaluated factors with their corresponding levels.

Three different values of Δ_n were evaluated: 2.5%, 5%, and 7.5%. For a given imputation in record r, the imputation difference is:

$$\delta_n = \frac{x_{new} - x_{old}}{x_{old}} \tag{4.10}$$

In the case of categorical variables, if the new imputed value stays the same in the last δ_c iterations, then, the algorithm stops imputing for that record and moves on to the next record r in X_i. Three options were also evaluated for Δ_c, where the imputation value did not: change in the last two iterations (Δ_c=2), change in the last three iterations (Δ_c=3), and did not change in the last three iterations (Δ_c=4).

Two factors were considered to decide how δ_n is calculated in the algorithm: (1) *stage* and (2) *stop rule*. Three stages were evaluated: stage 1, 2, and 3, denoting the amount of imputation differences considered for the final impute change in the iteration. These stages go along with three stopping criterias: simple, average, and maximum, which depict the aggregation of these stages to obtain the final imputation change. Table 4.1 shows all the possible combinations of *stages* and *stop rules* along with the equations used to calculate δ_n in the experiments. For example, if the combination stage 2/average is used, then, δ_n is calculated as the average of the difference between imputations j and $j-1$ and imputations $j-1$ and $j-2$.

Table 4.1: Equations used to calculate δ_n in the parameter tuning.

Stage	***Stop Criteria***	$\boldsymbol{\delta_n}$
1	Simple	$\frac{(x_j^{imp}-x_{j-1}^{imp})}{x_{j-1}^{imp}}$
2	Simple	$\frac{(x_j^{imp}-x_{j-2}^{imp})}{x_{j-2}^{imp}}$
3	Simple	$\frac{(x_j^{imp}-x_{j-3}^{imp})}{x_{j-3}^{imp}}$
2	Average	$\frac{\frac{(x_j^{imp}-x_{j-1}^{imp})}{x_{j-1}^{imp}}+\frac{(x_{j-1}^{imp}-x_{j-2}^{imp})}{x_{j-2}^{imp}}}{2}$
3	Average	$\frac{\frac{(x_j^{imp}-x_{j-1}^{imp})}{x_{j-1}^{imp}}+\frac{(x_{j-1}^{imp}-x_{j-2}^{imp})}{x_{j-2}^{imp}}+\frac{(x_{j-2}^{imp}-x_{j-3}^{imp})}{x_{j-3}^{imp}}}{3}$
2	Maximum	$Max\left\{\frac{(x_j^{imp}-x_{j-1}^{imp})}{x_{j-1}^{imp}},\frac{(x_{j-1}^{imp}-x_{j-2}^{imp})}{x_{j-2}^{imp}}\right\}$
3	Maximum	$Max\left\{\frac{(x_j^{imp}-x_{j-1}^{imp})}{x_{j-1}^{imp}},\frac{(x_{j-1}^{imp}-x_{j-2}^{imp})}{x_{j-2}^{imp}},\frac{(x_{j-2}^{imp}-x_{j-3}^{imp})}{x_{j-3}^{imp}}\right\}$

Finally, two options were evaluated for the overall imputation value of record r in numerical missing variable X_i: last value, meaning that the final imputed value is

x_j^{imp}, and average, which implies that the final imputed value es the average of the imputed values considered in the calculation of δ_n. If X_i is a categorical variable, then the final imputed value is the last value imputed (x_j^{imp}). Overall, forty two different combinations were evaluated for each missing ratio, for a total of 168 combinations.

Performance Measures

Both, regression and classification performance measures were evaluated due to the mixed-typed nature of the data sets.

Four performance measures for numerical variables were evaluated: (1) *coefficient of determination* (R^2), (2) *normalized root mean squared error* (NRMSE), (3) *index of agreement* (d_2), and (4) *overall numerical desirability* (D_n). The ***coefficient of determination*** is the square of the Pearson's product-moment correlation coefficient and describes the proportion of the total variance in the observed data that can be explained by the model (Carriquiry, 2004). The R^2 was implemented using the *gof* function of the `hydroGOF` R package (Zambrano, 2014). It ranges between 0 and 1, and the higher the value the more useful the model. R^2 is defined as follows:

$$R^2 = 1 - \frac{SSE}{SST} \tag{4.11}$$

where SSE measures the deviation of the observations (y_i) from the predicted values ($\hat{y}_i$):

$$SSE = \sum_i (y_i - \hat{y}_i)^2 \tag{4.12}$$

and SST, measures the deviations of the observations from their mean ($\bar{y}$):

$$SST = \sum_i (y_i - \bar{y})^2 \tag{4.13}$$

The ***normalized root mean square error (NRMSE)*** is a non-dimensional measure of the difference between the values predicted by a model and the values actually observed in the environment (Kaggle, 2017). It is given by:

$$NRMSE = \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)}}{var(y_i)} \tag{4.14}$$

Since NRMSE is an error measure, high values of NRMSE mean higher imputation error. Therefore, low values of NRMSE are desired. The NRMSE was also implemented using the *gof* function of the `hydroGOF` R package (Zambrano, 2014).

The ***index of agreement (*** d_2 ***)*** is a standardized measure of the degree of model prediction error as well. It is similar to R^2, but it based on the average relative error instead. Again, d_2 was implemented using the *gof* function of the `hydroGOF` R package (Zambrano, 2014). This index varies from 0 to 1, with higher values indicating better agreement between the prediction and observations (Willmott, 1981). The equation for d_2 is:

$$d_2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}\left(|\hat{y}_i - \bar{y}| + |y_i - \bar{y}|\right)^2} \tag{4.15}$$

The ***overall numerical desirability*** function combines the previous measures and treats them as equally important through an addition operation:

$$D_n = R^2 + d_2 + (1 - NRMSE) \tag{4.16}$$

Note that the complement of NRMSE is used in order to reflect that larger values of the desirability function are preferred. The desirability function was scale to values between 0 and 1 as follows:

$$D'_n = \frac{D_n - min(D_n)}{max(D_n) - min(D_n)} \tag{4.17}$$

Other four performance measures were also evaluated for categorical variables: (1) *classification error* (E), (2) *area under precision-recall curve* (AUPRC) (3) *kappa statistic* (κ), and (4) *overall categorical desirability* (D_c). The ***classification error*** is the proportion of sample cases incorrectly classified and is evaluated by:

$$E = \frac{f}{n} \tag{4.18}$$

where f is the number of sample cases incorrectly classified out of n total samples.

The ***area under the precision-recall curve*** is the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one. For multi-class problems, the AUPRC is a measure of the discriminability of multiple pair of classes. It is based on precision, which is related to what should have been detected, and recall, which is the fraction of what was detected (Fawcett, 2003). AUPRC values range between 0 and 1 as well, and the closer the value to 1, the better. The equation is given by:

$$AUPRC = \sum_{c_i \in C} AUPRC(c_i) * p(c_i) \tag{4.19}$$

where $AUPRC(c_i)$ is the area under de precision-recall curve for class i and $p(c_i)$ is a weight given based on the prevalence of the class in the data. The AUPRC was implemented using the *pr.curve* function of the `PRROC` R package (Grau and Keilwagen, 2015). It was preferred over the traditional area under the curve (AUC) because it does a better job at handling unbalanced data sets.

The ***kappa statistic*** (κ) is a measure of agreement between categorical variables. It compares an observed accuracy with an expected accuracy (random chance). In addition, it takes into account random chance (agreement with a random classifier). Kappa can take values between -1 and 1, but a $\kappa > 0$ is desired, specially values closer to one (Sharp et al., 2017):

$$\kappa = \frac{p_o - p_e}{1 - p_e} \tag{4.20}$$

where p_o is the observed agreement, and p_e is the probability of random agreement. The kappa statistic was implemented using the *cohen.kappa* function of the `psych` R package (Revelle, 2016).

Once again, the overall ***desirability*** function for the categorical variables was assessed and considered each as equally important.

$$D_c = \kappa + AUPRC + (1 - E) \tag{4.21}$$

The desirability function was scaled to values between 0 and 1 as follows:

$$D'_c = \frac{D_c - min(D_c)}{max(D_c) - min(D_c)} \tag{4.22}$$

In order to evaluate both, numerical and categorical variables at the same time, an overall desirability function (D_o) was calculated. Both are equally weighted by:

$$D_o = D'_n + D'_c \tag{4.23}$$

4.2 Evaluation of Imputation Methods

The performance of the proposed imputation method was compared to KNN and missForest. Figure 4.4 displays a general overview of the missing value imputation evaluation. Nine data sets (BreastCancer, Heart, EPR, LinReg203, LinClass145, NonLinReg70, NonLinReg125, NonLinReg38, and SimOriginal), described in Section 3.2, were used in this final evaluation. Thirty bootstrap samples were generated for each data set and missing values were randomly created for each one of them using the *ProdNA* function of the R package `missForest` (Stekhoven, 2013). These missing values were simulated at 5%, 10%, 15% and 20% missing ratios.

The KNN was implemented using the function *kNN* in the R package `VIM` (Kowarik and Templ, 2016). This algorithm uses Gower distance to find the k nearest neighbors. We used the default parameters of the function, that is, the number of nearest neighbors k=5. On the other hand, the function *missForest* was executed using the R package `missForest` (Stekhoven, 2013). Here, the default parameters of the function were used: the number of trees in the random forest (*ntrees=100*), the maximum

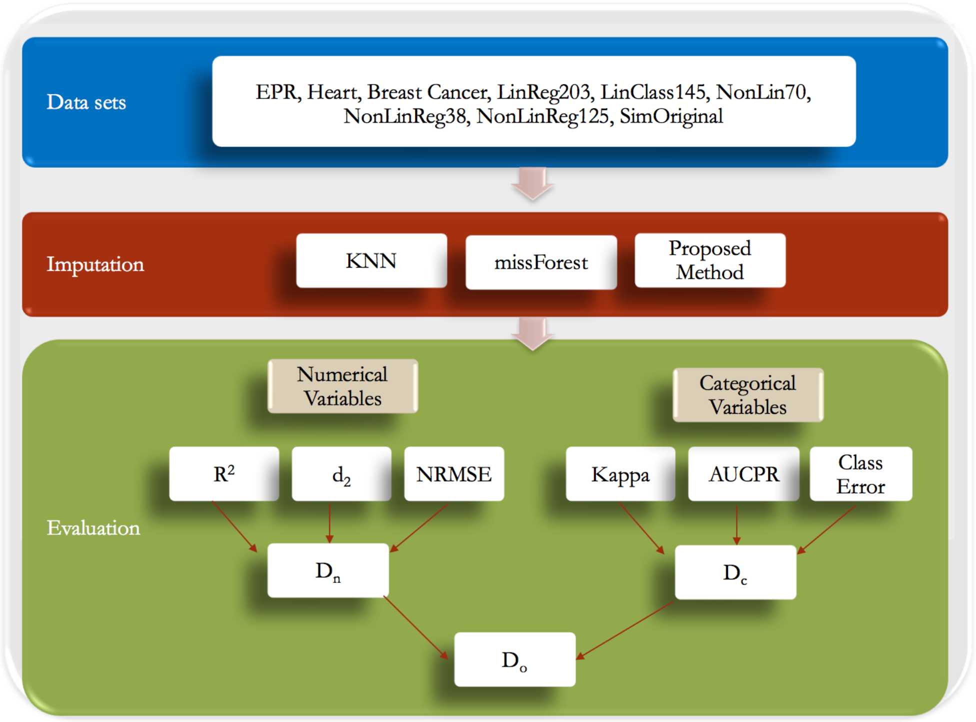


Fig. 4.4.: Flow chart of steps carried out for the performance evaluation of missing value imputation schemes.

number of iterations to be performed given the stopping criterion is not met beforehand ($maxiter$=*10*), among others.

Performance Measures

The performance measures used for the evaluation of the imputation methods are the same as the ones explained in Subsection 4.1.2.

5. RESULTS

This chapter summarizes the major results of the development of the proposed imputation scheme (Section 5.1), and the final assessment of the performance of the proposed method against KNN and missForest (Section 5.2). Detailed results for some of the analyses shown in this chapter are available in Appendix A, B, and C, respectively.

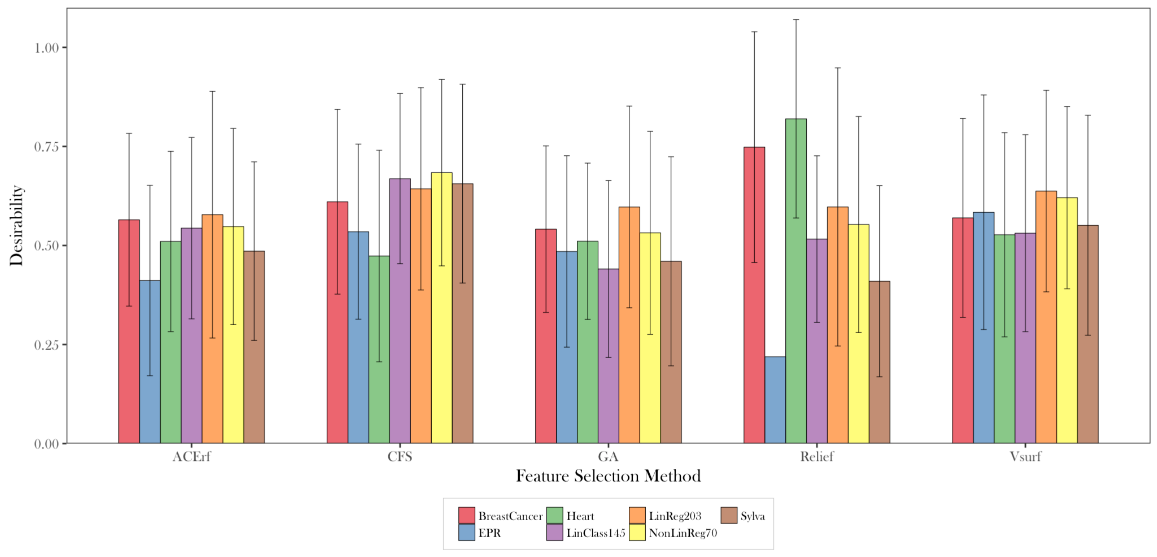
5.1 Proposed Imputation Method

5.1.1 Feature Selection

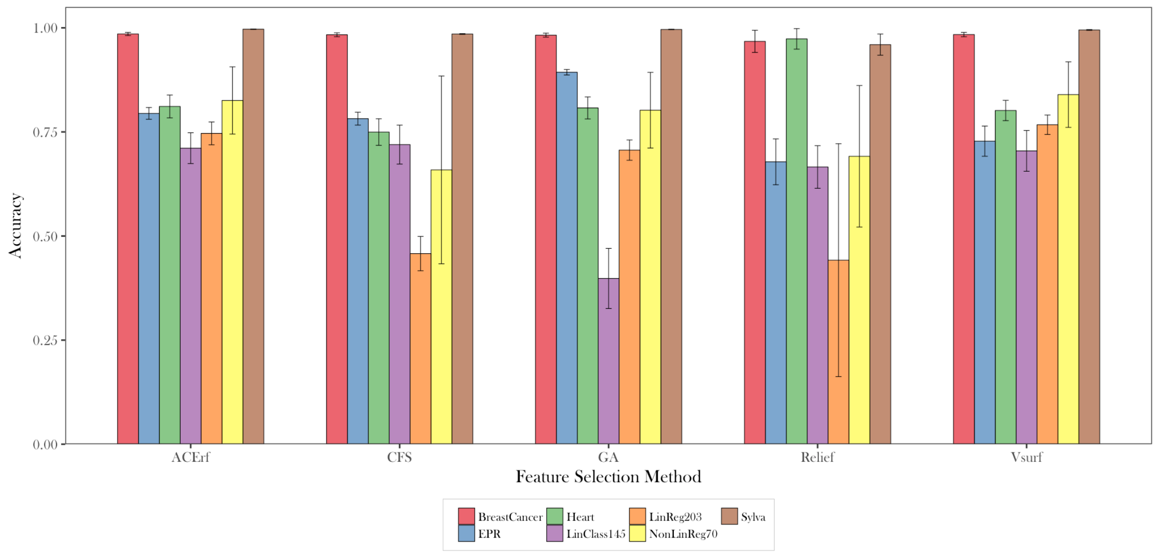
Table 5.1 shows the aggregated results of the five-fold cross-validation used in the selection of the feature selection method. Columns three through seven denote the average value of the performance metrics on each feature selection method across the seven data sets. Bold values in the D'_{cv} column represent the best results. Figure 5.1 also presents the average performance of the feature selection methods. One standard error rules are also provided for each plot. Figures 5.1(a) to 5.1(d) present the average desirability function, D_{cv}, average accuracy, average best subset size, and average run time for each feature selection method and data set evaluated, respectively. ACE has the highest average accuracy, as described in Figure 5.1(b), and CFS has the lowest average best subset size and average run time, as depicted by Figures 5.1(c) and 5.1(d).

Table 5.1: Feature selection five-fold cross-validation results.

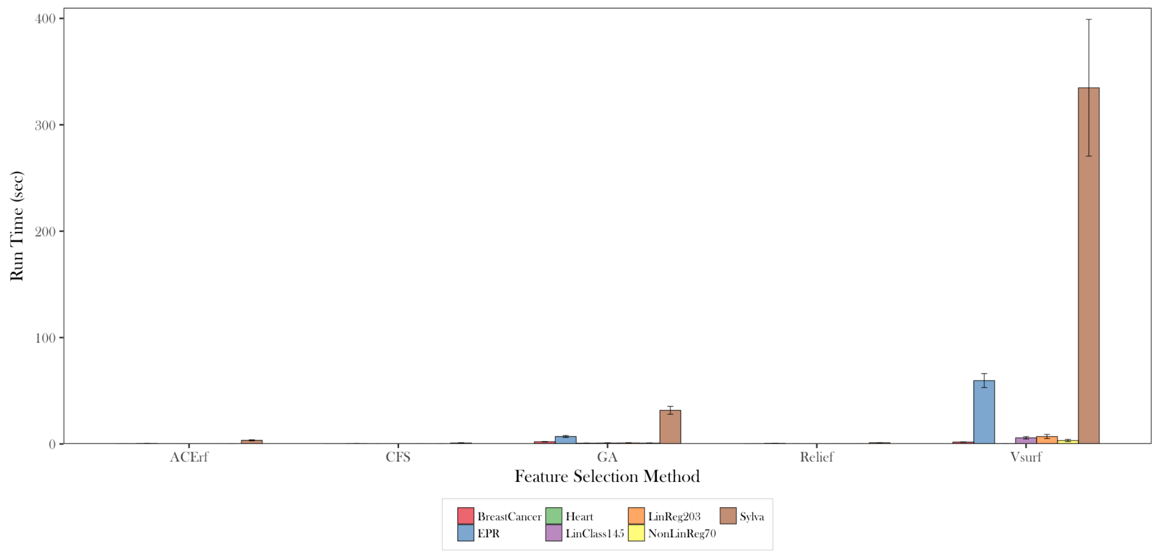
Data set	***FS Method***	***CV Accuracy***	***Run time***	***Best subset***	D_{cv}	D'_{cv}
BreastCancer	CFS	0.9834	0.0013	8	0.6103	0.3337
	GA	0.9824	2.1136	7	0.5413	0.0000
	ACE	0.9853	0.0883	10	0.5648	0.1137
	ReliefF	0.9675	0.0603	6	0.7483	**1.0000**
	VSURF	0.9839	1.7940	6	0.5696	0.1370
EPR	CFS	0.7819	0.2680	11	0.5347	0.8653
	GA	0.8934	7.0413	70	0.4849	0.7286
	ACE	0.7945	0.3943	28	0.4114	0.5271
	ReliefF	0.6782	0.5435	10	0.2193	0.0000
	VSURF	0.7278	59.5774	12	0.5838	**1.0000**
Heart	CFS	0.7497	0.0220	5	0.4734	0.0000
	GA	0.8077	0.7780	10	0.5106	0.1076
	ACE	0.8112	0.0057	8	0.5101	0.1062
	ReliefF	0.9735	0.0343	7	0.8197	**1.0000**
	VSURF	0.8016	0.1663	7	0.5270	0.1550
LinReg203	CFS	0.4579	0.0350	2	0.6431	**1.0000**
	GA	0.7064	1.0043	86	0.5971	0.2961
	ACE	0.7465	0.0023	73	0.5778	0.0000
	ReliefF	0.4420	0.0613	78	0.5973	0.2986
	VSURF	0.7673	7.0063	6	0.6373	0.9108
LinClass145	CFS	0.7196	0.0300	9	0.6687	**1.0000**
	GA	0.3981	0.9180	85	0.4406	0.0000
	ACE	0.7110	0.0463	29	0.5437	0.4521
	ReliefF	0.6658	0.0427	52	0.5161	0.3308
	VSURF	0.7045	5.8063	7	0.5311	0.3967
NonLinReg70	CFS	0.6588	0.0253	5	0.6838	**1.0000**
	GA	0.8022	0.8507	38	0.5319	0.0000
	ACE	0.8256	0.0243	22	0.5478	0.1045
	ReliefF	0.6915	0.0683	17	0.5528	0.1377
	VSURF	0.8396	3.2400	7	0.6206	0.5839
Sylva	CFS	0.9852	0.9700	7	0.6560	**1.0000**
	GA	0.9961	31.6630	147	0.4601	0.2047
	ACE	0.9967	3.4700	59	0.4858	0.3089
	ReliefF	0.9597	1.1637	18	0.4097	0.0000
	VSURF	0.9949	334.8100	8	0.5509	0.5735



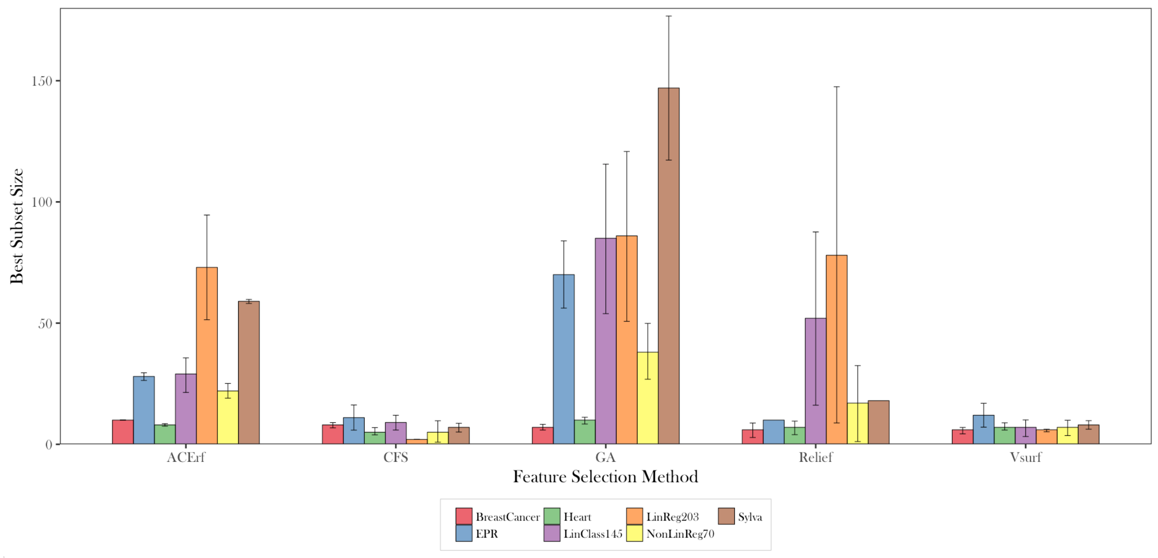
(a) Average desirability function, including one standard error bars, on Breast Cancer, EPR, Heart, LinClass145, LinReg203, NonLinReg70, and Sylva data sets.



(b) Average accuracy, including one standard error bars, on Breast Cancer, EPR, Heart, LinClass145, LinReg203, NonLinReg70, and Sylva data sets.



(c) Average run time, including one standard error bars, on Breast Cancer, EPR, Heart, LinClass145, LinReg203, NonLinReg70, and Sylva data sets.



