Master's Thesis

Feature Ranking for Incomplete Datasets by Modeling the Uncertainty of Missing Values

Feature Ranking für unvollständige Datensätze durch Modellierung der Unsicherheit von fehlenden Werten

Sebastian Rehfeldt

sebastian.rehfeldt@student.hpi.de

Submitted on October 22, 2018

Knowledge Discovery and Data Mining Group

Supervision:
Thomas Goerttler
Arvind Kumar Shekar
Prof. Dr. Emmanuel Müller

Abstract

Data mining describes the discovery of hidden patterns in large datasets with the goals of getting a deeper understanding of the data and creating models, e.g. to predict values or classes of unseen instances. Due to reduced costs, more sensor outputs, called features, are collected nowadays. However, not all of them are highly relevant to the model and can even lead to inferior predictions. Additionally, many real-world datasets suffer from missing values due to failing sensors or erroneous data processing pipelines. Missing values pose a severe challenge to many data mining tasks as standard algorithms cannot be applied anymore. Besides few works on classification, a dedicated preprocessing step is typically added to handle missing values. Deletion removes all incomplete samples whereas imputation replaces missing values with promising estimates. Deletion thus makes poor use of the available data while imputation can introduce a bias to the data and leads to notable errors when the estimated values differ significantly from their real counterparts. Multiple imputation models the uncertainty of missing values by repeated estimations. However, this uncertainty gets usually lost before applying further data mining tasks by replacing each missing value with its average estimation.

We propose an alternative in which we model the probability of a missing value to fall into a specific value range rather than estimating a fixed value. We implement this idea into a novel filter-based feature ranking framework called Relevance and Redundancy. RaR measures conditional dependencies between randomly drawn subspaces and the class to obtain relevance scores. Therefore, RaR gathers empirical estimations for conditional distributions by creating conditions on a subspace and finally measuring divergences to the marginal distribution. In this work, we introduce a weight function which enables the application of RaR on incomplete datasets by estimating probabilities of missing values to fulfill these conditions. We further propose an active sampling approach which improves both performance and quality of RaR and finally provide a general parameter setting which applies to datasets of different characteristics.

We perform extensive experiments on feature selection for incomplete datasets including synthetic and real-world data, a wide range of missing rates and missing mechanisms as well as an extensive collection of competitor methods. To the best of our knowledge, this is the first comprehensive comparison of feature selection methods on incomplete datasets which also examines the impact of imputation. The experimental evaluation demonstrates that the benefits of RaR on complete datasets are also notable on incomplete datasets and that RaR outperforms several state-of-the-art feature selection methods as well as imputation techniques. In our experiments, RaR is the only method which efficiently creates high-quality rankings on datasets containing missing values and mixed feature types while incorporating relevance, high order interactions, and redundancies among features. Finally, we conclude that feature selection is also effective in eliminating irrelevant features in incomplete datasets which leads to superior classification performances and more accurate imputations.

Contents

1 Introduction

Data mining is the process of extracting knowledge and hidden structures from datasets with the ultimate goal of turning the information into action. A common task of data mining is to match the discovered structures to unseen instances in order to estimate their classes or values, termed classification and regression. A dataset contains patterns which resemble objects, processes or events sharing the same attributes. An attribute, commonly defined as a feature, can be the output of a sensor to a specific time, the answer to a question in a survey or the result of a medical test. Over the past years, the number of features steadily increased and it is prevalent that not all of them are helpful for the decision making. Thus a smaller set of features can contain the same information and predictive power regarding the classification or regression task. Feature selection frequently describes the process of identifying smaller sets of features which are highly relevant to the model and non-redundant to each other.

Feature selection usually improves model accuracies while at the same time reducing complexity and promoting interpretability. Thus it has become a crucial part of data mining pipelines in many domains ranging from text classification to bioinformatics. [58] Datasets in the field of bioinformatics are characterized by an enormous number of features and a small number of samples. For example, one could be interested in which genes are responsible for causing cancer. Therefore it is necessary to measure gene expressions of healthy and sick patients which is very costly for many patients. When trying to fit classification models on these so-called "large-p-small-n" problems, modern approaches such as neural networks and deep learning cannot be applied as there are not enough samples to learn from. Also, many classical models suffer from high dimensional data as they cannot efficiently be applied, overfit the data or fall prey to the curse of dimensionality. Thus, techniques for reducing the dimensionality are required. In comparison to feature extraction techniques like principal component analysis, feature selection does not alter the data and comes with the advantage of interpretability. [34] In our cancer example, doctors are primarily interested in relevant genes rather than the best classification results. These genes could then be subject to follow up studies.

Medical datasets are also challenging as they frequently contain missing values. Reasons for the missingness are numerous including the non-applicability of a medical test to a patient, the dysfunction of equipment or too high costs for a test. Missing values are present in other domains as well, e.g. people do not want to answer personal questions in a survey, sensors fail due to power outages or data gets lost in complicated processing and transformation pipelines. In practice, 45% of all datasets in the UCI machine learning repository contain missing values [1, 23] and even in industrial datasets up to half of all values can be missing. [22] One of the severest problems with having missing values is that traditional data mining approaches cannot be applied anymore even though a few missing values do not mean a significant loss of information. [20] Many established feature selection methods rely on distances between instances which cannot be computed anymore when values are missing. Statistical measures and distributions can still be estimated on observed data but can lead to wrong estimations when the distribution behind missing data differs from the distribution of observed data.

During this work we use datasets hosted on the OpenML platform. [69] OpenML comes with the advantage over UCI that it provides an easy-to-use API for deriving statistics as well as for downloading datasets in different formats, also including feature types. The statistics

are used to identify realistic use cases for missing value aware data mining. First, the missing rate according to dataset dimensions is visualized in Figure 1.