(d) Average best subset size, including one standard error bars, on Breast Cancer, EPR, Heart, LinClass145, LinReg203, NonLinReg70, and Sylva data sets.

Fig. 5.1.: Five-fold cross-validation performance of feature selection methods.

Overall, CFS performed the best in four out of seven data sets (or 57% of the evaluated cases). Table 5.2 portrays the D'_{cv} score, which is given by the sum of the normalized desirability function through the data sets, being CFS the one with highest result.

Table 5.2: Normalized cross-validation desirability score across all data sets.

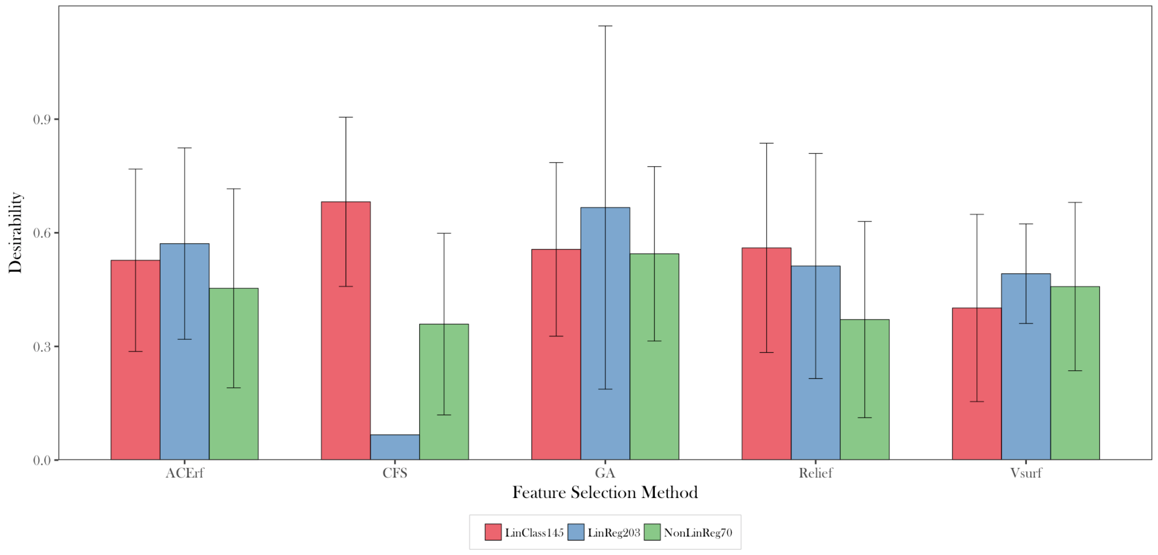
FS Method	***D'_{cv} Score***
ACE	1.6125
CFS	**5.1990**
GA	1.3369
ReliefF	2.7671
VSURF	3.7569

Additionally, the three simulated scenarios (LinReg203, LinClass145 and NonLinReg70) were evaluated in more detail since their structure and important variables were known. Table 5.3 shows the results of this analysis, where bold values in the D'_f column denote the best results.

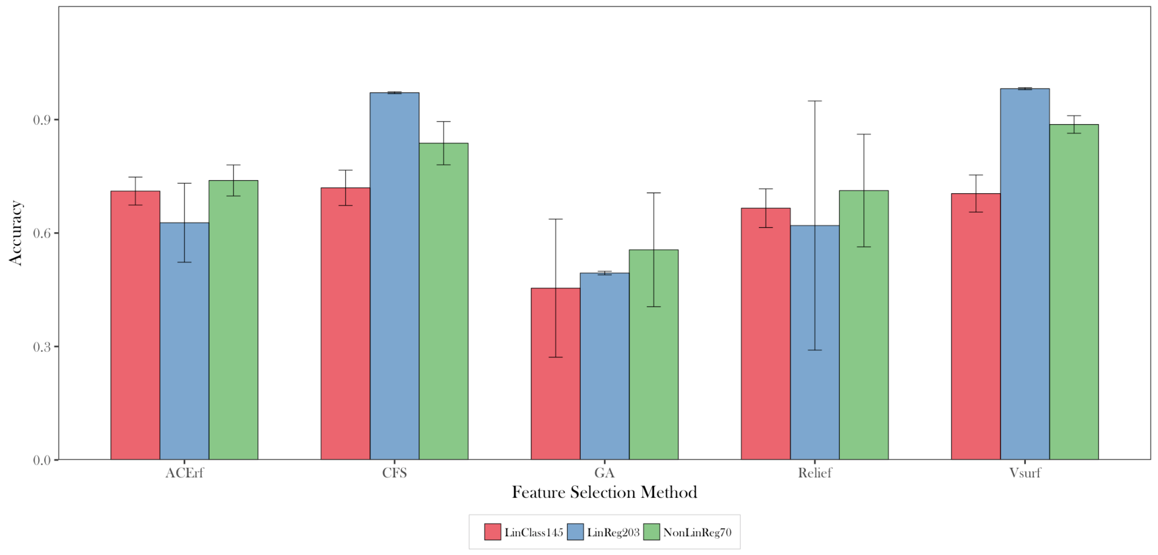
Table 5.3: Feature selection performance on simulated scenarios.

Data set	***FS Method***	***Sensitivity***	***Specificity***	***Accuracy***	***D_f***	***D'_f***
LinClass145	ACE	0.4467	0.8321	0.7110	0.5275	0.4494
	CFS	0.4644	0.9849	0.7196	0.6818	**1.0000**
	GA	0.6867	0.4274	0.4543	0.5562	0.5521
	ReliefF	0.4956	0.6582	0.6658	0.5602	0.5662
	VSURF	0.1978	0.9718	0.7045	0.4015	0.0000
LinReg203	ACE	0.1667	0.6365	0.6273	0.5714	0.8412
	CFS	0.0167	0.9903	0.9711	0.0667	0.0000
	GA	0.1667	0.5008	0.4943	0.6667	**1.0000**
	ReliefF	0.6250	0.6198	0.6199	0.5123	0.7428
	VSURF	0.7500	0.9864	0.9818	0.4920	0.7090
NonLinReg70	ACE	0.6900	0.7472	0.7390	0.4535	0.5090
	CFS	0.1967	0.9444	0.8376	0.3589	0.0000
	GA	0.8633	0.5044	0.5557	0.5446	**1.0000**
	ReliefF	0.3333	0.7756	0.7124	0.3710	0.0649
	VSURF	0.4433	0.9611	0.8871	0.4580	0.5337

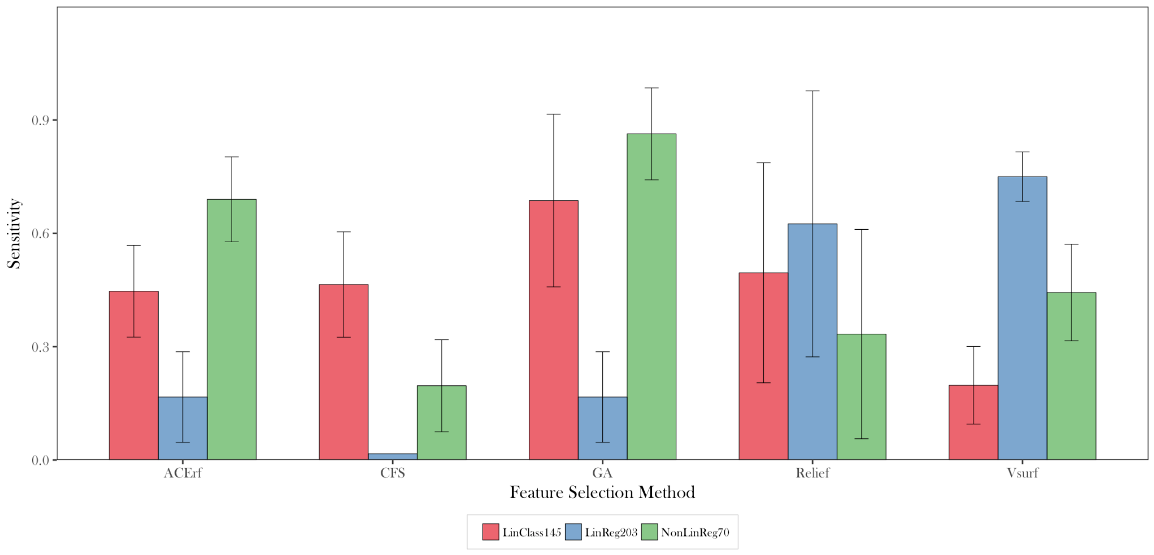
Figure 5.2 also shows the performance results of this analysis. Figures 5.2(a) to 5.2(d) displays the average desirability function (D_f), average sensitivity, average specificity, and average accuracy of each feature selection on the simulated scenarios; including one standard error bars as well.



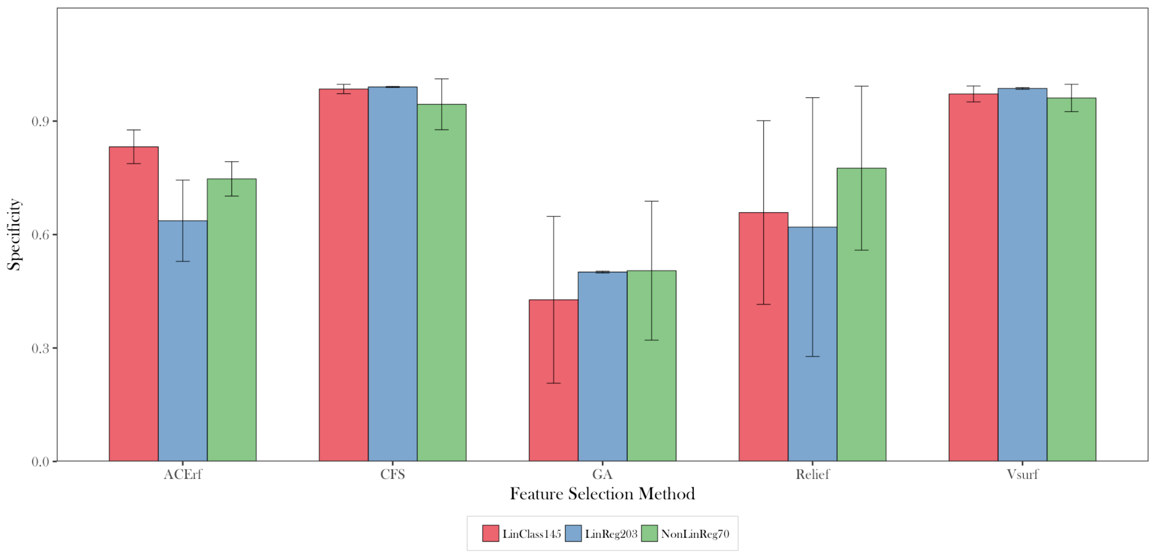
(a) Average desirability funtion, including error bars, of feature selection methods on LinClass145, LinReg203, NonLinReg70 data sets.



(b) Average accuracy, including error bars, of feature selection methods on LinClass145, LinReg203, NonLinReg70 data sets.



(c) Average sensitivity, including error bars, of feature selection methods on LinClass145, LinReg203, NonLinReg70 data sets.



(d) Average specificity, including error bars, of feature selection methods on LinClass145, LinReg203, NonLinReg70 data sets.

Fig. 5.2.: Performance of feature selection methods on simulated scenarios.

In this evaluation, GA performed better in two of the data sets, out of three (or 66% of the evaluated cases), as depicted by the overall desirability D'_f. Table 5.4 also confirms this result. It portrays the D'_f score, which once again is the sum of the normalized desirability, D'_f, through the three data sets; being GA the one with the highest score. Note that the issue here is failing to detect important variables since the proposed scheme imputes missing variables based on its important variables, selected by the feature selection method. Selecting variables in excess might not have much of an effect on the overall execution of the imputation scheme.

Table 5.4: Normalized desirability score across all data sets.

FS Method	***D'_f Score***
ACE	1.7996
CFS	1.0000
GA	**2.5521**
ReliefF	1.3738
VSURF	1.2427

5.1.2 Evaluation of Random Forest imputation

The parameter tuning of the random forest imputation was carried out using both CFS and GA, to evaluate which feature selection method in fact helps the imputation perform better. Table 5.5 shows the top 3 combinations of parameters for each missing ratio and feature selection method. Bold values in the column D'_o represent the best results. Figure 5.3 also portrays the top 3 performing combinations of parameters (using both GA and CFS) at each missing ratio in terms of the overall desirability function (D_o). Appendix A includes additional information on the the individual performance measures. The majority of the D_o results in Figure 5.3 fall in the lower boxes of the plot, specifically in the lower left. This corner includes for the most part, restrictive assumptions with regards of the impute difference calculation of numerical variables (stage 3) and its threshold ($\Delta_n = 2.5\%$). It can be seen that restrictive combinations of parameters perform better with higher ratios of missing values.

Table 5.5: Best performing combinations of parameters per missing ratio for GA and CFS.

Missing Ratio (%)	***FS Method***	***Impute Change (cat)***	***Impute Change (num)***	***Stage***	***Stop Rule***	***Imputation (cat)***	***Imputation (num)***	D_o	D'_o
5	GA	2	7.5	1	Simple	LastValue	LastValue	1.2649	**1.000**
		3	5	3	Simple	LastValue	LastValue	1.2604	0.9769
		2	7.5	1	Simple	LastValue	Avg	1.2603	0.964
	CFS	2	7.5	1	Simple	LastValue	Avg	1.0685	0.0099
		2	7.5	1	Simple	LastValue	LastValue	1.0669	0.0015
		4	2.5	1	Simple	LastValue	LastValue	1.0666	0.0000
10	GA	3	5	3	Simple	LastValue	Avg	1.2771	**1.0000**
		3	5	3	Simple	LastValue	LastValue	1.2763	0.9772
		2	7.5	1	Simple	LastValue	Avg	1.2761	0.9723
	CFS	2	7.5	1	Simple	LastValue	Avg	1.2417	0.0429
		3	5	1	Simple	LastValue	Avg	1.2409	0.0217
		3	5	3	Simple	LastValue	Avg	1.2401	0.0000
15	CFS	4	2.5	2	Simple	LastValue	Avg	1.2139	**1.0000**
		4	2.5	2	Simple	LastValue	LastValue	1.2135	0.9623
		4	2.5	3	Simple	LastValue	Avg	1.2123	0.8326
	GA	4	2.5	2	Simple	LastValue	LastValue	1.2110	0.6869
		4	2.5	2	Simple	LastValue	Avg	1.2069	0.2334
		4	2.5	3	Simple	LastValue	Avg	1.2048	0.0000
20	GA	4	2.5	3	Simple	LastValue	Avg	1.0942	**1.0000**
		4	2.5	3	Simple	LastValue	LastValue	1.0941	0.9972
		2	7.5	3	Simple	LastValue	Avg	1.0832	0.7308
	CFS	4	2.5	3	Simple	LastValue	Avg	1.0645	0.2764
		4	2.5	3	Simple	LastValue	LastValue	1.0626	0.2290
		2	7.5	3	Simple	LastValue	Avg	1.0531	0.0000

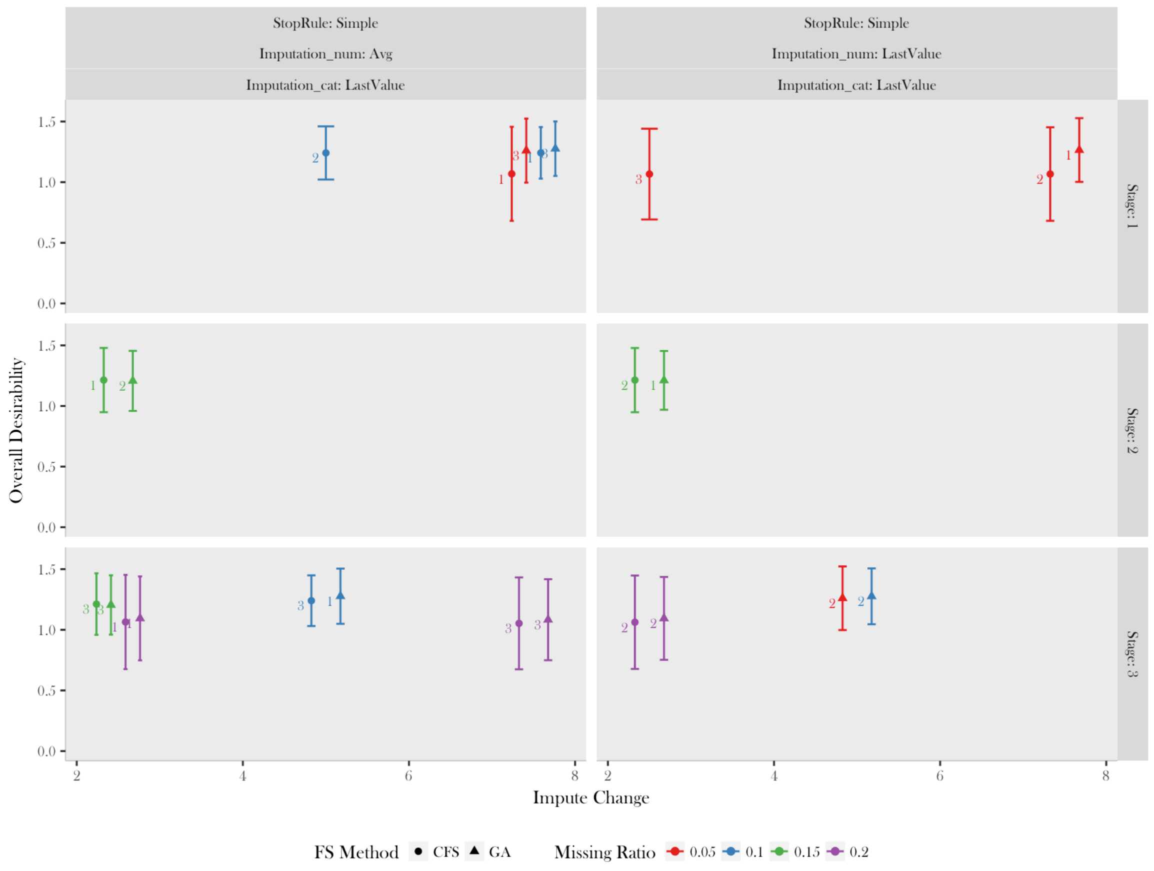


Fig. 5.3.: Top 3 performing combinations for each missing ratio and feature selection method. Triangular and circular shape points denote GA and CFS values, respectively.

In summary, the results in Table 5.5 confirm GA as the best feature selection method choice, giving higher values of overall desirability (D_o) in three out of four missing ratios (or 75% of the cases). Furthermore, the selected combinations of parameters for the proposed random-forest-based imputation is given in Table 5.6. This combination was chosen based on the premise that the proposed method is mostly focused on data with medium to high proportions of missing values. Thus, the proposed imputation scheme used GA as feature selection method along with the random-forest-based imputation having $\Delta_n = 2.5\%$, $\Delta_c = 4$, $\delta_n = \frac{x_j^{imp} - x_{j-3}^{imp}}{x_{j-3}^{imp}}$, with $x_n^{imp} = \frac{x_{j-3}^{imp} + x_j^{imp}}{2}$ being the final imputation of numerical missing values and $x_c^{imp} = x_j^{imp}$ the final imputation of categorical missing values.

Table 5.6: Selected combination of parameters for the proposed random forest-based imputation.

Parameter	***Suggested Value/Setting***
Impute change num	2.5
Impute change cat	4
Stage	3
Stop Rule	Simple
Imputation num	Avg
Imputation cat	Last Value

5.2 Evaluation of Imputation Methods

Table 5.7 shows the average performance of the three imputation methods in terms of D_n, D_c, and D_o. Bold values in columns D'_n, D'_c, and D'_o represent the best results for each data set and missing ratio. Figures 5.4 to 5.6 also portray D_n, D_c, and D_o results including one standard error bars. Appendix B includes individual plots of the categorical and numerical performance measures.

Table 5.7: Average performance of imputation methods. Note that the breast cancer data set is fully categorical, therefore *NA* values where placed in D_n and D_o. Also note that the NonLinReg70 data set is fully numerical, therefore *NA* values where placed in D_c and D_o.

Data set	***Missing Ratio (%)***	***Imputation Method***	D_n	D'_n	D_c	D'_c	D_o	D'_o
Breast Cancer	5	KNN	NA	NA	0.6929	0.6741	NA	NA
		Proposed Method	NA	NA	0.3587	0.0000	NA	NA
		missForest	NA	NA	0.8544	**1.0000**	NA	NA
	10	KNN	NA	NA	0.6861	0.6326	NA	NA
		Proposed Method	NA	NA	0.3664	0.0000	NA	NA
		missForest	NA	NA	0.8718	**1.0000**	NA	NA
	15	KNN	NA	NA	0.6780	0.6279	NA	NA
		Proposed Method	NA	NA	0.3717	0.0000	NA	NA
		missForest	NA	NA	0.8595	**1.0000**	NA	NA
	20	KNN	NA	NA	0.6696	0.6060	NA	NA
		Proposed Method	NA	NA	0.3977	0.0000	NA	NA
		missForest	NA	NA	0.8464	**1.0000**	NA	NA
EPR	5	KNN	0.8309	0.7920	0.0452	0.3482	0.8761	0.8225
		Proposed Method	0.5203	0.0000	0.0909	**1.0000**	0.6112	0.0000
		missForest	0.9124	**1.0000**	0.0209	0.0000	0.9333	**1.0000**
	10	KNN	0.8128	0.7861	0.0459	0.4211	0.8587	0.8223
		Proposed Method	0.4879	0.0000	0.0814	**1.0000**	0.5694	0.0000
		missForest	0.9011	**1.0000**	0.0201	0.0000	0.9212	**1.0000**
	15	KNN	0.7988	0.7585	0.0485	0.3579	0.8473	0.7809
		Proposed Method	0.4809	0.0000	0.0920	**1.0000**	0.5729	0.0000
		missForest	0.9001	**1.0000**	0.0242	0.0000	0.9242	**1.0000**
	20	KNN	0.7848	0.7602	0.0494	0.4571	0.8342	0.7918
		Proposed Method	0.4651	0.0000	0.0784	**1.0000**	0.5435	0.0000
		missForest	0.8856	**1.0000**	0.0250	0.0000	0.9106	**1.0000**

continued on next page

continued from previous page

Data set	***Missing Ratio (%)***	***Imputation Method***	D_n	D'_n	D_c	D'_c	D_o	D'_o
Heart	5	KNN	0.8939	0.7760	0.2746	**1.0000**	1.1685	0.8134
		Proposed Method	0.7203	0.0000	0.2474	0.0000	0.9677	0.0000
		missForest	0.9441	**1.0000**	0.2706	0.8512	1.2146	**1.0000**
	10	KNN	0.8892	0.7894	0.2628	**1.0000**	1.1519	0.8097
		Proposed Method	0.7004	0.0000	0.2525	0.0000	0.9530	0.0000
		missForest	0.9395	**1.0000**	0.2592	0.6488	1.1987	**1.0000**
	15	KNN	0.8933	0.8279	0.2676	0.9380	1.1609	0.8441
		Proposed Method	0.6900	0.0000	0.2279	0.0000	0.9179	0.0000
		missForest	0.9356	**1.0000**	0.2702	**1.0000**	1.2058	**1.0000**
	20	KNN	0.8833	0.8143	0.2594	0.0000	1.1426	0.7576
		Proposed Method	0.7123	0.0000	0.3084	**1.0000**	1.0207	0.0000
		missForest	0.9223	**1.0000**	0.2594	0.0005	1.1816	**1.0000**
LinClass145	5	KNN	0.2494	0.0000	0.3690	0.3401	0.6183	0.0000
		Proposed Method	0.4371	**1.0000**	0.3639	0.0000	0.8010	**1.0000**
		missForest	0.4219	0.9186	0.3789	**1.0000**	0.8007	0.9983
	10	KNN	0.2226	0.0000	0.3692	0.0000	0.5918	0.0000
		Proposed Method	0.4364	**1.0000**	0.3749	0.9839	0.8112	**1.0000**
		missForest	0.4171	0.9097	0.3750	**1.0000**	0.7920	0.9125
	15	KNN	0.2035	0.0000	0.3685	0.7293	0.5720	0.0000
		Proposed Method	0.3913	0.8789	0.3550	0.0000	0.7464	0.7969
		missForest	0.4172	**1.0000**	0.3735	**1.0000**	0.7908	**1.0000**
	20	KNN	0.1872	0.0000	0.3658	0.8117	0.5530	0.0000
		Proposed Method	0.3539	0.7429	0.3459	0.0000	0.6999	0.6413
		missForest	0.4117	**1.0000**	0.3704	**1.0000**	0.7821	**1.0000**

continued on next page

continued from previous page

Data set	***Missing Ratio (%)***	***Imputation Method***	D_n	D'_n	D_c	D'_c	D_o	D'_o
LinReg203	5	KNN	0.6192	0.0000	0.3542	0.0000	0.9734	0.0000
		Proposed Method	0.6391	0.3599	0.3564	0.1017	0.9955	0.2878
		missForest	0.6744	**1.0000**	0.3756	**1.0000**	1.0501	**1.0000**
	10	KNN	0.6125	0.0000	0.3562	0.0000	0.9687	0.0000
		Proposed Method	0.6196	0.1203	0.3639	0.8488	0.9836	0.2173
		missForest	0.6717	**1.0000**	0.3653	**1.0000**	1.0370	**1.0000**
	15	KNN	0.6051	0.0000	0.3547	0.0000	0.9598	0.0000
		Proposed Method	0.6458	0.6431	0.3690	**1.0000**	1.0149	0.8282
		missForest	0.6684	**1.0000**	0.3579	0.2243	1.0263	**1.0000**
	20	KNN	0.6024	0.0000	0.3554	0.8824	0.9578	0.0000
		Proposed Method	0.6830	**1.0000**	0.3208	0.0000	1.0038	0.6309
		missForest	0.6707	0.8469	0.3601	**1.0000**	1.0307	**1.0000**
NonLinReg125	5	KNN	0.3785	0.0000	0.3618	0.0000	0.7404	0.0000
		Proposed Method	0.5653	**1.0000**	0.3643	0.1301	0.9296	**1.0000**
		missForest	0.5406	0.8681	0.3807	**1.0000**	0.9213	0.9564
	10	KNN	0.3365	0.0000	0.3661	0.6330	0.7026	0.0000
		Proposed Method	0.5254	0.9386	0.3557	0.0000	0.8811	0.8612
		missForest	0.5378	**1.0000**	0.3721	**1.0000**	0.9099	**1.0000**
	15	KNN	0.3049	0.0000	0.3627	0.0000	0.6676	0.0000
		Proposed Method	0.5261	0.9848	0.3934	**1.0000**	0.9195	**1.0000**
		missForest	0.5295	**1.0000**	0.3708	0.2631	0.9003	0.9238
	20	KNN	0.2738	0.0000	0.3619	0.0000	0.6357	0.0000
		Proposed Method	0.4702	0.7869	0.3753	**1.0000**	0.8455	0.8189
		missForest	0.5234	**1.0000**	0.3685	0.4924	0.8919	**1.0000**

continued on next page

continued from previous page

Data set	***Missing Ratio (%)***	***Imputation Method***	D_n	D'_n	D_c	D'_c	D_o	D'_o
NonLinReg38	5	KNN	0.4546	0.0000	0.3873	**1.0000**	0.8420	0.0000
		Proposed Method	0.5851	0.8830	0.3744	0.0000	0.9594	0.8004
		missForest	0.6023	**1.0000**	0.3864	0.9286	0.9887	**1.0000**
	10	KNN	0.4142	0.0000	0.3865	0.0000	0.8007	0.0000
		Proposed Method	0.5448	0.7181	0.3896	**1.0000**	0.9344	0.7305
		missForest	0.5961	**1.0000**	0.3876	0.3748	0.9837	**1.0000**
	15	KNN	0.3773	0.0000	0.3856	0.7529	0.7629	0.0000
		Proposed Method	0.5616	0.8717	0.3776	0.0000	0.9392	0.8237
		missForest	0.5888	**1.0000**	0.3882	**1.0000**	0.9770	**1.0000**
	20	KNN	0.3501	0.0000	0.3885	**1.0000**	0.7386	0.0000
		Proposed Method	0.5365	0.7999	0.3831	0.0000	0.9196	0.7801
		missForest	0.5832	**1.0000**	0.3875	0.8124	0.9707	**1.0000**
NonLinReg70	5	KNN	0.4410	0.0000	NA	NA	NA	NA
		Proposed Method	0.6177	**1.0000**	NA	NA	NA	NA
		missForest	0.6142	0.9804	NA	NA	NA	NA
	10	KNN	0.4067	0.0000	NA	NA	NA	NA
		Proposed Method	0.5972	0.9310	NA	NA	NA	NA
		missForest	0.6113	**1.0000**	NA	NA	NA	NA
	15	KNN	0.3842	0.0000	NA	NA	NA	NA
		Proposed Method	0.5833	0.8956	NA	NA	NA	NA
		missForest	0.6065	**1.0000**	NA	NA	NA	NA
	20	KNN	0.3689	0.0000	NA	NA	NA	NA
		Proposed Method	0.5655	0.8496	NA	NA	NA	NA
		missForest	0.6003	**1.0000**	NA	NA	NA	NA

continued on next page

continued from previous page

Data set	***Missing Ratio (%)***	***Imputation Method***	D_n	D'_n	D_c	D'_c	D_o	D'_o
		KNN	0.0773	0.0000	0.3136	0.0000	0.3910	0.0000
	5	Proposed Method	0.9729	**1.0000**	0.3632	**1.0000**	1.3361	**1.0000**
		missForest	0.2271	0.1672	0.3345	0.4217	0.5616	0.1805
		KNN	0.8739	0.0892	0.3680	0.5006	1.2419	0.0000
	10	Proposed Method	0.9615	**1.0000**	0.3505	0.0000	1.3121	**1.0000**
SimOriginal		missForest	0.8653	0.0000	0.3854	**1.0000**	1.2507	0.1256
		KNN	0.0970	0.0000	0.3232	0.0000	0.4203	0.0000
	15	Proposed Method	0.9401	**1.0000**	0.3770	**1.0000**	1.3171	**1.0000**
		missForest	0.2240	0.1505	0.3390	0.2927	0.5629	0.1591
		KNN	0.8040	0.0000	0.3737	0.1929	1.1778	0.0000
	20	Proposed Method	0.9189	**1.0000**	0.3680	0.0000	1.2869	**1.0000**
		missForest	0.8611	0.4971	0.3974	**1.0000**	1.2586	0.7405

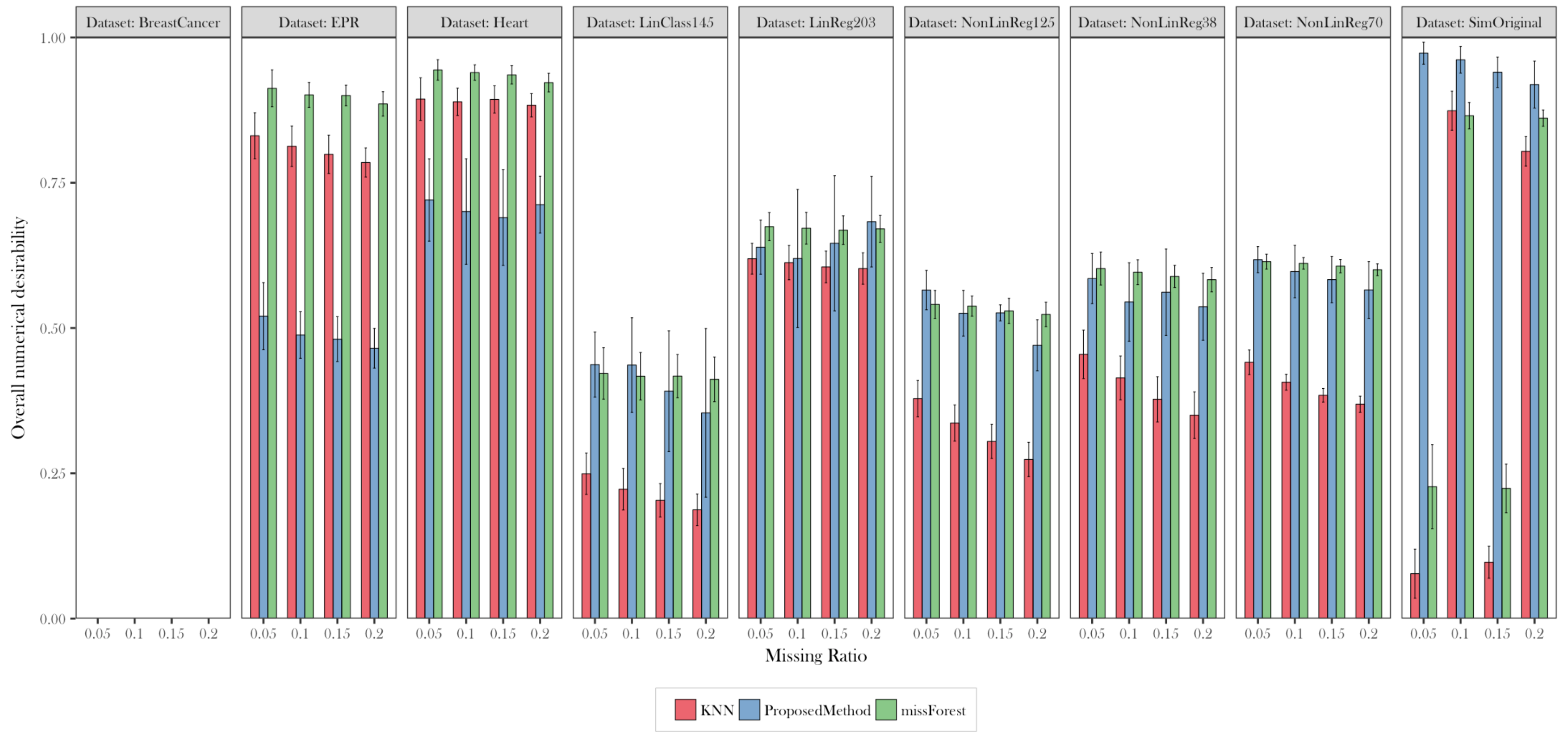


Fig. 5.4.: Average numerical desirability of imputation methods. Please note that results are not available for BreastCancer data set since it only has categorical variables.

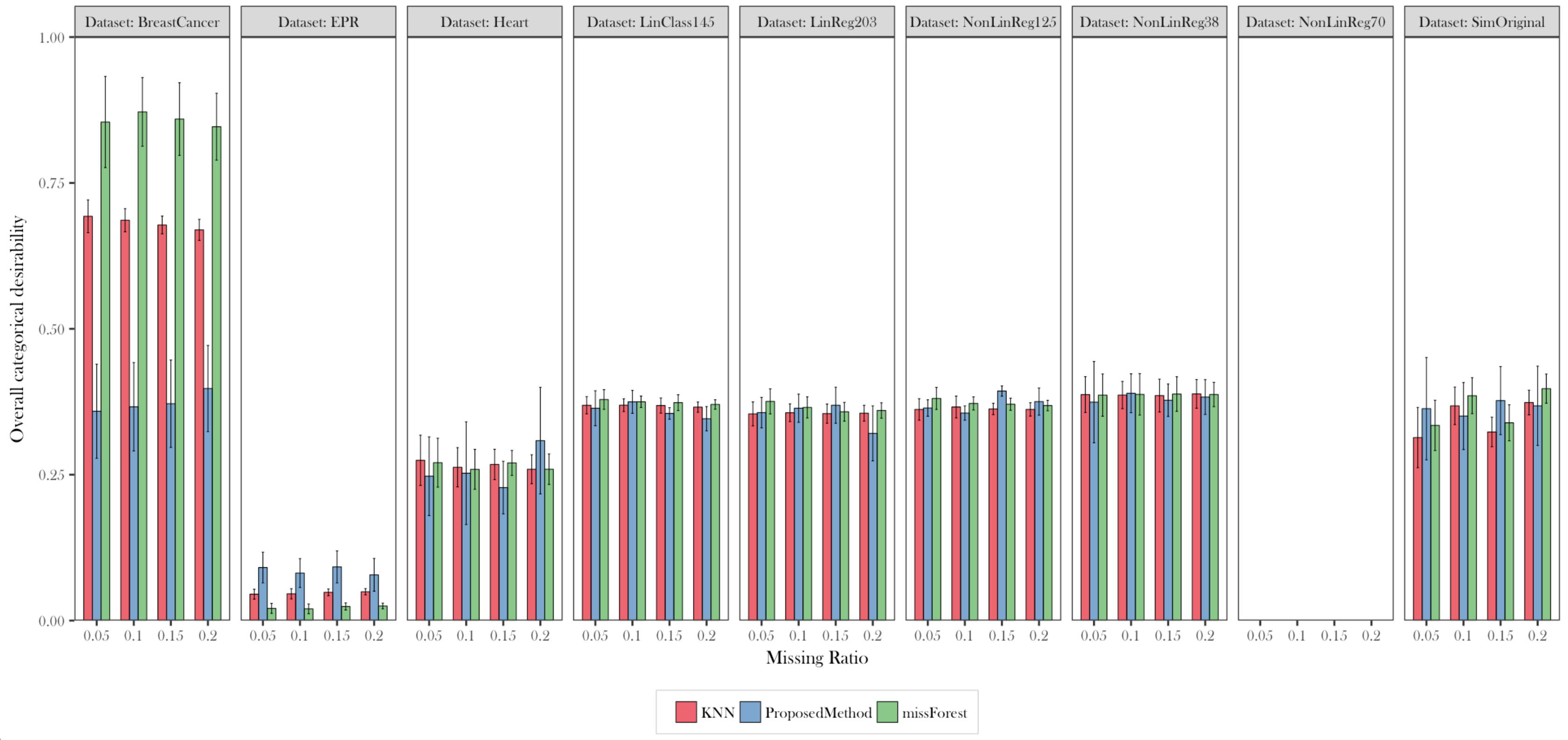


Fig. 5.5.: Average categorical desirability of imputation methods. Please note that results are not available for NonLinReg70 data set since it only has numerical variables.

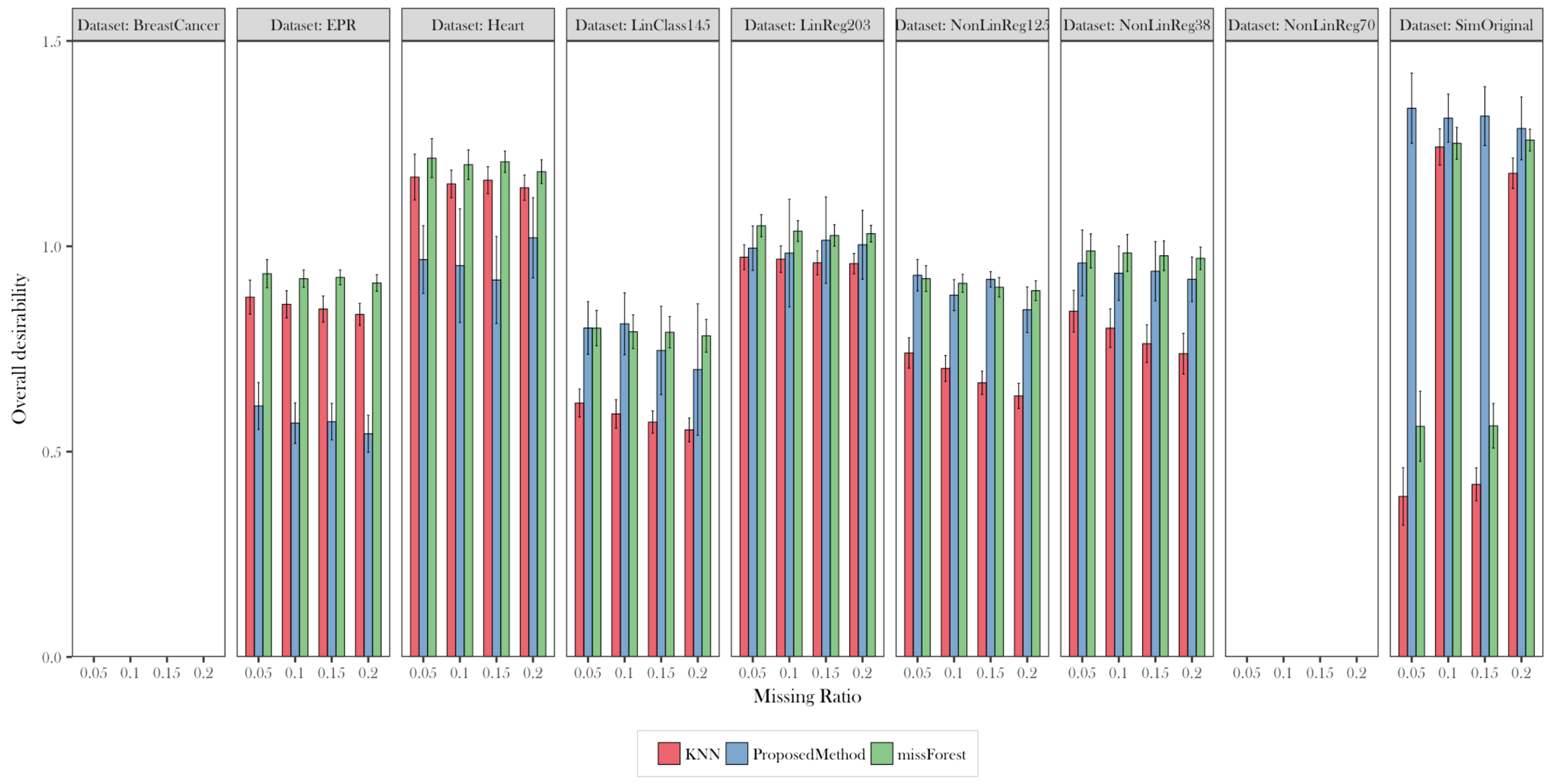


Fig. 5.6.: Average overall desirability of imputation methods. Please note that results are not available for BreastCancer and NonLinReg70 data sets since they all have categorical or numerical variables, respectively.

6. CONCLUSIONS AND FUTURE WORK

Properly imputed data gives the opportunity to retrieve, not only the best possible predictions for the missing values, but to replace them by reliable values. The goal of any successful missing value imputation scheme is to exploit the information in the incomplete cases and effectively develop approaches to better understand the underlying populations described in these data sets.

The aim of this study was to design a random-forest-based missing value imputation technique that would take into account the relationships among variables. The final proposed imputation scheme uses GA feature selection to get these relationships, along with the random-forest-based imputation. The proposed method was intended to be used for high-dimensional, mixed-type data with high volume of missing values. The scenarios evaluated included data sets from low dimensionality, ten variables, to high dimensionality, two hundred variables.

This work also evaluated the performance of the proposed method against two of the best missing value imputation techniques in the literature for high-dimensional, mixed-typed data: KNN and missForest. The former uses a clustering approach to imputation; the latter uses a supervised learning approach based on parallel ensembles of decision trees or random forests.

Overall, results show that the proposed method outperforms KNN in the simulated scenarios, which have complex linear and non-linear relationships. Also, the proposed method was the top performer in the SimOriginal data set. The SimOriginal data set has additive and multiplicative relationships between variables and non-linear relationships between categorical and numerical variables. A reason why the proposed method resulted to be the top performer in the SimOriginal data set could be due to the random-forest imputation tuning being performed using this data set only. This is just an speculation, the parameter tuning would have to be evaluated using other data sets to verify this. The results also suggest that the proposed method has better performance imputing categorical variables, specially at higher missing ratios, having lower classification error than KNN and missForest.

In general, the missForest method was the top performer, however, the proposed method still a reasonable approximation if considered that it did not use more than 50% (in average) of the total predictors to carry out the imputations, whereas missForest did use them all. In fact, the proposed method used between 3% to 10% of the variables in six out of the nine data sets evaluated. This is the main difference between missForest and the proposed method. missForest is a multivariate imputation method that uses all the variables in the data set to impute a missing variable, whereas the proposed method is a univariate imputation that uses only those variables important to the missing variable. This is more advantageous in high-dimensional data since it significantly reduces the dimensionality of the problem (the amount of preliminary imputations).

The biggest limitation of the proposed imputation scheme is its computational complexity. This due to the feature selection method and to the imputation being carried out per individual variable. It is important to note that the missForest algorithm is parallelized and, thus, there is no way to compete with it until the proposed scheme is parallelized. As of now, the proposed imputation method is 52% slower than missForest. As future improvements, it is also suggested to evaluate other imputation mechanisms for the initial guess of the data (e.g. a random forest-based imputation). This has a direct impact on the performance of the feature selection method and therefore, in the imputation.

Finally, it is also suggested to explore the randomization of the order in which variables are imputed as well as the order in which the missing values are imputed within the variables. Right now, the proposed scheme imputes the missing variables in ascending order of missing values and the imputed values are substituted in the original data set to impute the new values. This means that performance of the imputation increases as more variables are imputed, therefore, variables that are imputed last are better imputed than the ones imputed at the beginning.

A. PARAMETER TUNNING OF RANDOM FOREST-BASED IMPUTATION

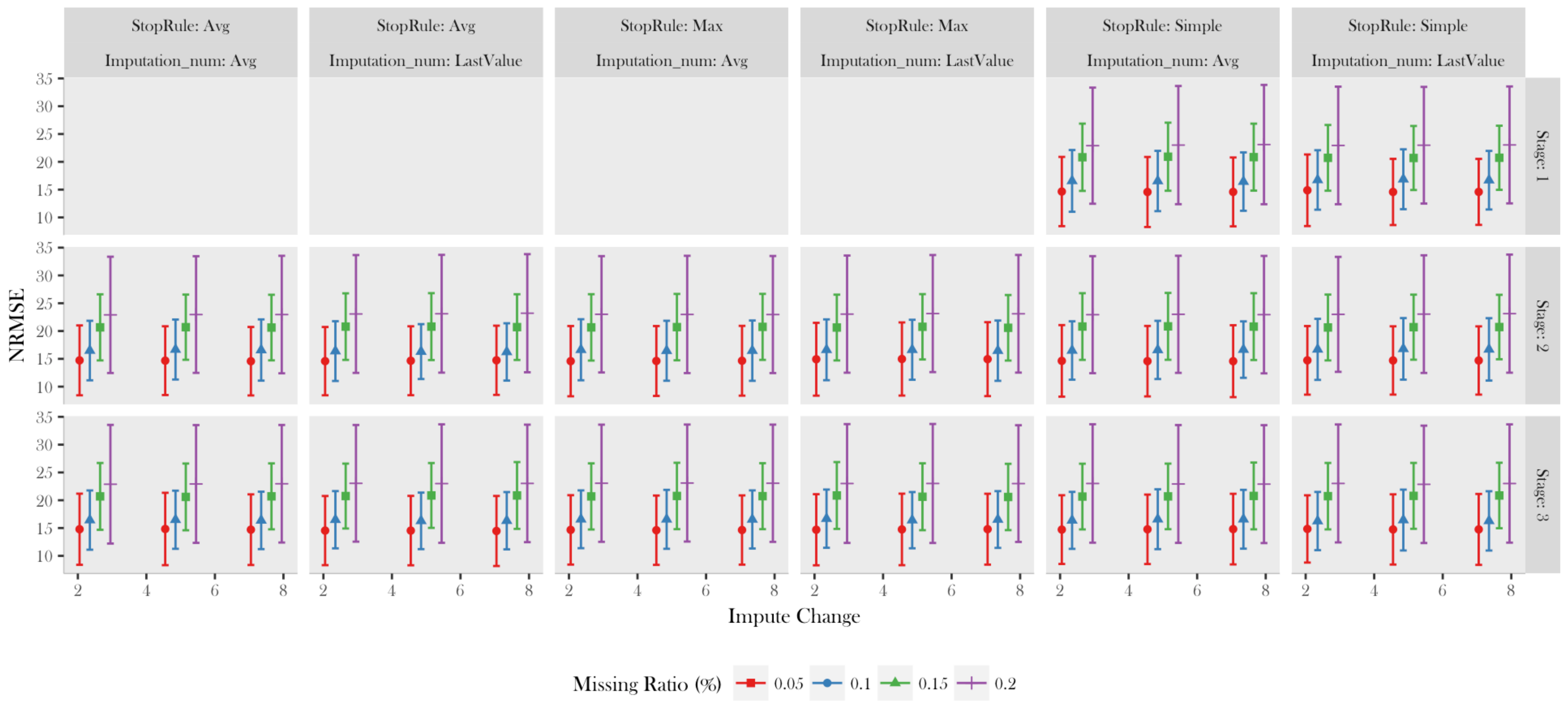


Fig. A.1.: Average NRMSE of combinations for random forest imputation parameter tuning using GA incidence matrix.

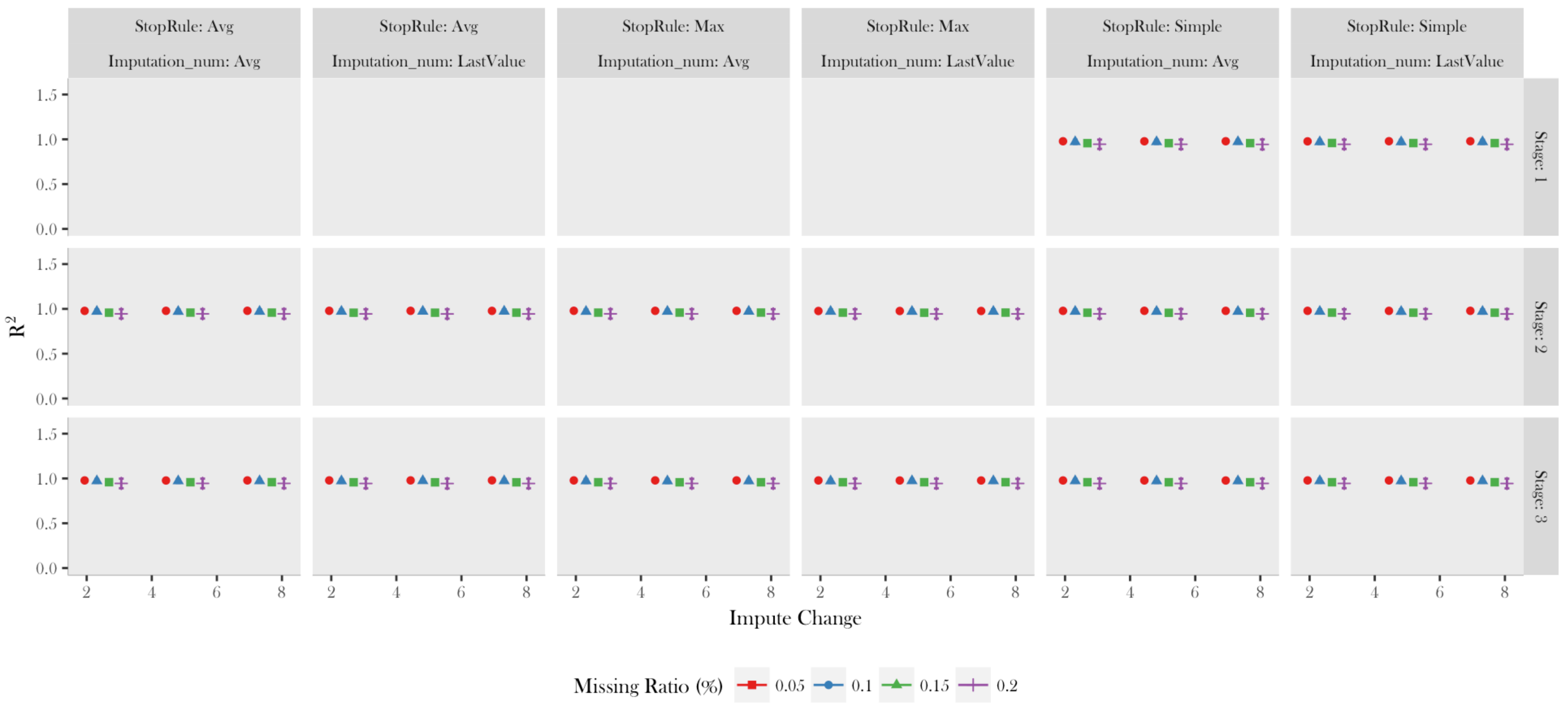


Fig. A.2.: Average R^2 of combinations for random forest imputation parameter tuning using GA incidence matrix.

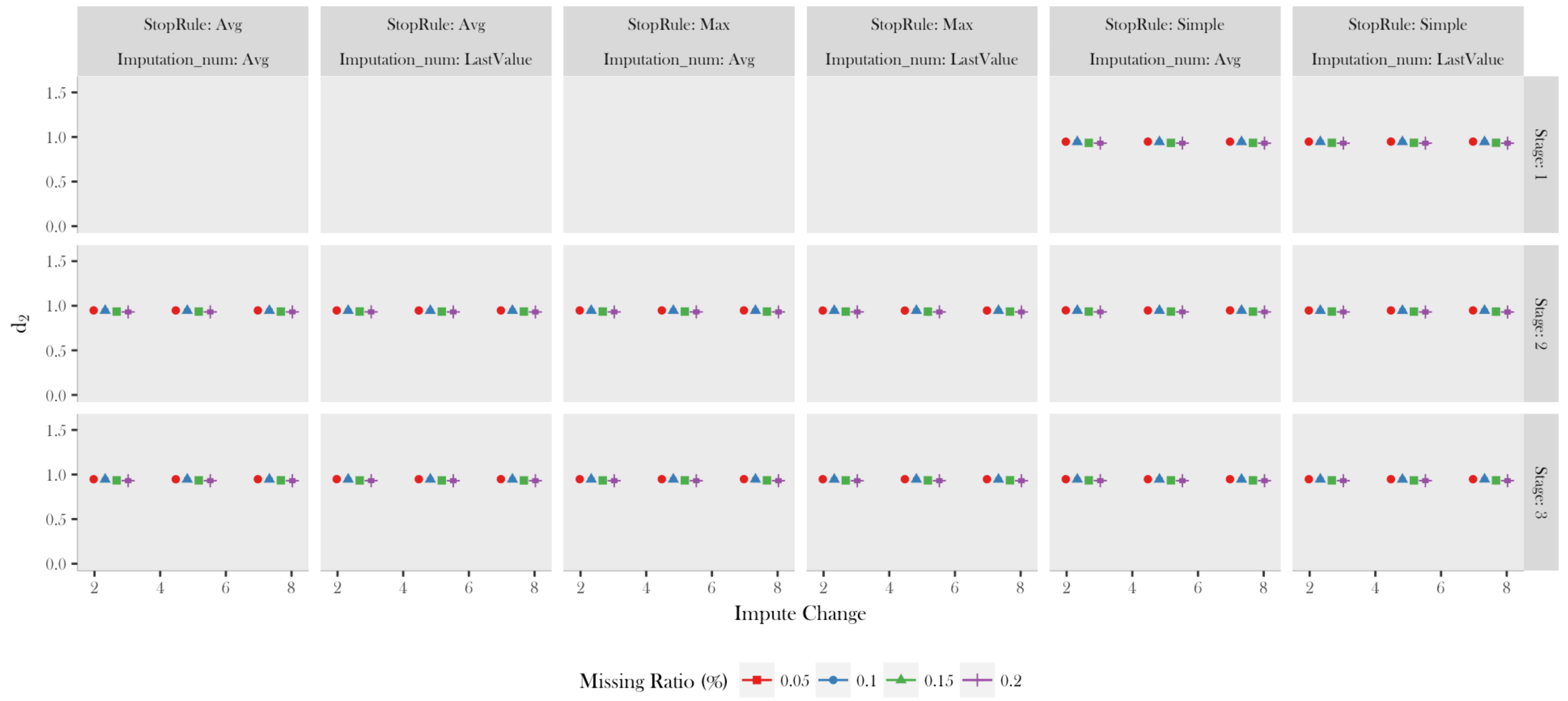


Fig. A.3.: Average d_2 of combinations for random forest imputation parameter tuning using GA incidence matrix.

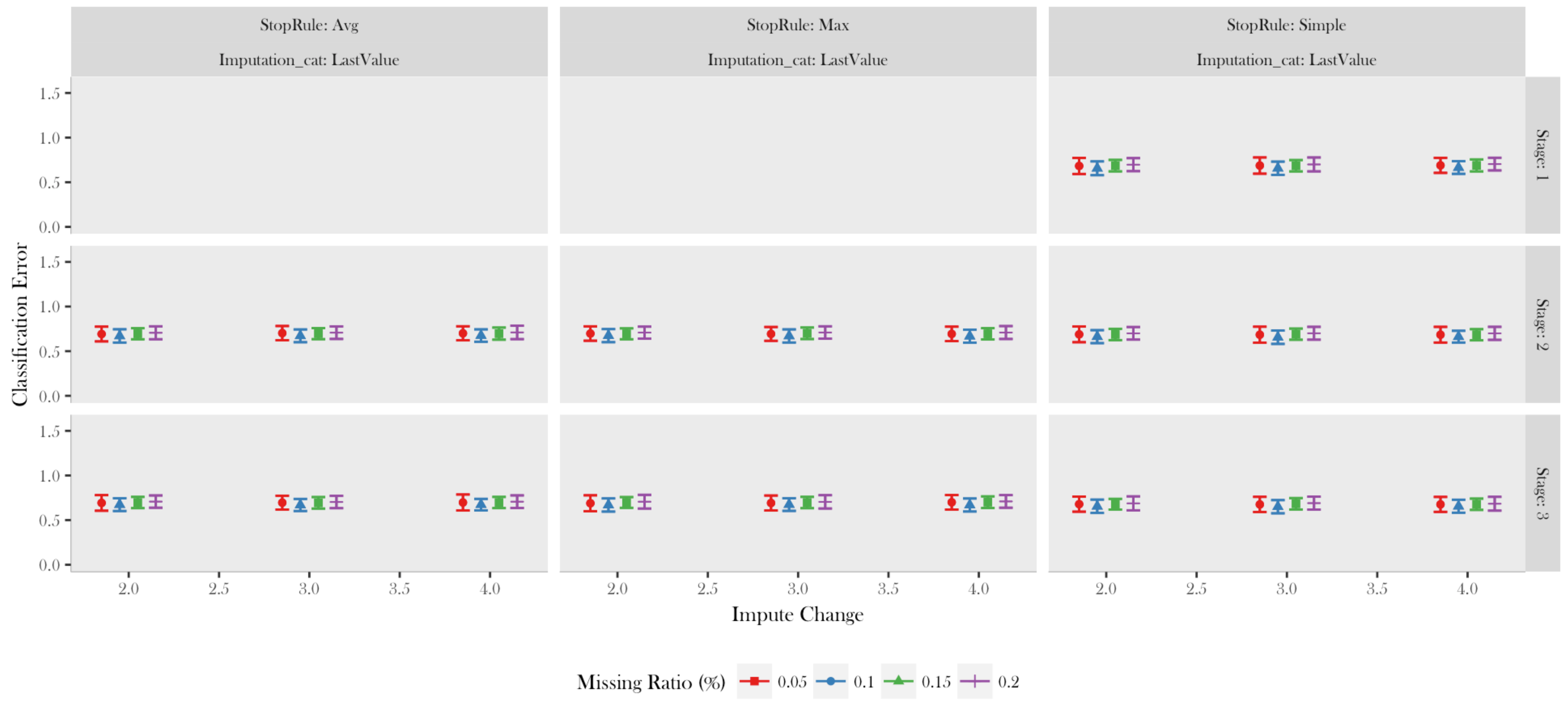


Fig. A.4.: Average classification error of combinations for random forest imputation parameter tuning using GA incidence matrix.

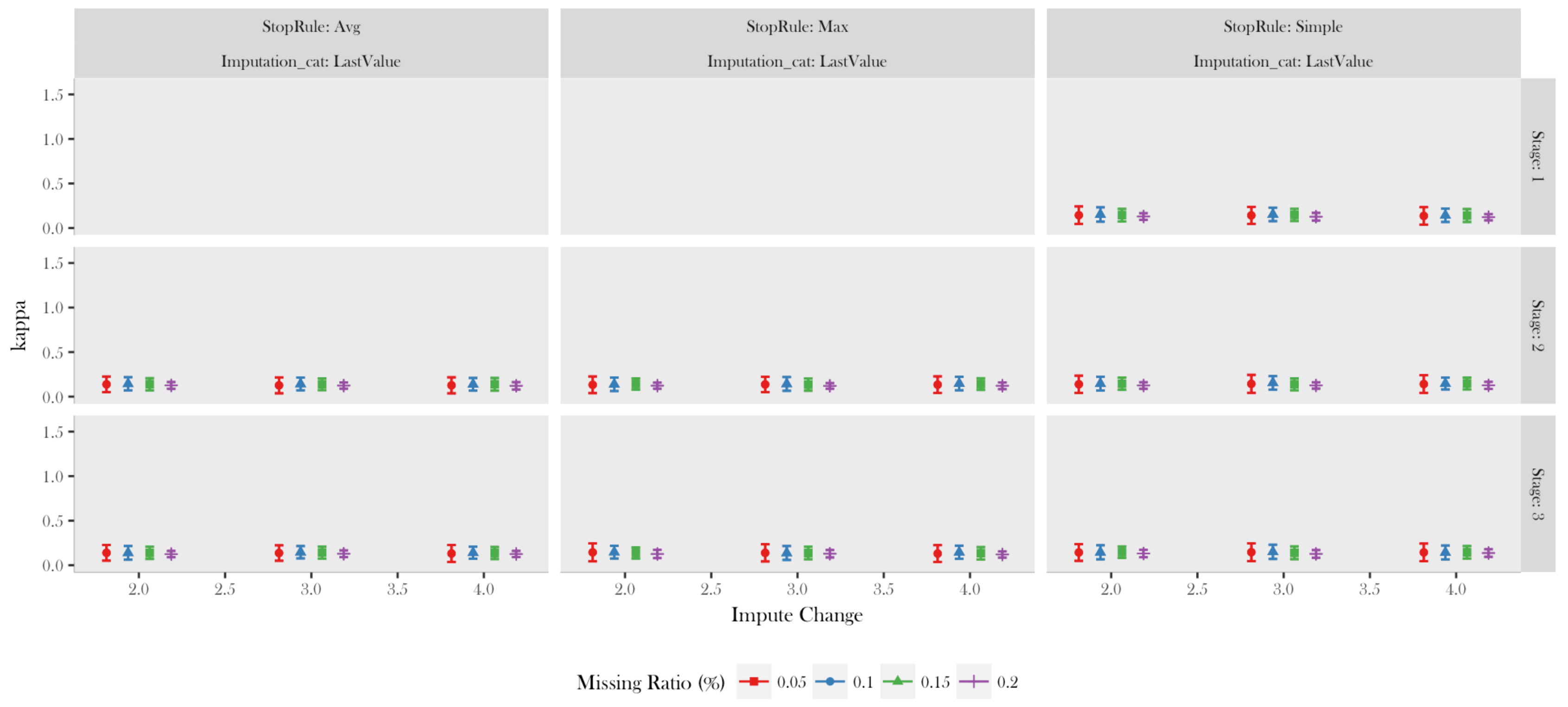


Fig. A.5.: Average kappa statistic of combinations for random forest imputation parameter tuning using GA incidence matrix.

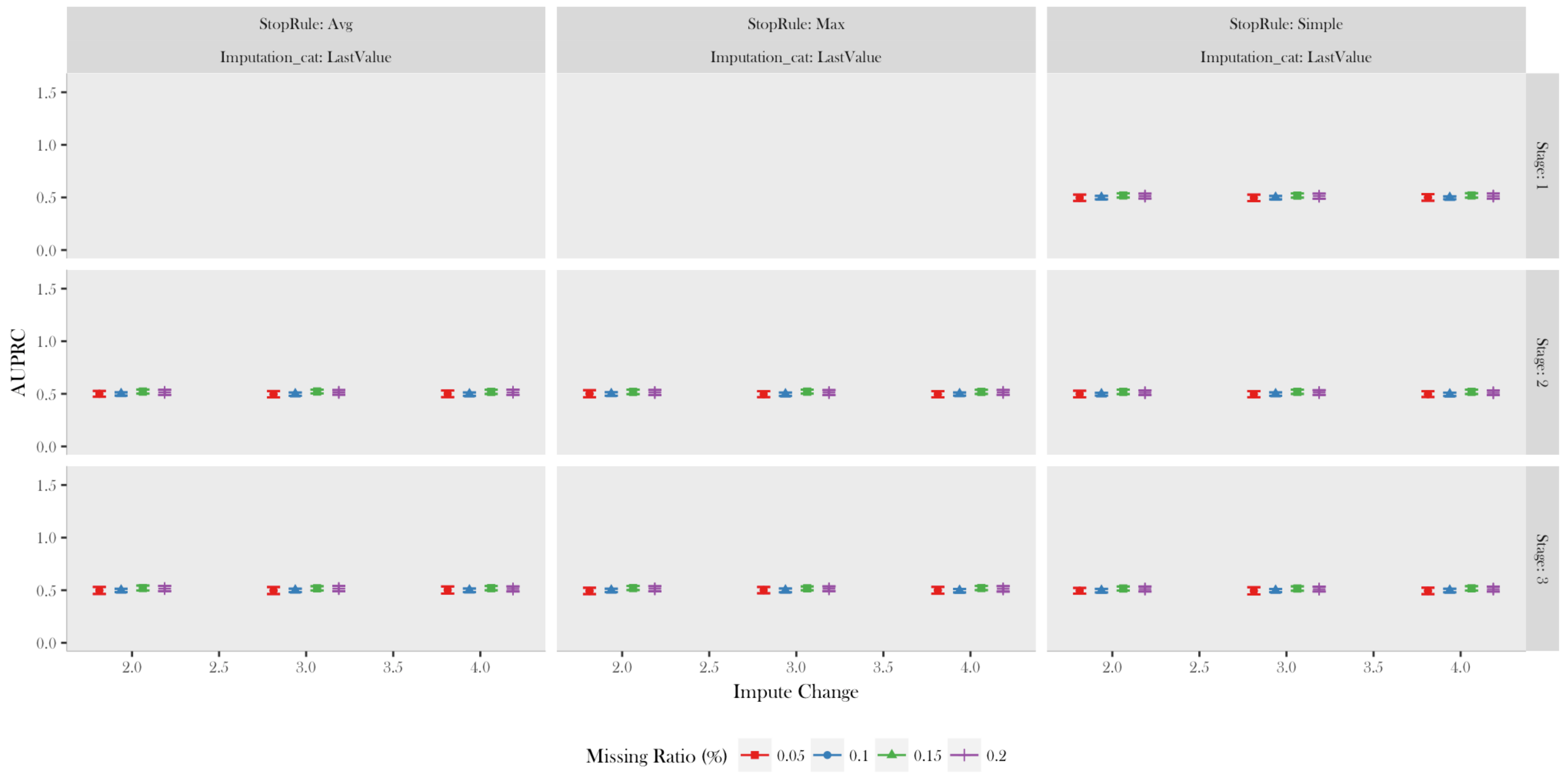


Fig. A.6.: Average AUPRC of combinations for random forest imputation parameter tuning using GA incidence matrix.

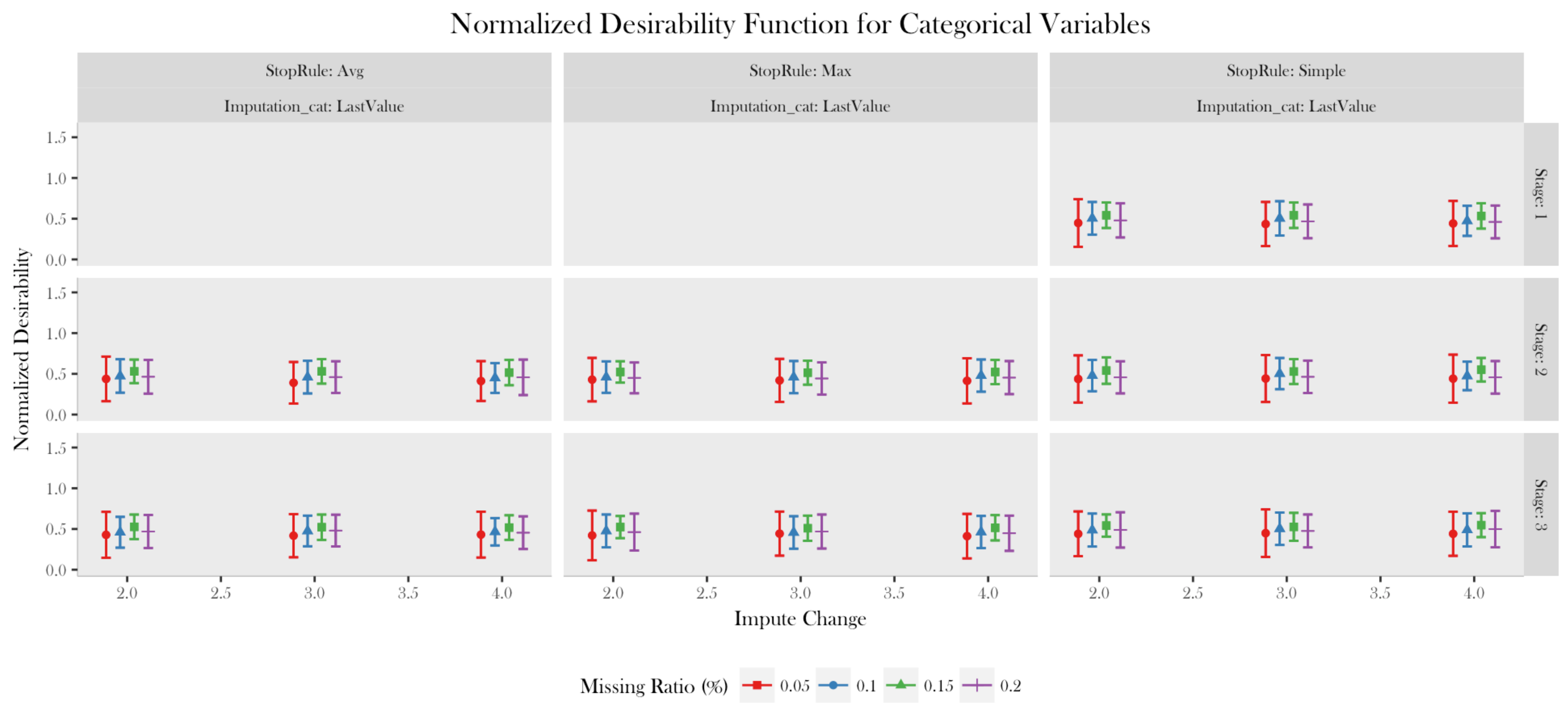


Fig. A.7.: Average categorical desirability (D_c) of combinations for random forest imputation parameter tuning using GA incidence matrix.

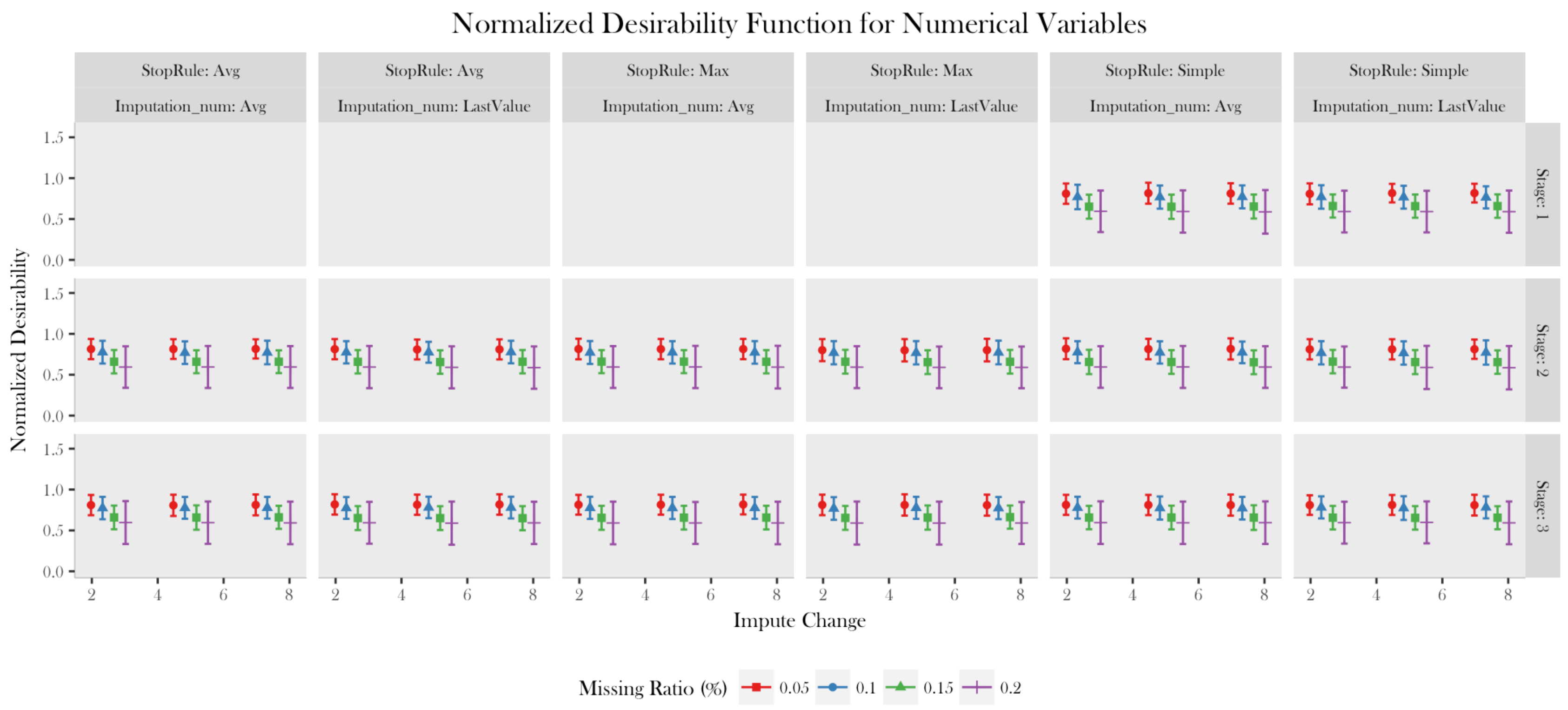


Fig. A.8.: Average numerical desirability (D_n) of combinations for random forest imputation parameter tuning using GA incidence matrix.

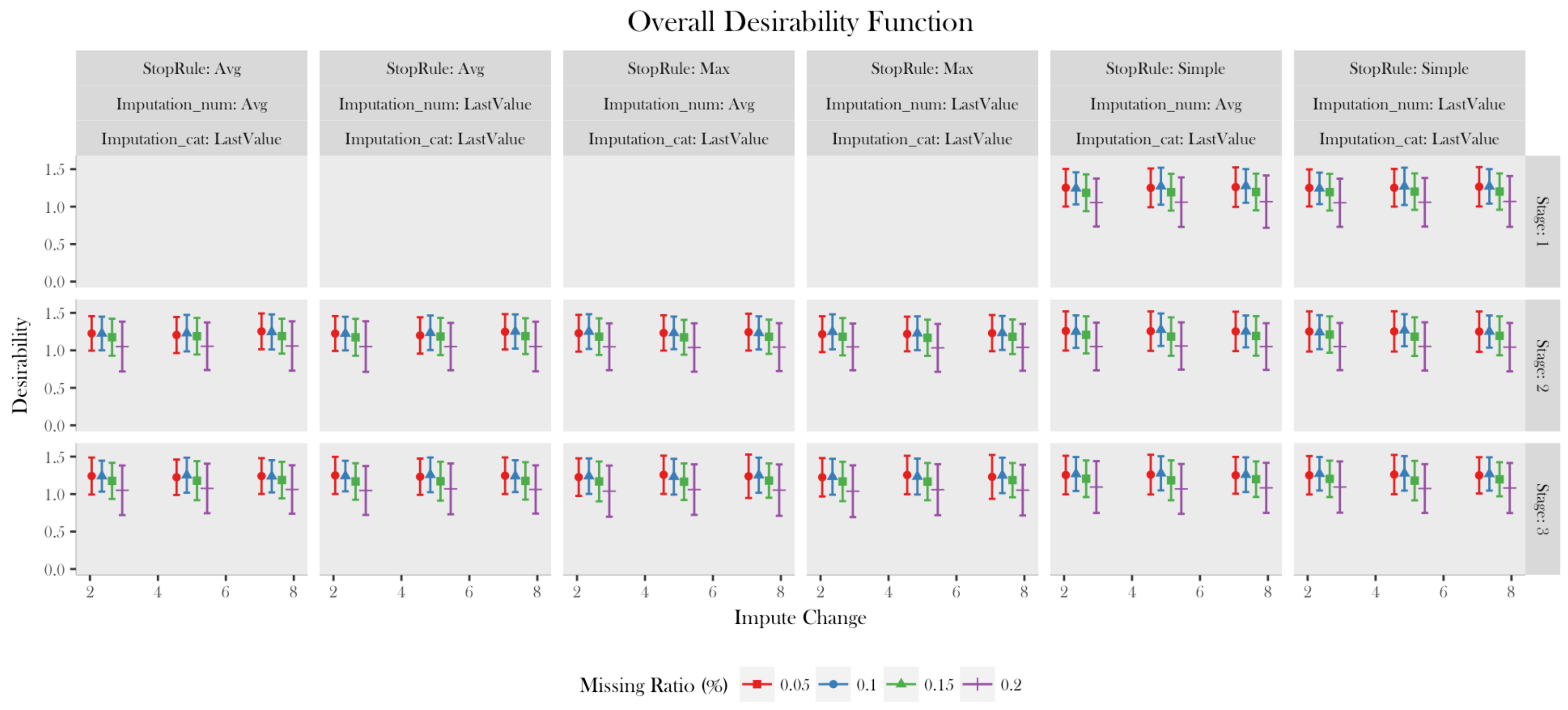


Fig. A.9.: Average overall desirability (D_o) of combinations for random forest imputation parameter tuning using GA incidence matrix.

B. EVALUATION OF IMPUTATION METHODS

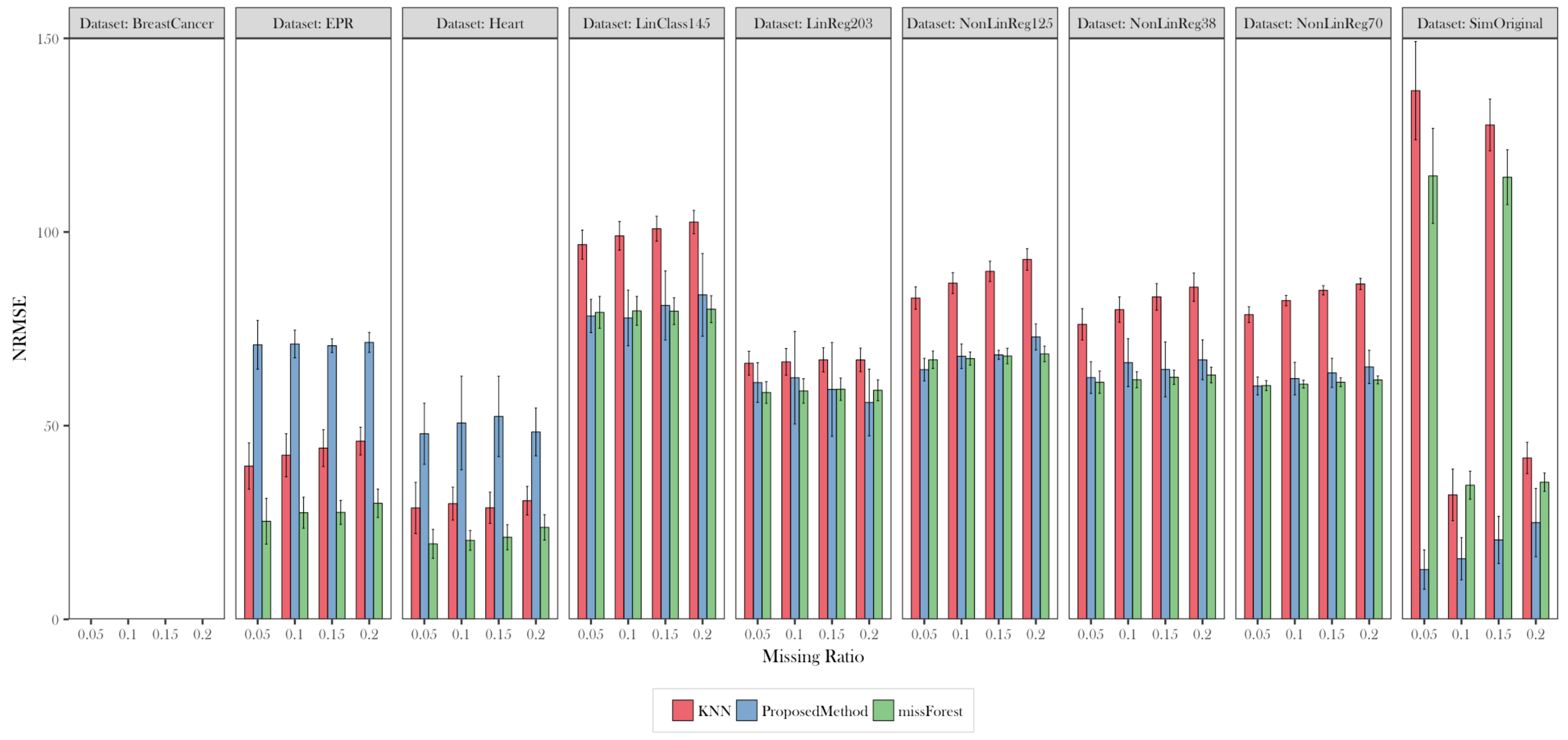


Fig. B.1.: Average NRMSE of imputation methods. Please note that results are not available for Breast Cancer data set since it only has categorical variables.

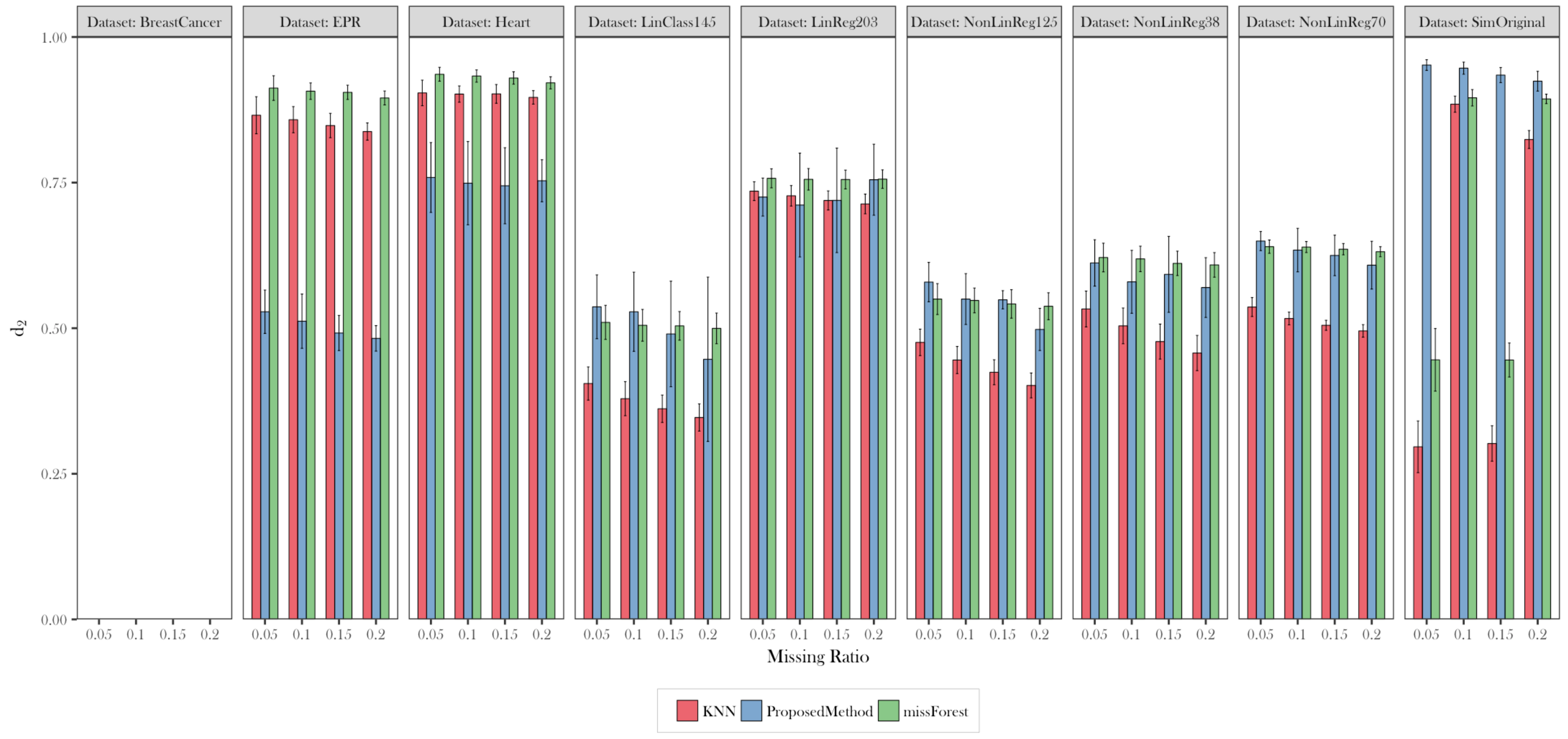


Fig. B.2.: Average d_2 of imputation methods. Please note that results are not available for Breast Cancer data set since it only has categorical variables.

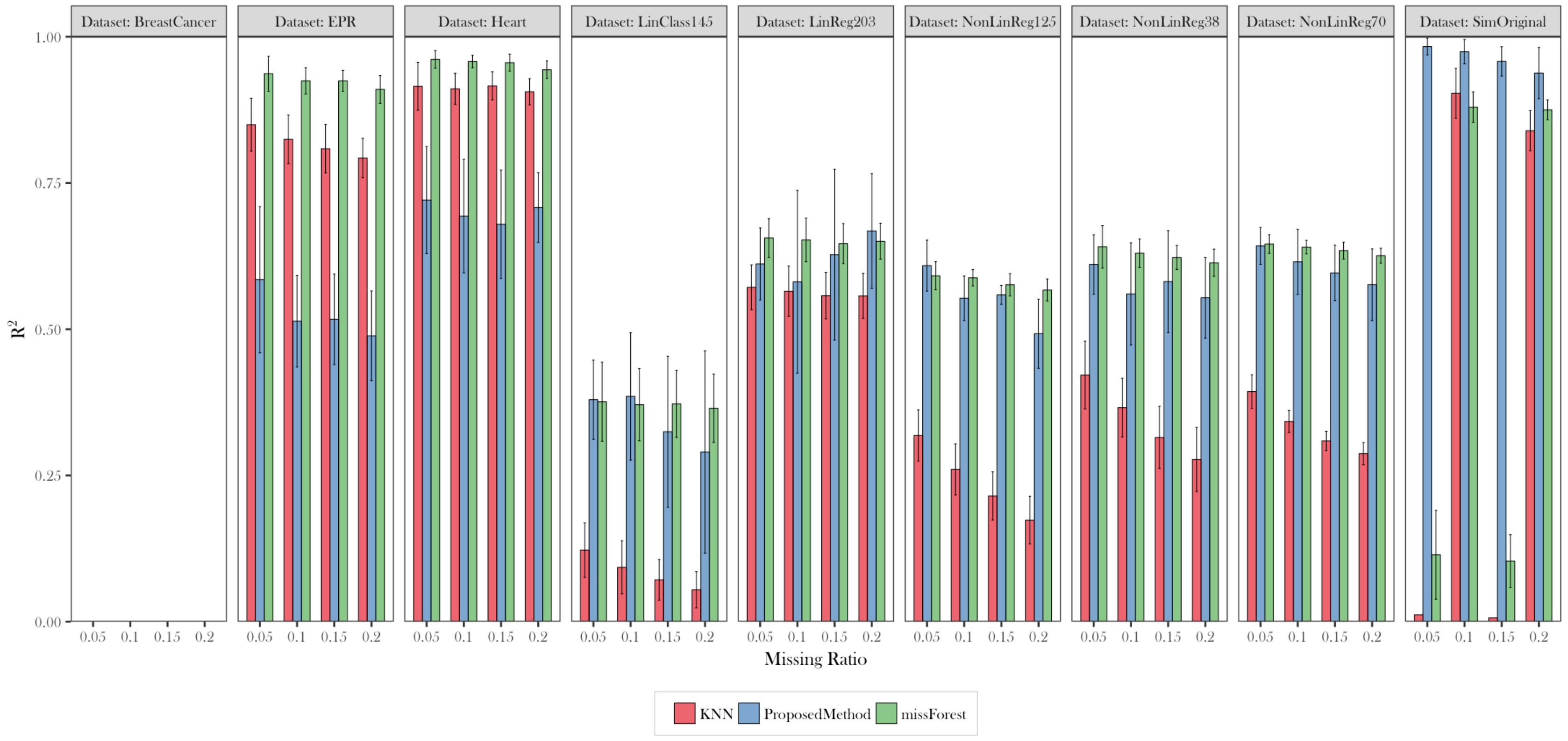


Fig. B.3.: Average R^2 of imputation methods. Please note that results are not available for Breast Cancer data set since it only has categorical variables.

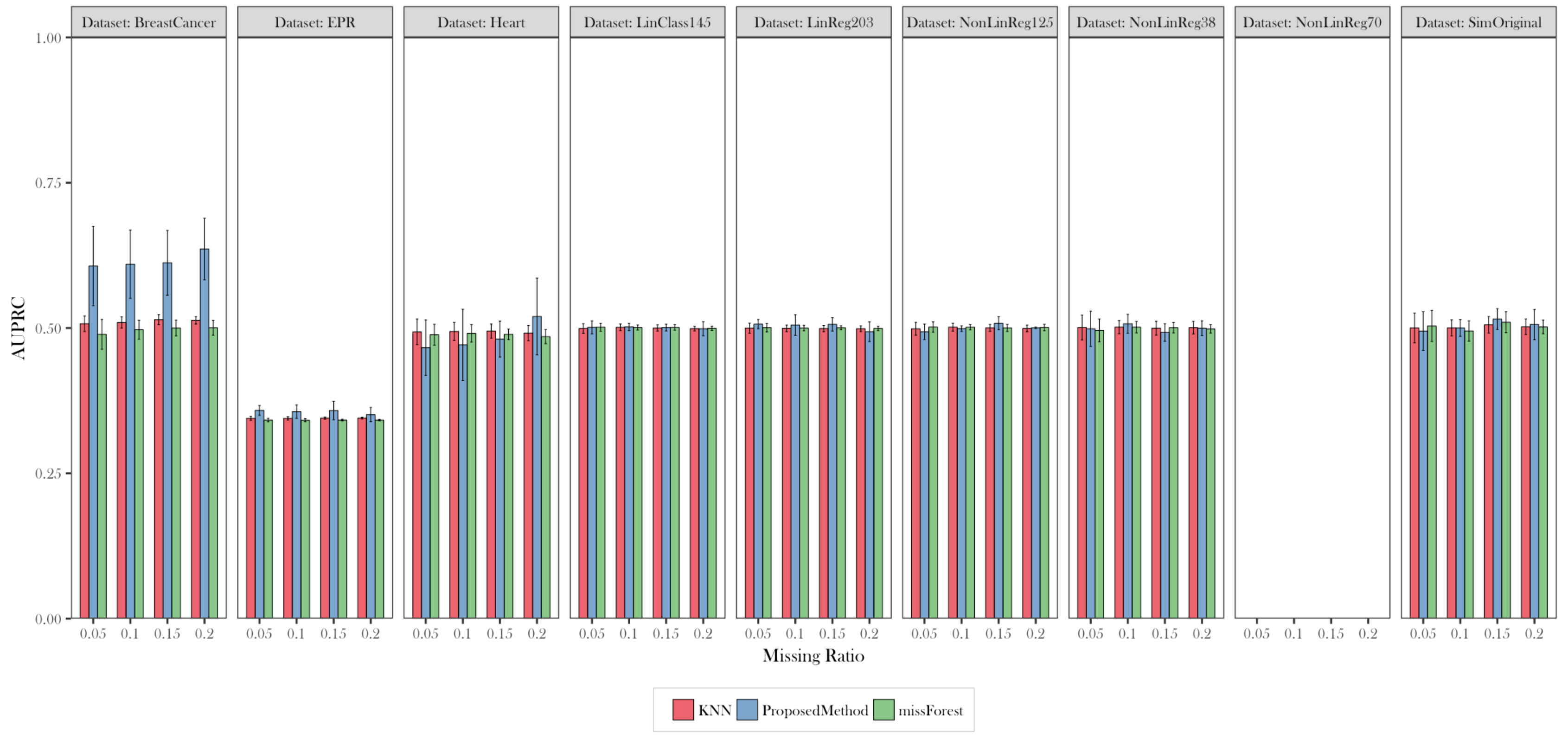


Fig. B.4.: Average AUPRC of imputation methods. Please note that results are not available for NonLinReg70 data set since it only has numerical variables.

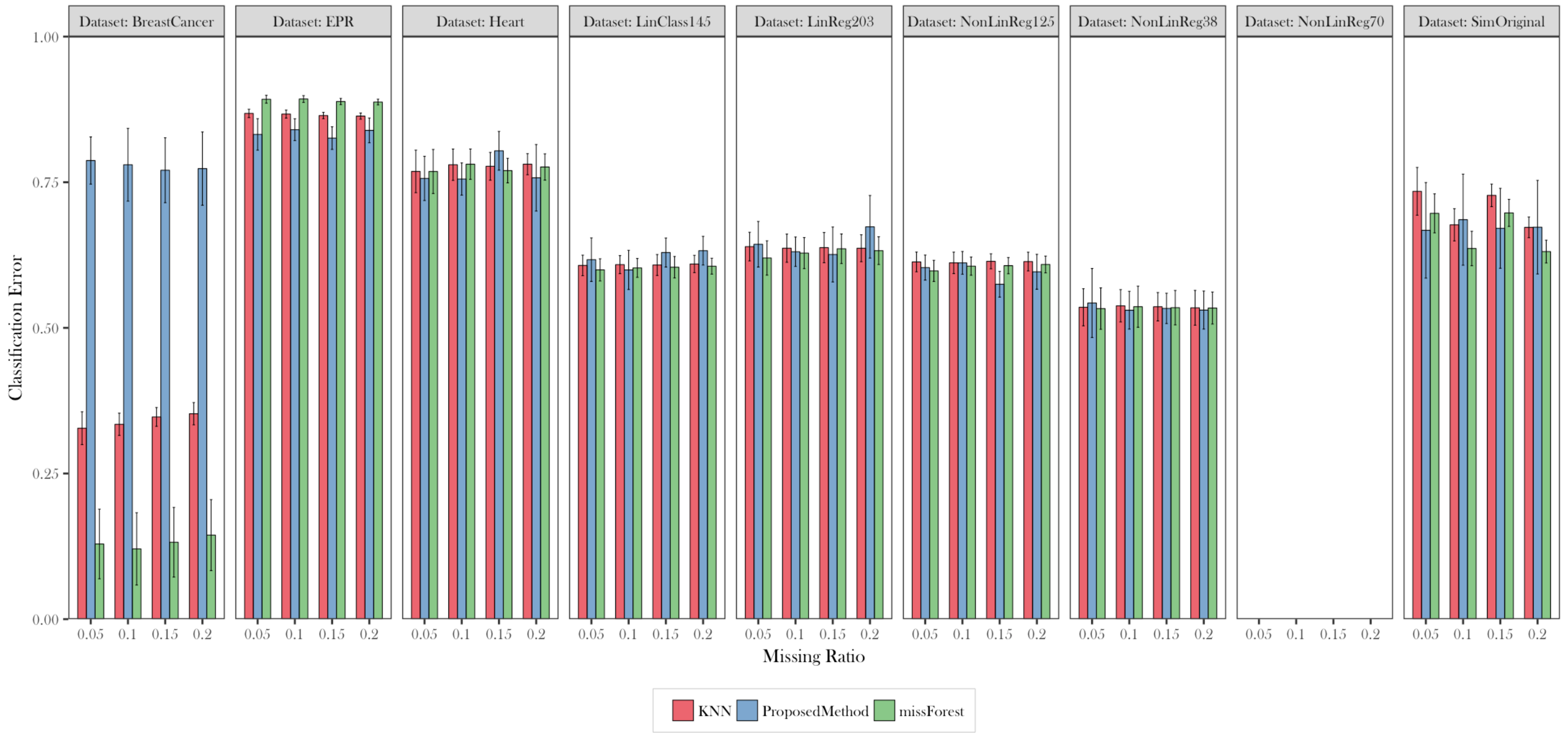


Fig. B.5.: Average classification error of imputation methods. Please note that results are not available for NonLinReg70 data set since it only has numerical variables.

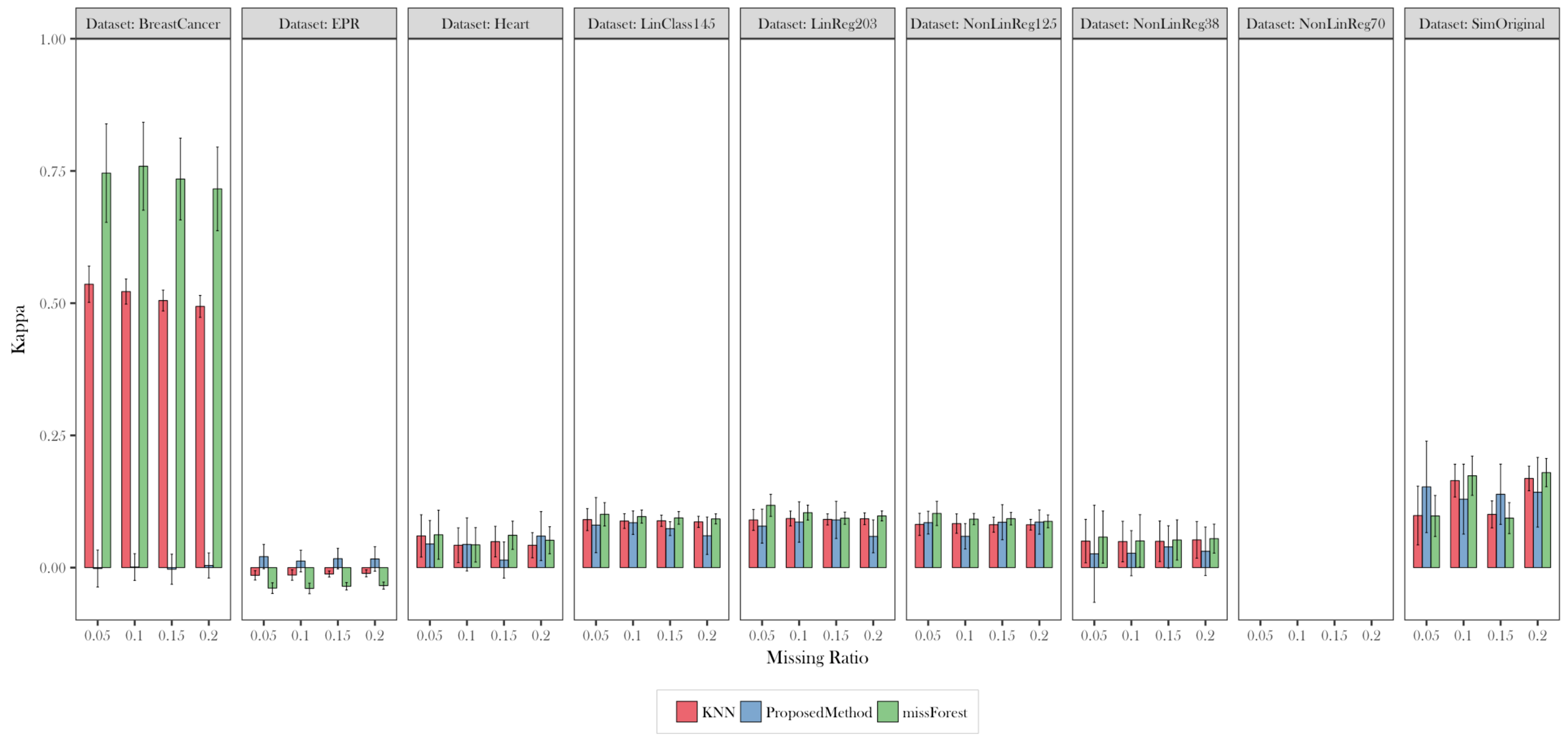


Fig. B.6.: Average kappa statistic of imputation methods. Please note that results are not available for NonLinReg70 data set since it only has numerical variables.

C. R CODES

C.1 Proposed Imputation Code

```r
## Proposed imputation Code

install.packages("randomForest")
install.packages("ForImp")
install.packages("hydroGOF")
install.packages("modeest")
install.packages("roughrf")
install.packages("psych")
install.packages("PRROC")
install.packages("caret")
library(PRROC)
library(psych)
library(randomForest)
library(ForImp)
library(hydroGOF)
library(modeest)
library(roughrf)
library(caret)

nPerm = 30  # how many times will impute same value
nRep = 10  #how many iterations of each combination


setwd("~/")  # save at My documents

# function to calculate final value of imputation
mMfunction = function(x) {
    x = x[which(x != -9999)]
    L = length(x)
    if (typeof(x) == "double") {
        mean(c(x[L], x[L - 3]))
    } else {
        x[L]
    }
}

Results = NULL
partialResults = NULL
RepResults = NULL
partialRep = NULL
ResultsCat = NULL
partialResultsCat = NULL
RepResultsCat = NULL
partialRepCat = NULL

comb = 1
comb2 = 1
comb3 = 1

iChangeCat = 4  # categorical impute chage threshold
iChange = 0.025  #numerical impute change threshold

setwd("~/")
```

```r
# GA code to get important variables for proposed
# method and creates the incidence matrix This
# incidence matrix is then used as an input in the
# imputation Data sets ###

Data = read.csv("~/Feature Selection/Data sets/Heart/HeartDataset_1.csv")
Data = Data[, -ncol(Data)]

if (sum(is.na(Data)) > 0) {
    Data = na.roughfix(Data)
}
nVar = ncol(Data)

impVarAux = matrix(nrow = nVar, ncol = 2)
Results = matrix(0, nrow = ncol(Data), ncol = ncol(Data))  #incidence matrix


for (k in 1:nVar) {
    initialtime = proc.time()
    impVars = NULL
    names(Data) = paste("X", 1:ncol(Data), sep = "")
    names(Data)[k] = "Y"

    DataX = Data[, -k]  # x variables data frame
    Y = Data[, k]  # y

    ga.fs = gafs(x = DataX, y = Y, iters = 1, gafsControl = gafsControl(functions = rfGA,
        method = "cv", number = 5), ntree = 125)  # ga fs
    impVars = ga.fs$ga$final  #important variables for variable Y been evaluated
    ones = which(names(Data) %in% c(impVars))  # extract variable number from impVars
    Results[ones, k] = 1  #assign 1 at incidence matrix to detected important
    # variables in variable evaluated
    impVarAux[k, ] = c(k, paste(impVars, collapse = ";"))  # save important variables
    #
    Newtime = proc.time() - initialtime  # run time
}

impVarAux = data.frame(impVarAux, Newtime[2])
names(impVarAux) = c("Variable", "ImpVars", "RunTime")
write.table(Results, "IncidenceMat_Heart_GA.csv", sep = ",",
    row.names = FALSE, col.names = FALSE)
write.table(impVarAux, "ImpVars_Heart_GA.csv", sep = ",",
    row.names = FALSE)


## Proposed imputation Update: Stop rule evaluating
## simple 3 stage, taking the last value imputed
## (num) or Last Val (cat) Date: Mar 12 2017 Stage:3
## Stop Crit: simple Imputation: Avg (num) LastVal
## (cat) Proposed Method-GA random forest Impute
## change num: 2.5% cat: 4 Running the more
## restrictive combination of best combination
## results
```

```r
for (k in 1:nRep) {

    for (misPercentage in c(0.05, 0.1, 0.15, 0.2)) {
        # missing ratio in data sets
        initialtime = proc.time()
        # change data, data2 and impVars depending on data
        # set
        data = read.csv(paste("~/Heizel/SimData/SimDataWITH.",
            k, ".", misPercentage, ".csv", sep = ""))  #data set with missingvalues
        data2 = read.csv(paste("~/Heizel/SimData/SimDataWITHOUT.",
            k, ".", misPercentage, ".csv", sep = ""))  #complete dataset
        impVars = read.csv("~/Feature Selection/Feature selection for proposed
                    #imputation/Incidence Matrices CFS/IncidenceMat_
                    SimDataOriginal_CFS.csv",
            header = FALSE)

        # Evaluate which variables are categorical and
        # numerical
        for (v in 1:ncol(data)) {
            cat = length(unique(data[, v]))
            if (cat <= 12) {
                data[, v] = as.factor(data[, v])
            } else {
                data[, v] = as.numeric(data[, v])
            }
        }

        data = as.data.frame(data)
        data2 = as.data.frame(data2)

        # Data goes here
        X = data[, -ncol(data)]
        Y = data.frame(data[, ncol(data)])
        missDF = data.frame(X, Y)
        names(missDF) = paste("X", 1:ncol(X), sep = "")
        names(missDF)[ncol(data)] = "Y"

        # Variable Imputation Order Ascending
        varMV = matrix(nrow = ncol(missDF) - 1, ncol = 1)
        for (numCol in 1:ncol(missDF) - 1) {
            # For each column, identify the number of missing
            # values
            varMV[numCol] = sum(is.na(missDF[, numCol]))
        }

        MVdf = data.frame(1:(ncol(missDF) - 1), varMV)
        names(MVdf) = c("NumCol", "varMV")
        AscOrder = order(MVdf[, ncol(MVdf)])  # How to order variables for missing
        # value imputation

        for (i in AscOrder) {
            misRows = which(is.na(missDF[, i]) == TRUE)  #ID missing values in variable
            numMis = length(misRows)
```

```r
imputedData = matrix(-9999, ncol = nPerm,
    nrow = numMis)  # matrix with j imputations
# of observations
imputedDataCat = matrix(-9999, ncol = nPerm,
    nrow = numMis)
imp = which(impVars[, i] == 1)  #importantn variables to be used in rF
imputedDataAux = matrix(-9999, ncol = nPerm,
    nrow = numMis)  # aux matrix with j
# imputations of observations
imputedDataAuxCat = matrix(-9999, ncol = nPerm,
    nrow = numMis)  # aux matrix with j
# imputations of observations

if (typeof(data[, i]) == "double") {

    imputeChange = 0.1  #initial value of numerical impute change

    for (r in 1:numMis) {
      j = 1  #impute round

      while (j <= 3 | (j <= 30 & imputeChange >
        iChange)) {
        # Variables to use in the imputation process
        if (length(imp) >= 1) {
          cols = c(i, imp, ncol(missDF))
        } else {
          cols = c(i, ncol(missDF))
        }

        impDF = na.roughfix(missDF)
        NAloc = is.na(missDF)
        obsi = !NAloc[, cols[1]]
        misi = NAloc[, cols[1]]
        obsX = impDF[obsi, cols[-1]]
        obsY = impDF[obsi, cols[1]]
        misx = impDF[misi, cols[-1]]

        rF = randomForest(y = obsY, x = obsX,
          ntree = 150, replace = TRUE)
        imputedData[, j] = predict(object = rF,
          newdata = misx, type = "response")
        imputedDataAux[r, j] = c(imputedData[r,
          j])

        if (j > 3) {
          imputeChange = abs((imputedDataAux[r,
            j] - imputedDataAux[r, j -
            3])/(imputedDataAux[r, j -
            3]))
        } else {
          imputeChange = 0.1
        }
```

```
          if (j >= 3 & imputedDataAux[r,
            j] == -9999) {
            missDF[misRows[r], i] = imputedDataAux[r,
              j - 1]
          }
          j = j + 1

        }
      }
      partialRes = data.frame(k, i, misPercentage,
        iChange, cbind(data2[misRows, i],
          apply(imputedDataAux, 1, mMfunction)))
      names(partialRes) = c("Rep", "Variable",
        "%Missing", "Impute Change", "Actual",
        "Imputed")
      if (comb == 1) {
        Results = partialRes
      } else {
        Results = rbind(Results, partialRes)
      }
      names(Results) = c("Rep", "Variable",
        "%Missing", "Impute Change", "Actual",
        "Imputed")

      write.table(Results, "StopRule_3stageSimpleLastVal_ActImpNum_SimDataOrig.csv",
        row.names = FALSE, sep = ",")  #change depending data set

      # calculate performance metrics on numeric
      # variables
      repData = subset(Results, Rep == k &
        Results[, 3] == misPercentage & Results[,
        4] == iChange, 5:6)
      repData = data.frame(repData)
      names(repData) = c("Actual", "Imputed")
      metrics = gof(sim = repData[, 2], obs = repData[,
        1])
      NRMSE = metrics[5]
      R2 = metrics[17]
      d2 = metrics[13]
      partialRep = data.frame(k, i, misPercentage,
        iChange, R2, NRMSE, d2)
      names(partialRep) = c("Rep", "Variable",
        "%Missing", "ImputeChange", "R2",
        "NRMSE", "d2")
      if (comb == 1) {
        RepResults = partialRep
      } else {
        RepResults = rbind(RepResults, partialRep)
      }

      write.table(RepResults, "ResultsStopRule_3stageSimpleLastValNum_SimDataOrig.csv",
        row.names = FALSE, sep = ",")
      comb = comb + 1
```

```r
      } else {

        imputeChangeCat = 10  #initial value of categorical impute change

        for (r in 1:numMis) {
          z = 1  #impute round

          while (z <= 3 | (z <= 30 & imputeChangeCat >
            iChangeCat)) {

            # Variables to use in the imputation process
            if (length(imp) >= 1) {
              cols = c(i, imp, ncol(missDF))
            } else {
              cols = c(i, ncol(missDF))
            }

            impDF = na.roughfix(missDF)
            NAloc = is.na(missDF)
            obsi = !NAloc[, cols[1]]
            misi = NAloc[, cols[1]]
            obsX = impDF[obsi, cols[-1]]
            obsY = impDF[obsi, cols[1]]
            misx = impDF[misi, cols[-1]]

            rF = randomForest(y = obsY, x = obsX,
              ntree = 150, replace = TRUE)
            imputedDataCat[, z] = predict(object = rF,
              newdata = misx, type = "response")
            imputedDataAuxCat[r, z] = c(imputedDataCat[r,
              z])

            if (z > 3) {
              if (imputedDataAuxCat[r, z] ==
                imputedDataAuxCat[r, z - 1] &
                imputedDataAuxCat[r, z - 1] ==
                  imputedDataAuxCat[r, z -
                    2] & imputedDataAuxCat[r,
                z - 2] == imputedDataAuxCat[r,
                z - 3]) {
                imputeChangeCat = 4
              } else {
                imputeChangeCat = 10
              }
            }

            if (z >= 3 & imputedDataAuxCat[r,
              z] == -9999) {
              missDF[misRows[r], i] = imputedDataAuxCat[r,
                z - 1]
            }
            z = z + 1
```

```
                  }
                }
                partialResCat = data.frame(k, i, misPercentage,
                  iChangeCat, cbind(data2[misRows,
                    i], apply(imputedDataAuxCat, 1,
                    mMfunction)))
                names(partialResCat) = c("Rep", "Variable",
                  "%Missing", "Impute Change", "Actual",
                  "Imputed")
                if (comb2 == 1) {
                  ResultsCat = partialResCat
                } else {
                  ResultsCat = rbind(ResultsCat, partialResCat)
                }
                names(ResultsCat) = c("Rep", "Variable",
                  "%Missing", "Impute Change", "Actual",
                  "Imputed")

                write.table(ResultsCat, "StopRule_3stageSimpleLastVal_ActImpCat_SimDataOrig.csv",
                  row.names = FALSE, sep = ",")

                Newtime = proc.time() - initialtime  # run time
                # calculate performance metrics on categorical
                # variables
                repDataCat = subset(ResultsCat, Rep ==
                  k & ResultsCat[, 3] == misPercentage &
                  ResultsCat[, 4] == iChangeCat, 5:6)
                repDataCat = data.frame(repDataCat)
                names(repDataCat) = c("Actual", "Imputed")
                error = sum(repDataCat$Imputed != repDataCat$Actual)/nrow(repDataCat)
                Kappa = cohen.kappa(repDataCat)$kappa
                AUC = pr.curve(repDataCat$Actual, repDataCat$Imputed)$auc.integral
                partialRepCat = data.frame(k, i, misPercentage,
                  iChangeCat, error, Kappa, AUC, Newtime[2])
                names(partialRepCat) = c("Rep", "Variable",
                  "%Missing", "ImputeChange", "ClassError",
                  "Kappa", "PR AUC", "RunTime")
                if (comb2 == 1) {
                  RepResultsCat = partialRepCat
                } else {
                  RepResultsCat = rbind(RepResultsCat,
                    partialRepCat)
                }
                comb2 = comb2 + 1

                write.table(RepResultsCat, "ResultsStopRule_3stageSimpleLastValCat_SimDataOrig.csv",
                  row.names = FALSE, sep = ",")

            }  #else closes
        }
    }
}
```

C.2 KNN and missForest Codes

```r
######### missForest imputation Code #####################

install.packages("cutoffR")
install.packages("hydroGOF")
install.packages("missForest")
install.packages("psych")
install.packages("PRROC")
library(PRROC)
library(psych)
library(cutoffR)
library(hydroGOF)
library(missForest)

setwd("~/")

nRep = 30
Results = NULL
partialRes = NULL
RepResults = NULL
partialRep = NULL
ResultsCat = NULL
partialResCat = NULL
RepResultsCat = NULL
partialRepCat = NULL
comb = 1


for (k in 1:nRep) {
    for (misPercentage in c(0.05, 0.1, 0.15, 0.2)) {
        library(missForest)
        DataImp = NULL
        data = NULL
        data2 = NULL
        initialtime = proc.time()

        data = read.csv(paste("~/Missing Value Imputation/Datasets with simulated Missing
          Values/SimLinDataReg203 with MVs/SimLinearData_203_reg_",
            k, "_", misPercentage, ".csv", sep = ""))
        data2 = read.csv(paste("~/Feature Selection/Data sets/SimLinearData_203_reg
                         /LinearRelDataset_",
            k, ".csv", sep = ""))

        # Evaluate which variables are categorical and
        # numerical
        catVars = matrix(0, nrow = 1, ncol = ncol(data))  #vector que ID cat vars
        # 1=cat 0=num
        for (v in 1:ncol(data)) {
            cat = length(unique(data[, v]))
            if (cat <= 10) {
                data[, v] = as.factor(data[, v])
                catVars[, v] = 1
            } else {
                data[, v] = as.numeric(data[, v])
            }
        }

        data = as.data.frame(data)
        data2 = as.data.frame(data2)

        DataImp = missForest(xmis = data, ntree = 100)


        # matrix with categorical variables only
        dataCat = data[, which(catVars == 1)]  #miss matrix
        data2Cat = data2[, which(catVars == 1)]  #real matrix
        DataImpCat = DataImp$ximp[, which(catVars ==
            1)]  #pred matrix

        # matrix with numerical variables only
        dataNum = data[, which(catVars == 0)]  #miss matrix
        data2Num = data2[, which(catVars == 0)]  #real matrix
        DataImpNum = DataImp$ximp[, which(catVars ==
            0)]  #pred matrix


        # Performance evaluation of catecorical vars
        YrealCat = NULL
        YpredCat = NULL

        for (n in 1:ncol(dataCat)) {
            MissVal = which(is.na(dataCat[, n]) ==
                TRUE)
            YrealCat = append(YrealCat, data2Cat[MissVal,
                n])
            YpredCat = append(YpredCat, DataImpCat[MissVal,
                n])
        }

        partialResCat = data.frame(k, misPercentage,
            YrealCat, YpredCat)
        names(partialResCat) = c("Rep", "%Missing",
            "Actual", "Imputed")
        if (comb == 1) {
            ResultsCat = partialResCat
        } else {
            ResultsCat = rbind(ResultsCat, partialResCat)
        }

        # save imputation data (categorical)
        write.table(ResultsCat, "missForest_ActualImpDataCat_SimLinReg203.csv",
            row.names = FALSE, sep = ",")

        repDataCat = subset(ResultsCat, Rep == k &
            ResultsCat[, 2] == misPercentage, c(3,
            4))
        repDataCat = data.frame(repDataCat)
```

```
    names(repDataCat) = c("Actual", "Imputed")
    error = sum(repDataCat$Imputed != repDataCat$Actual)/nrow(repDataCat)
    Kappa = cohen.kappa(repDataCat)$kappa
    AUC = pr.curve(repDataCat$Actual, repDataCat$Imputed)$auc.integral
    partialRepCat = data.frame(k, misPercentage,
        error, Kappa, AUC)
    names(partialRepCat) = c("Rep", "%Missing",
        "ClassError", "Kappa", "AUC")
    if (comb == 1) {
        RepResultsCat = partialRepCat
    } else {
        RepResultsCat = rbind(RepResultsCat, partialRepCat)
    }

    write.table(RepResultsCat, "missForestResultsCat_SimLinReg203.csv",
        row.names = FALSE, sep = ",")

    Yreal = NULL
    Ypred = NULL

    for (n in 1:ncol(dataNum)) {
        MissVal = which(is.na(dataNum[, n]) ==
            TRUE)
        Yreal = append(Yreal, data2Num[MissVal,
            n])
        Ypred = append(Ypred, DataImpNum[MissVal,
            n])
    }

    partialRes = data.frame(k, misPercentage, Yreal,
        Ypred)
    names(partialRes) = c("Rep", "%Missing", "Actual",
        "Imputed")
    if (comb == 1) {
        Results = partialRes
    } else {
        Results = rbind(Results, partialRes)
    }

    # save imputation data (numerical)
    write.table(Results, "missForest_ActualImpDataNum_SimLinReg203.csv",
        row.names = FALSE, sep = ",")

    Newtime = proc.time() - initialtime  # run time
    repData = subset(Results, Rep == k & Results[,
        2] == misPercentage, c(3, 4))
    repData = data.frame(repData)
    names(repData) = c("Actual", "Imputed")
    repData[, 1] = as.numeric(repData[, 1])
    repData[, 2] = as.numeric(repData[, 2])
    detach("package:missForest", unload = TRUE)
    metrics = gof(sim = repData[, 2], obs = repData[,
        1])
```

```r
        NRMSE = metrics[5]
        R2 = metrics[17]
        d2 = metrics[13]
        partialRep = data.frame(k, misPercentage, R2,
            NRMSE, d2, Newtime[2])
        names(partialRep) = c("Rep", "%Missing", "R2",
            "NRMSE", "d2", "RunTime")
        if (comb == 1) {
            RepResults = partialRep
        } else {
            RepResults = rbind(RepResults, partialRep)
        }
        comb = comb + 1

        write.table(RepResults, "missForestResultsNum_SimLinReg203.csv",
            row.names = FALSE, sep = ",")
    }
}

RepResults
```

```r
######### KNN imputation Code ######################
install.packages("cutoffR")
install.packages("hydroGOF")
install.packages("psych")
install.packages("PRROC")
install.packages("VIM")
install.packages("dplyr")
library(dplyr)
library(VIM)
library(PRROC)
library(psych)
library(cutoffR)
library(hydroGOF)

setwd("~/")

nRep = 30
Results = NULL
partialRes = NULL
RepResults = NULL
partialRep = NULL
ResultsCat = NULL
partialResCat = NULL
RepResultsCat = NULL
partialRepCat = NULL
comb = 1
options(warn = -1)
for (k in 5:nRep) {
    for (misPercentage in c(0.05, 0.1, 0.15, 0.2)) {
        DataImp = NULL
        data = NULL
        data2 = NULL
        initialtime = proc.time()

        data = read.csv(paste("~/Missing Value Imputation/Datasets with simulated Missing
            Values/Sylva Datasets with MVs/Sylva_",
            k, "_", misPercentage, ".csv", sep = ""))
        data2 = read.csv(paste("~/Feature Selection/Data sets/SYLVA/SylvaDataset_",
            k, ".csv", sep = ""))

        # Evaluate which variables are categorical and
        # numerical

        catVars = matrix(0, nrow = 1, ncol = ncol(data))  #vector que ID cat vars 1=cat 0=num
        for (v in 1:ncol(data)) {
            cat = n_distinct(data[, v], na.rm = TRUE)
            if (cat <= 10) {
                data[, v] = as.factor(data[, v])
                catVars[, v] = 1
            } else {
                data[, v] = as.numeric(data[, v])
            }
        }
        data = as.data.frame(data)
        data2 = as.data.frame(data2)

        DataImp = kNN(data, imp_var = FALSE)  #  knn imputation

        # matrix with categorical variables only
        dataCat = data[, which(catVars == 1)]  #miss matrix
        data2Cat = data2[, which(catVars == 1)]  #real matrix
        DataImpCat = DataImp[, which(catVars == 1)]  #pred matrix

        # matrix with numerical variables only
        dataNum = data[, which(catVars == 0)]  #miss matrix
        data2Num = data2[, which(catVars == 0)]  #real matrix
        DataImpNum = DataImp[, which(catVars == 0)]  #pred matrix


        # Performance evaluation of catecorical vars
        YrealCat = NULL
        YpredCat = NULL

        for (n in 1:ncol(dataCat)) {
            MissVal = which(is.na(dataCat[, n]) ==
                TRUE)
            YrealCat = append(YrealCat, data2Cat[MissVal,
                n])
            YpredCat = append(YpredCat, DataImpCat[MissVal,
                n])
        }

        partialResCat = data.frame(k, misPercentage,
            YrealCat, YpredCat)
        names(partialResCat) = c("Rep", "%Missing",
            "Actual", "Imputed")
        if (comb == 1) {
            ResultsCat = partialResCat
        } else {
            ResultsCat = rbind(ResultsCat, partialResCat)
        }

        # save imputation data (categorical)
        write.table(ResultsCat, "kNN_ActualImpDataCat_Sylva2.csv",
            row.names = FALSE, sep = ",")

        repDataCat = subset(ResultsCat, Rep == k &
            ResultsCat[, 2] == misPercentage, c(3,
            4))
        repDataCat = data.frame(repDataCat)
        names(repDataCat) = c("Actual", "Imputed")
        error = sum(repDataCat$Imputed != repDataCat$Actual)/nrow(repDataCat)
        Kappa = cohen.kappa(repDataCat)$kappa
        AUC = pr.curve(repDataCat$Actual, repDataCat$Imputed)$auc.integral
        partialRepCat = data.frame(k, misPercentage,
            error, Kappa, AUC)
        names(partialRepCat) = c("Rep", "%Missing",
            "ClassError", "Kappa", "AUC")
        if (comb == 1) {
            RepResultsCat = partialRepCat
        } else {
            RepResultsCat = rbind(RepResultsCat, partialRepCat)
        }

        write.table(RepResultsCat, "kNNResultsCat_Sylva2.csv",
            row.names = FALSE, sep = ",")

        # Performance evaluation of numerical vars
        Yreal = NULL
        Ypred = NULL

        for (n in 1:ncol(dataNum)) {
            MissVal = which(is.na(dataNum[, n]) ==
                TRUE)
            Yreal = append(Yreal, data2Num[MissVal,
                n])
            Ypred = append(Ypred, DataImpNum[MissVal,
                n])
        }
        partialRes = data.frame(k, misPercentage, Yreal,
            Ypred)
        names(partialRes) = c("Rep", "%Missing", "Actual",
            "Imputed")
        if (comb == 1) {
            Results = partialRes
        } else {
            Results = rbind(Results, partialRes)
        }

        # save imputation data (numerical)
        write.table(Results, "kNN_ActualImpDataNum_Sylva2.csv",
            row.names = FALSE, sep = ",")

        Newtime = proc.time() - initialtime  # run time
        repData = subset(Results, Rep == k & Results[,
            2] == misPercentage, c(3, 4))
        repData = data.frame(repData)
        names(repData) = c("Actual", "Imputed")
        repData[, 1] = as.numeric(repData[, 1])
        repData[, 2] = as.numeric(repData[, 2])
        metrics = gof(sim = repData[, 2], obs = repData[,
            1])
        R2 = metrics[17]
        NRMSE = metrics[5]
        d2 = metrics[13]
        partialRep = data.frame(k, misPercentage, R2,
            NRMSE, d2, Newtime[2])
        names(partialRep) = c("Rep", "%Missing", "R2",
            "NRMSE", "d2", "RunTime")
        if (comb == 1) {
            RepResults = partialRep
        } else {
            RepResults = rbind(RepResults, partialRep)
        }
        comb = comb + 1

        write.table(RepResults, "kNNResultsNum_Sylva2.csv",
            row.names = FALSE, sep = ",")
        options(warn = 0)
    }
}

RepResults
```

C.3 Proposed Method Parameter Tunning Example Code

```r
## Proposed imputation Stopping rule DOE Update:
## Stop rule evaluating simple 1 stage, taking the
## average of 2 values evaluated (num) or Last Val
## (cat) Stage:1 Stop Crit: simple Imputation: Avg
## (num) LastVal (cat)

install.packages("randomForest")
install.packages("ForImp")
install.packages("hydroGOF")
install.packages("modeest")
install.packages("roughrf")
install.packages("psych")
install.packages("PRROC")
library(PRROC)
library(psych)
library(randomForest)
library(ForImp)
library(hydroGOF)
library(modeest)
library(roughrf)

nPerm = 30  # how many times will impute same value
nRep = 10  #how many iterations of each combination


setwd("~/")  # save at My documents

# function to calculate final value of imputation
mMfunction = function(x) {
    x = x[which(x != -9999)]
    L = length(x)
    if (typeof(x) == "double") {
        mean(c(x[(L - 1)], x[L]))
    } else {
        x[L]
    }
}

Results = NULL
partialResults = NULL
RepResults = NULL
partialRep = NULL
ResultsCat = NULL
partialResultsCat = NULL
RepResultsCat = NULL
partialRepCat = NULL

comb = 1
comb2 = 1
comb3 = 1

for (k in 1:nRep) {
    for (misPercentage in c(0.05, 0.1, 0.15, 0.2)) {
        # missing ratio in data sets

        data = read.csv(paste("~/Heizel/SimData/SimDataWITH.",
            k, ".", misPercentage, ".csv", sep = ""))  #data set with missingvalues
        data2 = read.csv(paste("~/Heizel/SimData/SimDataWITHOUT.",
            k, ".", misPercentage, ".csv", sep = ""))  #complete dataset
        impVars = read.csv("~/Feature Selection/Feature selection for proposed
          imputation/GA/Incidence Matrices/IncidenceMat_SimDataOriginal_GA.csv",
            header = FALSE)

        # Evaluate which variables are categorical and
        # numerical
        for (v in 1:ncol(data)) {
            cat = length(unique(data[, v]))
            if (cat <= 12) {
                data[, v] = as.factor(data[, v])
            } else {
                data[, v] = as.numeric(data[, v])
            }
        }

        data = as.data.frame(data)
        data2 = as.data.frame(data2)

        # Data goes here
        X = data[, -ncol(data)]
        Y = data.frame(data[, ncol(data)])
        missDF = data.frame(X, Y)
        names(missDF) = paste("X", 1:ncol(X), sep = "")
        names(missDF)[ncol(data)] = "Y"

        # Variable Imputation Order Ascending
        varMV = matrix(nrow = ncol(missDF) - 1, ncol = 1)
        for (numCol in 1:ncol(missDF) - 1) {
            # For each column, identify the number of missing
            # values
            varMV[numCol] = sum(is.na(missDF[, numCol]))
        }

        MVdf = data.frame(1:(ncol(missDF) - 1), varMV)
        names(MVdf) = c("NumCol", "varMV")
        AscOrder = order(MVdf[, ncol(MVdf)])  # How to order variables for
        # missing value imputation

        for (i in AscOrder) {
            misRows = which(is.na(missDF[, i]) == TRUE)  #ID missing values in variable
            numMis = length(misRows)
            imputedData = matrix(-9999, ncol = nPerm,
                nrow = numMis)  # matrix with j
            # imputations of observations
            imputedDataCat = matrix(-9999, ncol = nPerm,
                nrow = numMis)
```

```
imp = which(impVars[, i] == 1)  #importantn variables to be used in rF
imputedDataAux = matrix(-9999, ncol = nPerm,
    nrow = numMis)  # aux matrix with j
# imputations of observations
imputedDataAuxCat = matrix(-9999, ncol = nPerm,
    nrow = numMis)  # aux matrix with j
# imputations of observations

if (typeof(data[, i]) == "double") {

    for (iChange in c(0.025, 0.05, 0.075)) {
      imputeChange = 0.1  #initial value of numerical impute change

      for (r in 1:numMis) {
        j = 1  #impute round

        while (j <= 3 | (j <= 30 & imputeChange >
          iChange)) {
          # Variables to use in the imputation process
          if (length(imp) >= 1) {
            cols = c(i, imp, ncol(missDF))
          } else {
            cols = c(i, ncol(missDF))
          }

          impDF = na.roughfix(missDF)
          NAloc = is.na(missDF)
          obsi = !NAloc[, cols[1]]
          misi = NAloc[, cols[1]]
          obsX = impDF[obsi, cols[-1]]
          obsY = impDF[obsi, cols[1]]
          misx = impDF[misi, cols[-1]]

          rF = randomForest(y = obsY, x = obsX,
            ntree = 150, replace = TRUE)
          imputedData[, j] = predict(object = rF,
            newdata = misx, type = "response")
          imputedDataAux[r, j] = c(imputedData[r,
            j])

          if (j >= 3) {
            imputeChange = abs((imputedDataAux[r,
              j] - imputedDataAux[r, j -
              1])/(imputedDataAux[r, j -
              1]))
          } else {
            imputeChange = 0.1
          }

          if (j >= 3 & imputedDataAux[r,
            j] == -9999) {
            missDF[misRows[r], i] = imputedDataAux[r,
              j - 1]
```

```
                  }
                  j = j + 1

                }
              }
              partialRes = data.frame(k, i, misPercentage,
                iChange, cbind(data2[misRows, i],
                  apply(imputedDataAux, 1, mMfunction)))
              names(partialRes) = c("Rep", "Variable",
                "%Missing", "Impute Change", "Actual",
                "Imputed")
              if (comb == 1) {
                Results = partialRes
              } else {
                Results = rbind(Results, partialRes)
              }
              names(Results) = c("Rep", "Variable",
                "%Missing", "Impute Change", "Actual",
                "Imputed")
              write.table(Results, "StopRule_1stageSimpleAvg_ActImpNum_SimDataOrig.csv",
                row.names = FALSE, sep = ",")

              # calculate performance metrics on numeric
              # variables
              repData = subset(Results, Rep ==
                k & Results[, 3] == misPercentage &
                Results[, 4] == iChange, 5:6)
              repData = data.frame(repData)
              names(repData) = c("Actual", "Imputed")
              metrics = gof(sim = repData[, 2],
                obs = repData[, 1])
              NRMSE = metrics[5]
              R2 = metrics[17]
              d2 = metrics[13]
              partialRep = data.frame(k, i, misPercentage,
                iChange, R2, NRMSE, d2)
              names(partialRep) = c("Rep", "Variable",
                "%Missing", "ImputeChange", "R2",
                "NRMSE", "d2")
              if (comb == 1) {
                RepResults = partialRep
              } else {
                RepResults = rbind(RepResults,
                  partialRep)
              }
              write.table(RepResults, "ResultsStopRule_1stageSimpleAvgNum_SimDataOrig.csv",
                row.names = FALSE, sep = ",")
              comb = comb + 1
            }
        } else {

            for (iChangeCat in c(2, 3, 4)) {
```

```r
imputeChangeCat = 10  #initial value of categorical impute change

for (r in 1:numMis) {
  z = 1  #impute round

  while (z <= 3 | (z <= 30 & imputeChangeCat >
    iChangeCat)) {

    # Variables to use in the imputation process
    if (length(imp) >= 1) {
      cols = c(i, imp, ncol(missDF))
    } else {
      cols = c(i, ncol(missDF))
    }

    impDF = na.roughfix(missDF)
    NAloc = is.na(missDF)
    obsi = !NAloc[, cols[1]]
    misi = NAloc[, cols[1]]
    obsX = impDF[obsi, cols[-1]]
    obsY = impDF[obsi, cols[1]]
    misx = impDF[misi, cols[-1]]

    rF = randomForest(y = obsY, x = obsX,
      ntree = 150, replace = TRUE)
    imputedDataCat[, z] = predict(object = rF,
      newdata = misx, type = "response")
    imputedDataAuxCat[r, z] = c(imputedDataCat[r,
      z])

    if (z > 3) {
      if (iChangeCat == 2) {
        if (imputedDataAuxCat[r,
          z] == imputedDataAuxCat[r,
          z - 1]) {
          imputeChangeCat = 2
        } else {
          imputeChangeCat = 10
        }
      } else if (iChangeCat == 3) {
        if (imputedDataAuxCat[r,
          z] == imputedDataAuxCat[r,
          z - 1] & imputedDataAuxCat[r,
          z - 1] == imputedDataAuxCat[r,
          z - 2]) {
          imputeChangeCat = 3
        } else {
          imputeChangeCat = 10
        }
      } else {
        if (imputedDataAuxCat[r,
          z] == imputedDataAuxCat[r,
          z - 1] & imputedDataAuxCat[r,
          z - 1] == imputedDataAuxCat[r,
          z - 2] & imputedDataAuxCat[r,
          z - 2] == imputedDataAuxCat[r,
          z - 3]) {
          imputeChangeCat = 4
        } else {
          imputeChangeCat = 10
        }
      }
    }

    if (z >= 3 & imputedDataAuxCat[r,
      z] == -9999) {
      missDF[misRows[r], i] = imputedDataAuxCat[r,
        z - 1]
    }
    z = z + 1

  }
}
partialResCat = data.frame(k, i,
  misPercentage, iChangeCat, cbind(data2[misRows,
    i], apply(imputedDataAuxCat,
    1, mMfunction)))
names(partialResCat) = c("Rep", "Variable",
  "%Missing", "Impute Change", "Actual",
  "Imputed")
if (comb2 == 1) {
  ResultsCat = partialResCat
} else {
  ResultsCat = rbind(ResultsCat,
    partialResCat)
}
names(ResultsCat) = c("Rep", "Variable",
  "%Missing", "Impute Change", "Actual",
  "Imputed")
write.table(ResultsCat, "StopRule_1stageSimpleAvg_ActImpCat_SimDataOrig.csv",
  row.names = FALSE, sep = ",")

# calculate performance metrics on categorical
# variables
repDataCat = subset(ResultsCat, Rep ==
  k & ResultsCat[, 3] == misPercentage &
  ResultsCat[, 4] == iChangeCat,
  5:6)
repDataCat = data.frame(repDataCat)
names(repDataCat) = c("Actual", "Imputed")
error = sum(repDataCat$Imputed !=
  repDataCat$Actual)/nrow(repDataCat)
Kappa = cohen.kappa(repDataCat)$kappa
AUC = pr.curve(repDataCat$Actual,
  repDataCat$Imputed)$auc.integral
partialRepCat = data.frame(k, i,
                 misPercentage, iChangeCat, error,
                 Kappa, AUC)
               names(partialRepCat) = c("Rep", "Variable",
                 "%Missing", "ImputeChange", "ClassError",
                 "Kappa", "PR AUC")
               if (comb2 == 1) {
                 RepResultsCat = partialRepCat
               } else {
                 RepResultsCat = rbind(RepResultsCat,
                   partialRepCat)
               }
               comb2 = comb2 + 1
               write.table(RepResultsCat, "ResultsStopRule_1stageSimpleAvgCat_SimDataOrig.csv",
                 row.names = FALSE, sep = ",")
             }
         }  #else closes
     }
 }
}
```

C.4 Feature Selection Codes

```r
# Feature Selection with genetic algorithms

install.packages("caret")
install.packages("randomForest")
library(caret)
library(randomForest)

rep = 1
indice = 1
Results = NULL  # results matrix
partialRes = NULL
c = matrix(ncol = 1, nrow = ncol(Data))

NonLinImpVars = c("X1", "a", "b", "c")  #Relevant varibles for linear dataset
RealIV = 4  # real amount of important variables in data set
RedVars = 99  # amount of redundant variables in dataset
NoiseVars = 100  # amount of noise variables in dataset

for (rep in 1:30) {
    Data = read.csv(paste("~/Feature Selection/Data sets/SimLinearData_203_Reg/LinearRelDataset_",
        rep, ".csv", sep = ""))

    if (sum(is.na(Data)) > 0)
        {
            Data = na.roughfix(Data)
        }  # impute missing values with median/mode (in case of MV's)

    # identify constant variables and remove them from
    # data set
    for (i in 1:ncol(Data)) {
        c[i, ] = length(unique(Data[, i])) == 1
    }
    constVar = which(c == "TRUE")

    if (length(constVar > 0)) {
        Data = Data[, -constVar]
    } else {
        Data = Data
    }

    names(Data)[1:ncol(Data)] = paste("X", 1:ncol(Data),
        sep = "")  #name variables

    DataX = Data[, 1:(ncol(Data) - 1)]  # x variables data frame
    Y = Data[, ncol(Data)]  # y

    ga.Data = gafs(x = DataX, y = Y, iters = 2, gafsControl = gafsControl(functions = rfGA,
        method = "cv", number = 5), ntree = 125)  #feature
    # selection using GA

    # performance metrics for fs method itself
    impVNS = setdiff(NonLinImpVars, ga.Data$ga$final)  #important variables
    # not selected by CFS
    impVD = length(NonLinImpVars) - length(impVNS)  #important variables
    # accurately detected
    ExcessVars = length(ga.Data$ga$final) - impVD

    Sens1 = impVD/(RealIV + RedVars)  #sensitivity
    Sens2 = impVD/(RealIV)  #sensitivity
    Specif = 1 - (ExcessVars/(NoiseVars + RedVars))  #specificity
    Accuracy = (impVD + (NoiseVars + RedVars - ExcessVars))/(length(Data) -
        1)  #accuracy

    partialRes = data.frame(rep, length(ga.Data$ga$final),
        paste(ga.Data$ga$final, collapse = ";"), Sens1,
        Sens2, Specif, Accuracy, ga.Data$times$everything[2],
        ga.Data$averages$RMSE, ga.Data$averages$Rsquared)
    names(partialRes) = c("Rep", "BestSubsetSize",
        "ImpVariables", "Sensitivy1", "Sensitivy2",
        "Specificity", "Accuracy", "RunTime", "RMSE",
        "R2")
    if (indice == 1) {
        Results = partialRes
    } else {
        Results = rbind(Results, partialRes)
    }
    write.table(Results, "GAResults_LinearData.csv",
        sep = ",", row.names = FALSE)
    indice = indice + 1

}
```

```r
# CFS code for Classification data sets: EPR,
# Heart, Gina, Sylva

install.packages("stringr")
install.packages("randomForest")
install.packages("FSelector")
library(FSelector)
library(stringr)
library(randomForest)

nFolds = 5  # folds for CV
rep = 1
indice = 1

Results = matrix(nrow = 1, ncol = 6)  # results matrix
Results = data.frame(Results)

for (rep in 1:30) {
    Data = read.csv(paste("~/Feature Selection/Data sets/BreastCancer/BreastCancerDataset_",
        rep, ".csv", sep = ""))
    # change depending on data set

    names(Data)[ncol(Data)] = "Y"
    if (sum(is.na(Data)) > 0)
        {
            Data = na.roughfix(Data)
        }  # impute missing values with
    # median/mode (in case of MV's)

    initialtime = proc.time()

    impVars = cfs(Y ~ ., Data)  # perform feature selection
    impVarsAux = paste(impVars, collapse = ";")

    # performance metric for fs method itself
    BestSubset = length(impVars)  #best subset size


    # Cross validation (uses random forest)
    impVarNum = which(names(Data) %in% c(impVars))  # extract column numbers from impvars
    DataNew = data.frame(Data[, impVarNum], as.factor(Data$Y))  # new data set created
    # with the features selected and Y
    names(DataNew)[ncol(DataNew)] = "Y"

    permRows = sample(x = 1:nrow(DataNew), size = nrow(DataNew),
        replace = FALSE)
    error = matrix(nrow = nFolds, ncol = 1)  #CV error matrix
    acc = matrix(nrow = nFolds, ncol = 1)  # accuracy matrix

    # Create testing and training folds
    obsFold = floor(nrow(DataNew)/nFolds)
    pending = nrow(DataNew) - floor(nrow(DataNew)/nFolds) *
        nFolds
    j = 0

    for (i in 1:nFolds) {
        if (i >= (nFolds - pending + 1) & pending >
            0) {
            assign(paste("F", i, sep = ""), DataNew[permRows[(j +
                1):(j + obsFold)], ])
            j = j + obsFold + 1
        } else {
            assign(paste("F", i, sep = ""), DataNew[permRows[(j +
                1):(j + obsFold)], ])
            j = j + obsFold
        }
    }

    # Fit model
    for (i in 1:nFolds) {
        testing = get(paste("F", i, sep = ""))
        trainingRows = setdiff(1:nrow(DataNew), as.numeric(row.names(testing)))
        training = DataNew[trainingRows, ]

        myRF = randomForest(Y ~ ., data = training)  #fit random forest using new data
        # set with selected features
        predicted = predict(myRF, newdata = testing)
        actual = testing$Y

        # performance metrics
        error[i, ] = sum(actual != predicted)/nrow(testing)  # CV error
        acc[i, ] = 1 - error[i, ]  #Accuracy
    }

    CVerror = mean(error)
    Accuracy = mean(acc)


    Newtime = proc.time() - initialtime  # run time

    Results[indice, ] = cbind(rep, impVarsAux, BestSubset,
        Newtime[2], CVerror, Accuracy)
    names(Results) = c("Rep", "ImpVariables", "BestSubset Size",
        "RunTime", "CV Error", "Accuracy")
    write.table(Results, "CFS_Results_BreastCancer.csv",
        sep = ",", row.names = FALSE)  #Change Doc
    # name depending on data set
    indice = indice + 1

}
Results
```

```r
# VSURF-Variable selection using random forest

install.packages("VSURF")
install.packages("randomForest")
library(VSURF)
library(randomForest)

nFolds = 5  # folds for CV
rep = 1
indice = 1
c = matrix(ncol = 1, nrow = ncol(Data))

NonLinImpVars = c(1:4)  #Relevant varibles for linear dataset
RealIV = 4  # real amount of important variables in data set
RedVars = 99  # amount of redundant variables in dataset
NoiseVars = 100  # amount of noise variables in dataset

Results = matrix(nrow = 1, ncol = 10)  # results matrix
Results = data.frame(Results)

for (rep in 1:30) {

    Data = read.csv(paste("~/Feature Selection/Data sets/SimLinearData/LinearRelDataset_",
        rep, ".csv", sep = ""))

    initialtime = proc.time()

    if (sum(is.na(Data)) > 0)
        {
            Data = na.roughfix(Data)
        }  # impute missing values with median/mode (in case of MV's)

    # identify constant variables and remove them from
    # data set
    for (i in 1:ncol(Data)) {
        c[i, ] = length(unique(Data[, i])) == 1
    }
    constVar = which(c == "TRUE")

    if (length(constVar > 0)) {
        Data = Data[, -constVar]
    } else {
        Data = Data
    }

    names(Data)[ncol(Data)] = "Y"


    DataX = Data[, 1:(ncol(Data) - 1)]
    DataY = Data$Y

    vsurf.Data = VSURF(x = DataX, y = DataY, ntree = 100,
        nfor.thres = 20, nfor.interp = 10, nfor.pred = 10)
```

```
impVars = paste("X", vsurf.Data$varselect.pred,
    sep = "", collapse = ";")

# performance metric for fs method itself
BestSubset = length(vsurf.Data$varselect.pred)  #best subset size

# performance metrics for fs method itself
impVNS = setdiff(NonLinImpVars, vsurf.Data$varselect.pred)  #important
# variables not selected
impVD = length(NonLinImpVars) - length(impVNS)  #important variables accurately detected
ExcessVars = BestSubset - impVD

Sens1 = impVD/(RealIV + RedVars)  #sensitivity
Sens2 = impVD/(RealIV)  #sensitivity
Specif = 1 - (ExcessVars/(NoiseVars + RedVars))  #specificity
Accuracy = (impVD + (NoiseVars + RedVars - ExcessVars))/(length(Data) -
    1)  #accuracy


# Cross validation (uses random forest)
impVarNum = vsurf.Data$varselect.pred
DataNew = data.frame(Data[, impVarNum], Data$Y)  # new data set created with the
# features selected and Y
names(DataNew)[ncol(DataNew)] = "Y"

permRows = sample(x = 1:nrow(DataNew), size = nrow(DataNew),
    replace = FALSE)
error = matrix(nrow = nFolds, ncol = 1)  #CV error matrix
mean_adev = matrix(nrow = nFolds, ncol = 1)  # av dev matrix

# Create testing and training folds
obsFold = floor(nrow(DataNew)/nFolds)
pending = nrow(DataNew) - floor(nrow(DataNew)/nFolds) *
    nFolds
j = 0

for (i in 1:nFolds) {
    if (i >= (nFolds - pending + 1) & pending >
        0) {
        assign(paste("F", i, sep = ""), DataNew[permRows[(j +
            1):(j + obsFold)], ])
        j = j + obsFold + 1
    } else {
        assign(paste("F", i, sep = ""), DataNew[permRows[(j +
            1):(j + obsFold)], ])
        j = j + obsFold
    }
}

# Fit model
for (i in 1:nFolds) {
    testing = get(paste("F", i, sep = ""))
    trainingRows = setdiff(1:nrow(DataNew), as.numeric(row.names(testing)))
```

```
        training = DataNew[trainingRows, ]

        myRF = randomForest(Y ~ ., data = training)  #fit random forest using new
        # data set with selected features
        predicted = predict(myRF, newdata = testing)
        actual = testing$Y

        # performance metrics

        error[i, ] = sum((actual - predicted)^2)  # PRESS
        mean_adev[i, ] = sum(abs(actual - predicted)/length(predicted))  #MAD
    }

    PRESS = mean(error)
    MAD = mean(mean_adev)


    Newtime = proc.time() - initialtime  # run time

    Results[indice, ] = cbind(rep, impVarsAux, BestSubset,
        Sens1, Sens2, Specif, Accuracy, Newtime[2],
        PRESS, MAD)
    names(Results) = c("Rep", "ImpVariables", "BestSubset Size",
        "Sensitivy1", "Sensitivy2", "Specificity",
        "Accuracy", "RunTime", "PRESS CV", "MAD CV")
    write.table(Results, "VSurfResults_LinearData.csv",
        sep = ",", row.names = FALSE)
    # Change Doc name depending on data set!!!
    indice = indice + 1

}
Results
```

```r
# ACE feature selection using random forest to get importance score

install.packages("randomForest")
install.packages("stringr")
library(randomForest)
library(stringr)

nFolds=5 # folds for CV
qArtificial=0.9
nPerm=30
nTrees=125
rep=1

Results=matrix(nrow = 1,ncol = 10) #results matrix
Results=data.frame(Results)
indice=1

q=NULL
impor=NULL
dfPVAL=NULL
impVars=NULL
impVarsAux=NULL


NonLinImpVars=paste("X",1:15, sep = "") #Relevant varibles for linear dataset
RealIV=15 # real amount of important variables in data set
RedVars=30 # amount of redundant variables in dataset
NoiseVars=100 # amount of noise variables in dataset


for (rep in 1:30 )
{
  data=read.csv(paste("~/Feature Selection/Data sets/SimLinear_145_class/
                      LinearRelDataset_Class145_",rep,".csv", sep=""))

  names(data)[ncol(data)]="Y"

  if (sum(is.na(data))>0){data=na.roughfix(data)}
  nVar=ncol(data)-1

  initialtime=proc.time()

  impor=matrix(nrow=(nVar*2),ncol=nPerm)
  q=matrix(nrow=nPerm,ncol=1)
  pval=matrix(nrow=nVar,ncol=1)
  pval=data.frame(pval)

  for (i in 1:nPerm)
  {

    X = data.frame(matrix(nrow = nrow(data),ncol = (2 * nVar)))
    X[,(1:nVar)] = data[,(1:nVar)]
    for (j in 1:nVar)# Artificial variables

    {
      X[,nVar + j] = sample(X[,j],length(X[,j]),replace = FALSE)
    }

    data2=cbind(X,as.factor(data$Y)) # New data frame with original
    #Xs, artificial Xs, and Y at the end
    names(data2)[ncol(data2)]="Y"

    if (typeof(data2$Y)=="double"){rF=randomForest(Y~.,data=data2,ntree=nTrees,
                    importance=TRUE,replace=FALSE,na.action=na.roughfix)
    }else{rF=randomForest(as.factor(Y)~.,data=data2,ntree=nTrees,
                    importance=TRUE,replace=FALSE)}
    impor[,i]=cbind(rF$importance[,ncol(rF$importance)]) # Gini/IncNodePurity
    q[i]=quantile(impor[(nVar+1):(2*nVar),i],probs=qArtificial)
  }

  for (w in 1:nVar)
  {
    test=wilcox.test(x=cbind(impor[w,]),y=cbind(q),alternative="greater",
                     paired=TRUE,conf.level=0.99)
    pval[w,]<-test$p.value
  }

  dfPVAL=cbind(names(X)[1:nVar],pval)

  if (length(which(pval<(0.05/nVar)))>=1){
    impVars=subset(dfPVAL,pval<(0.05/nVar),1)[,1]
    impVarsAux=paste(subset(dfPVAL,pval<(0.05/nVar),1)[,1],sep="",
                     collapse=";")} else {impVars=""
                     impVarsAux=""}

  impVNS= setdiff(NonLinImpVars,impVars) #important variables not selected
  impVD=length(NonLinImpVars)-length(impVNS) #important variables accurately detected
  ExcessVars=length(impVars)-impVD

  Sens1=impVD/(RealIV+RedVars) #sensitivity
  Sens2=impVD/(RealIV) #sensitivity
  Specif=1-(ExcessVars/(NoiseVars+RedVars)) #specificity
  Accuracy=(impVD+(NoiseVars+RedVars-ExcessVars))/(length(data)-1) #accuracy
  BestSubset=length(impVars) #best subset size

  #Cross validation (uses random forest)
  impVarNum=as.numeric(str_extract(impVars, "[[:digit:]]+")) # extract
  #numbers from impvars
  DataNew=data.frame(data[,impVarNum],as.factor(data$Y)) # new data set created
  #with the features selected and Y
  names(DataNew)[ncol(DataNew)]="Y"

  permRows=sample(x=1:nrow(DataNew),size=nrow(DataNew),replace=FALSE)
  error=matrix(nrow=nFolds,ncol=1) #cv error matrix
  acc=matrix(nrow=nFolds,ncol=1) # accuracy matrix
```

```
  # Create testing and training folds
  obsFold=floor(nrow(DataNew)/nFolds)
  pending=nrow(DataNew)-floor(nrow(DataNew)/nFolds)*nFolds
  j=0

  for (i in 1:nFolds){
    if (i>=(nFolds-pending+1) & pending>0) {
      assign(paste("F",i,sep=""),DataNew[permRows[(j+1):(j+obsFold)],]) ; j= j + obsFold + 1 }
    else
    { assign(paste("F",i,sep=""),DataNew[permRows[(j+1):(j+obsFold)],]); j= j + obsFold }
  }

  #Fit model
  for (i in 1:nFolds){
    testing=get(paste("F",i,sep=""))
    trainingRows=setdiff(1:nrow(DataNew),as.numeric(row.names(testing)))
    training=DataNew[trainingRows,]

    myRF=randomForest(Y~., data=training) #fit random forest using new data
    #set with selected features
    predicted=predict(myRF,newdata=testing)
    actual=testing$Y

    #performance metrics

    error[i,]=sum(actual!=predicted)/nrow(testing) # CV error
    acc[i,]=1-error[i,] #Accuracy
  }

  CVerror=mean(error)
  Accuracy=mean(acc)

  Newtime=proc.time() - initialtime # medida de tiempo de corrida

  Results[indice,]=cbind(rep,impVarsAux,BestSubset,Sens1,Sens2,Specif,Accuracy,
                         Newtime[2],CVerror,Accuracy)
  names(Results)=c("Rep","ImpVariables","BestSubsetSize",
        "Sensitivy1","Sensitivy2","Specificity","Accuracy","RunTime", "CV Error",
        "CV Accuracy")
  write.table(Results,"ACErf_Results_SimLinDataClass.csv",sep=",",row.names = FALSE)
  #Change Doc name depending on data set
  indice=indice+1

}
Results
```

```r
# Relief code for simulated linear data set

install.packages("stringr")
install.packages("randomForest")
install.packages("FSelector")
library(FSelector)
library(stringr)
library(randomForest)


nFolds = 5  # folds for CV
rep = 1
indice = 1
c = matrix(ncol = 1, nrow = ncol(Data))

NonLinImpVars = c(1:4)  #Relevant varibles for linear dataset
RealIV = 4  # real amount of important variables in data set
RedVars = 99  # amount of redundant variables in dataset
NoiseVars = 100  # amount of noise variables in dataset

Results = matrix(nrow = 1, ncol = 10)  # results matrix
Results = data.frame(Results)

for (rep in 1:30) {
    BestSubset = NULL
    Data = read.csv(paste("~/Feature Selection/Data sets/SimLinearData/LinearRelDataset_",
        rep, ".csv", sep = ""))

    if (sum(is.na(Data)) > 0)
        {
            Data = na.roughfix(Data)
        }  # impute missing values with median/mode (in case of MV's)

    initialtime = proc.time()

    # identify constant variables and remove them from
    # data set
    for (i in 1:ncol(Data)) {
        c[i, ] = length(unique(Data[, i])) == 1
    }
    constVar = which(c == "TRUE")

    if (length(constVar > 0)) {
        Data = Data[, -constVar]
    } else {
        Data = Data
    }

    impVars = relief(Y ~ ., data = Data, neighbours.count = 10,
        sample.size = round(0.05 * (nrow(Data))))
    # plot(impVars[order(-impVars$attr_importance),])
    df = data.frame(impVars, 1:nrow(impVars))
    names(df) = c("AtrrImp", "Var")
```

```
ord_df = df[order(-df$AtrrImp), ]
ord = ord_df[, 1]

PerChange = 0.1
i = 1
while (PerChange <= 0.3) {
    # le cambie porque algunas corridas no estaban
    # dando el %
    i = i + 1
    PerChange = abs((ord[i - 1] - ord[i])/ord[i -
        1])
}

selVars = ord_df$Var[c(1:(i - 1))]

# performance metric for fs method itself
SelimpVars = paste("X", c(selVars), sep = "", collapse = ";")
BestSubset = length(selVars)  #best subset size

# performance metrics for fs method itself
impVNS = setdiff(NonLinImpVars, selVars)  #imp variables not selected by CFS
impVD = length(NonLinImpVars) - length(impVNS)  #imp variables accurately detected
ExcessVars = length(selVars) - impVD

Sens1 = impVD/(RealIV + RedVars)  #sensitivity
Sens2 = impVD/(RealIV)  #sensitivity
Specif = 1 - (ExcessVars/(NoiseVars + RedVars))  #specificity
Accuracy = (impVD + (NoiseVars + RedVars - ExcessVars))/(length(Data) -
    1)  #accuracy

# Cross validation (uses random forest)
DataNew = data.frame(Data[, selVars], Data$Y)  # new data set created with
# the features selected and Y
names(DataNew)[ncol(DataNew)] = "Y"

permRows = sample(x = 1:nrow(DataNew), size = nrow(DataNew),
    replace = FALSE)
error = matrix(nrow = nFolds, ncol = 1)  #CV error matrix
mean_adev = matrix(nrow = nFolds, ncol = 1)  # mean deviation matrix

# Create testing and training folds
obsFold = floor(nrow(DataNew)/nFolds)
pending = nrow(DataNew) - floor(nrow(DataNew)/nFolds) *
    nFolds
j = 0

for (i in 1:nFolds) {
    if (i >= (nFolds - pending + 1) & pending >
        0) {
        assign(paste("F", i, sep = ""), DataNew[permRows[(j +
            1):(j + obsFold)], ])
        j = j + obsFold + 1
    } else {
```

```r
            assign(paste("F", i, sep = ""), DataNew[permRows[(j +
                1):(j + obsFold)], ])
            j = j + obsFold
        }
    }

    # Fit model
    for (i in 1:nFolds) {
        testing = get(paste("F", i, sep = ""))
        trainingRows = setdiff(1:nrow(DataNew), as.numeric(row.names(testing)))
        training = DataNew[trainingRows, ]

        myRF = randomForest(Y ~ ., data = training)  #fit random forest using new
        # data set with selected features
        predicted = predict(myRF, newdata = testing)
        actual = testing$Y

        # performance metrics

        error[i, ] = sum((actual - predicted)^2)  # PRESS
        mean_adev[i, ] = sum(abs(actual - predicted)/length(predicted))  #MAD
    }

    PRESS = mean(error)
    MAD = mean(mean_adev)

    Newtime = proc.time() - initialtime  # run time

    Results[indice, ] = cbind(rep, impVarsAux, BestSubset,
        Sens1, Sens2, Specif, Accuracy, Newtime[2],
        PRESS, MAD)
    names(Results) = c("Rep", "ImpVariables", "BestSubset Size",
        "Sensitivy1", "Sensitivy2", "Specificity",
        "Accuracy", "RunTime", "PRESS CV", "MAD CV")
    write.table(Results, "ReliefResults_LinearData.csv",
        sep = ",", row.names = FALSE)  #Change Doc name depending on data set!!!
    indice = indice + 1

}
Results
```

REFERENCES

Andridge, R. R. and Little, R. J. A. (2010). A review of hot deck imputation for survey non-response. *International statistical review = Revue internationale de statistique*, 78(1):40–64.

Bland, J. M. and Altman, D. G. (1995). Multiple significance tests: the Bonferroni method. *BMJ*, 310(6973):170.

Blum, A. and Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97:245–271.

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

Breiman, L., Friedman, J., Stone, C., and Olshen, R. (1984). *Classification and regression trees.*

Brownlee, J. (2014). An Introduction to Feature Selection.

Burgette, L. F. and Reiter, J. P. (2010). Multiple imputation for missing data via sequential regression trees. *American Journal of Epidemiology*, 172(9):1070–1076.

Carriquiry, A. (2004). Regression inference.

Chandrashekar, G. and Sahin, F. (2014). A survey on feature selection methods. *Computers and Electrical Engineering*, 40:16–28.

Comulada, W. S. (2015). Model specification and bootstrapping for multiply imputed data: An application to count models for the frequency of alcohol use. *The Stata journal*, 15(3):833–844.

Doshi, M. and Chaturvedi (2014). Correlation based feature selection (cfs) technique to predict student perfromance. *International Journal of Computer Networks & Communications*, 6(3).

Enders, C. K. (2010). *Applied Missing Data Analysis.* The Guiford Press, New York, NY, USA.

Fawcett, T. (2003). ROC Graphs: Nos and Practical Considerations for Data Mining Researchers. *HP Laboratories.*

Friedman, J. (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232.

Gelman, A. and Hill, J. (2006). Missing-data imputation. In *Data Analysis Using Regression and Multilevel/Hierarchical Models*, pages 529–544. Cambridge University Press. Cambridge Books Online.

Genuer, R., Poggi, J., and Tuleau-Malot, C. (2015a). VSURF: An R Package for Variable Selection Using Random Forests. *The R Journal*, 7(2).

Genuer, R., Poggi, J., and Tuleau-Malot, C. (2015b). VSURF: An R Package for Variable Selection Using Random Forests. *The R Journal*, 7(2).

Gould, J. (2000). Classification and regression trees (cart) documentation.

Gower, J. C. (1971). A General Coefficient of Similarity and Some of Its Properties. *Biometrics*, 27(4):857–871.

Grau, J. and Keilwagen, J. (2015). Package "PRROC".

Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182.

Hall, M. A. (2000). Correlation-based feature selection for discrete and numeric class machine learning. In *Proceedings of the Seventeenth International Conference on*

Machine Learning, ICML '00, pages 359–366, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Holmes, W. (2010). Imputation methods for missing categorical questionnaire data: A comparison of approaches. *Journal of Data Science*, Holmes8:361–378.

Islam, Z. and Giggins, H. (2011). Knowledge discovery through sysfor: A systematically developed forest of multiple decision trees. In *Proceedings of the Ninth Australasian Data Mining Conference - Volume 121*, AusDM '11, pages 195–204, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

Janosi, A., Steinbrunn, W., Pfisterer, M., and Detrano, R. (1988). UCI machine learning repository.

Jonsson, P. and Wohlin, C. (2004). An evaluation of k-nearest neighbour imputation using likert data. In *10th International Symposium on Software Metrics, 2004. Proceedings*, pages 108–118.

Kaggle, K. (2017). Root Mean Squared Error.

Kaiser, J. (2014). Dealing with missing values in data. *JOURNAL OF SYSTEMS INTEGRATION*, 1.

Kalton, G. and Kasprzyk, D. (1982). Imputing for missing survey responses. *Proceedings of the survey research methods section, American Statistical Association*, pages 22–31.

Kira, K. and Rendell, L. (1992). The feature selection problem: Traditional methods and a new algorithm. *AAAI Proceedings*.

Kohavi, R. and John, G. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, 97:273–324.

Kowarik, A. and Templ, M. (2016). Imputation with the R package VIM. *Journal of Statistical Software*, 74(7).

Kuhn, M. (2016). Package "caret".

Langley, P. and Iba, W. (1993). Average-case analysis of a nearest neighbor algorthim. In *Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 2*, IJCAI'93, pages 889–894, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Liao, S. G., Lin, Y., Kang, D. D., Chandra, D., Bon, J., Kaminski, N., Sciurba, F. C., and Tseng, G. C. (2014). Missing value imputation in high-dimensional phenomic data: imputable or not, and how? *BMC Bioinformatics*, 15:346.

Liaw, A. and Wiener, M. (2014). RandomForest in R. Technical report.

Lichman, M. (2013). UCI machine learning repository.

Little, R. J. and Rubin, D. B. (1986). *Statistical Analysis with Missing Data*. John Wiley & Sons, Inc., New York, NY, USA.

Little, R. J. A. and Schluchter, M. D. (1985). Maximum Likelihood Estimation for Mixed Continuous and Categorical Data with Missing Values. *Biometrika*, 72(3):497–512.

López, V. (2005). *Comparación de los métodos de imputación con respecto al poder de separación del modelo de regresión logistíca*. PhD thesis, University of Puerto Rico.

Mohamad, M., Deris, S., and Razib, M. (2004). FEATURE SELECTION METHOD USING GENETIC ALGORITHM FOR THE CLASSIFCATION OF SMALL AND HIGH DIMENSION DATA. *First Internation Symposium on Information and Communications Technologies*.

Pantanowitz, A. and Marwala, T. (2009). Missing data imputation through the use of the random forest algorithm. In Kacprzyk, J., Yu, W., and Sanchez, E. N., editors, *Advances in Computational Intelligence*, volume 116, pages 53–62. Springer Berlin Heidelberg, Berlin, Heidelberg.

Penny, K. I. and Atkinson, I. (2012). Approaches for dealing with missing data in health care studies. *Journal of Clinical Nursing*, 21(19-20):2722–2729.

R Core Team (2016a). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

R Core Team (2016b). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Rahman, G. and Islam, Z. (2011). A decision tree-based missing value imputation technique for data pre-processing. In *Proceedings of the Ninth Australasian Data Mining Conference - Volume 121*, AusDM '11, pages 41–50, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

Rahman, M. G. and Islam, M. Z. (2013). Missing value imputation using decision trees and decision forests by splitting and merging records: Two novel techniques. *Knowledge-Based Systems*, 53:51–65.

Revelle, W. (2016). Package "psych".

Robnik-Sikonja, M. and Kononenko, I. (2003). Theoretical and Empirical Analysis of ReliefF and RReliefF. *Machine Learning Journal*, (53):23–69.

Rogier, A., Donders, T., van der Heijden, G., and Stijnend, T. (2006). Review: A gentle introduction to imputation of missing values. *Journal of Clinical Epidemiology*, 59:1087e1091.

Romanski, P. and Kotthoff, L. (2016). F Selector R.

Rubin, D. (1976). Inference and missing data. *Biometrika*, 63(3):581–592.

Saeys, Y., Inza, I. n., and Larrañaga, P. (2007). A review of feature selection techniques in bioinformatics. *Bioinformatics*.

Schafer, J. L. (1997). *Analysis of Incomplete Multivariate Data*. CRC Press.

Schneider, T. (2001). Analysis of Incomplete Climate Data: Estimation of Mean Values and Covariance Matrices and Imputation of Missing Values. *Journal of Climate*, 14.

Sertkaya, A., Birkenbach, A., Berlind, A., and Eyraud, J. (2014). Examination of clinical trial costs and barriers for drug development. Technical report, Department of Health and Human Services.

Service, U. F. (2006). SYLVA.

Shardlow, M. (2008). An analysis of feature selection techniques.

Sharp, T., Lengerich, R., and Bai, S. (2017). 18.7 - Cohen's Kappa Statistic for Measuring Agreement | STAT 509.

Silva, E., Pino, R., Lopez, M., and Cubiles, M. (2011). Missing value imputation on missing completely at random data using multilayer perceptrons. *Neural Networks*, 24:121–129.

Sim, J., Lee, J. S., and Kwon, O. (2015). Missing Values and Optimal Selection of an Imputation Method and Classification Algorithm to Improve the Accuracy of Ubiquitous Computing Applications. *Mathematical Problems in Engineering*, 2015.

Singh, A., Yadav, A., and Rana, A. (2013). K-means with Three different Distance Metrics. *International Journal of Computer Applications*, 67(10).

Staff, M. C. (2014). Heart disease.

Stekhoven, D. (2013). Package "missForest".

Stekhoven, D. J. and Bülmann, P. (2012). Missforest-non-parametric missing value imputation for mixed-type data. *Bioinformatics*, 28(1):112–118.

Sulis, I. and Porcu, M. (2008). Assessing the Effectiveness of a Stochastic Regression Imputation Method for Ordered Categorical Data.

Taylor, C. (2016). Learn More About Chebyshev's Inequality in Probability.

Tuv, E., Borisov, A., Runger, G., and Torkkola, K. (2009). Feature selection with ensembles, artificial variables, and redundancy elimination. *J. Mach. Learn. Res.*, 10:1341–1366.

Wild, C. (2011). Wilcoxon Handout.

William, W. (1992). UCI machine learning repository.

Willmott, C. J. (1981). On the validation of models. *Physical Geography*, 2(2):184–194.

Wolberg, W. H. and Mangasarian, O. L. (1990). Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the National Academy of Sciences of the United States of America*, 87(23):9193–9196.

Wood, A. M., White, I. R., and Thompson, S. G. (2004). Are missing outcome data adequately handled? a review of published randomized controlled trials in major medical journals. *Clinical Trials (London, England)*, 1(4):368–376.

Yeşilova, A., Kaya, Y., and Almali, N. (2010). A comparison of hot deck imputation and substitution methods in the estimation of missing data. *Gazi University Journal of Science*, 24(1):69–75.

Yu, L. and Liu, H. (2007). Feature selection for high-dimensional data: A fast correlation-based filter solution. Technical report, Arizona State University.

Zambrano (2014). Package "hydroGOF".