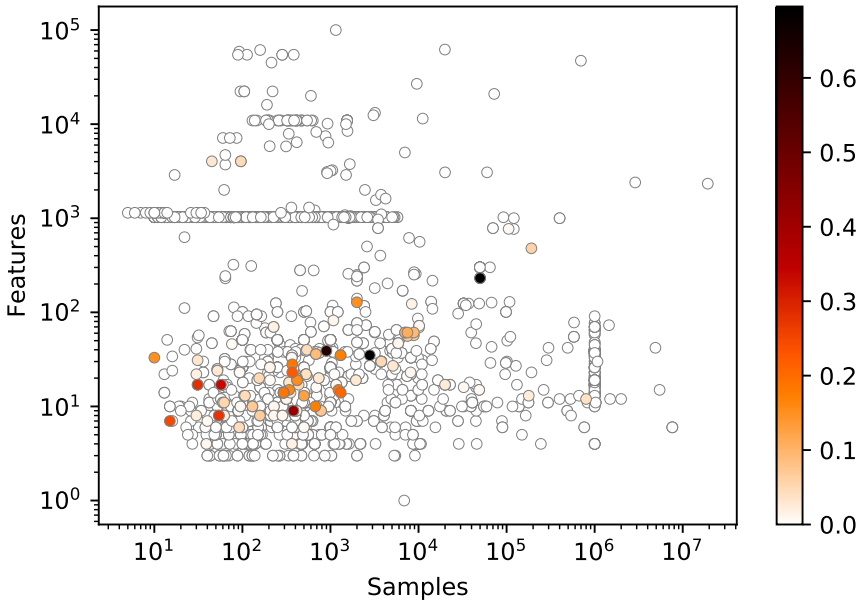
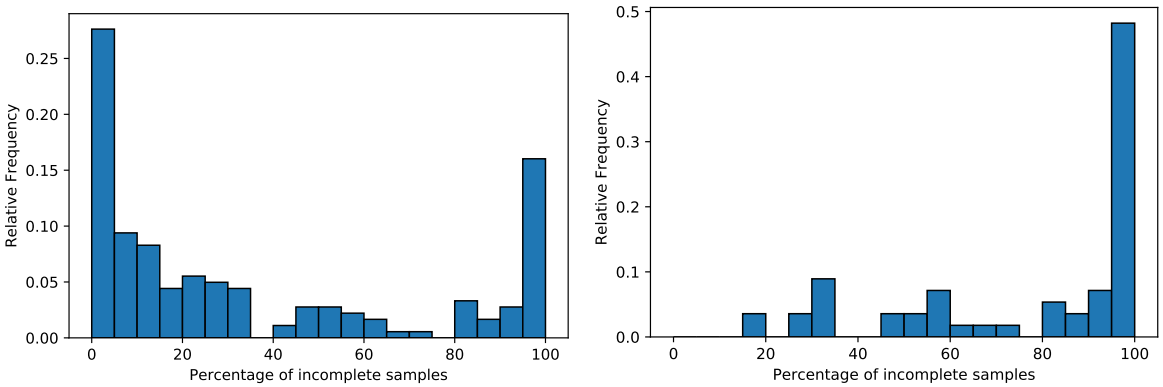


Figure 1: Missing rate by dataset dimensions

According to the scatter plot, missing values occur most frequently in datasets with relatively small numbers of samples and features. 365 instances and 17 features give the median dimension of these datasets. The missing rate, described by the percentage of missing values among all values, reaches a maximum of close to 70%. The median and mean missing rates are significantly lower with 2% and 7.25%. Between the different feature types, we could not identify notable differences in the missing rate with a small plus on nominal features (8%) over numerical features (6.6%). However, in combination with high dimensionality, even those low missing rates can result in many incomplete samples. To be more precise, the chance that a sample contains missing values increases exponentially with the dataset dimension under the MCAR assumption. Among the OpenML datasets which include missing values, we observed that on average 20% of all samples are incomplete. The median is even higher with 35.8%. Figure 2 illustrates the percentage of incomplete samples in datasets with missing rates of greater than 0% (left) or 5% (right).

The left histogram shows the relative frequencies of different percentages of incomplete samples among all datasets containing missing values. It can be seen that slightly more than a quarter of all datasets contain $0 - 5\%$ incomplete instances. However, many datasets include significantly more incomplete samples and roughly 15% of all datasets consist solely of incomplete samples. Figure 2b shows how the histogram changes when only datasets with a missing rate of at least 5% are evaluated. Here, almost half of the datasets entirely consist of incomplete instances. This shows that a deletion approach, which is still in practical use, is a poor choice to tackle missing values and should be avoided for datasets with at least 5% missing values. Instead, more sophisticated methods are required to tackle for missing values.



(a) Missing rate $> 0\%$ (b) Missing rate $\geq 5\%$

Figure 2: Percentage of incomplete samples

1.1 Research Question

Feature selection has been an active research area since the 1970's and was historically motivated to speed up classification models. [75] While these computational issues are only severe for large and high dimensional datasets nowadays, e.g. microarray data in the bioinformatics domain, feature selection still comes with numerous other benefits. Feature selection has been shown to be effective in removing irrelevant and redundant features and thus leading to a better generalization [29], interpretability [58] and performance [75] of learned models. Further, the dimensionality reduction helps to save measurement and storage costs [31], to visualize and explore data [28] as well as to deal with the curse of dimensionality. [20]

Much of the research has focused on the development of feature selection methods which can efficiently select the most relevant and least redundant features out of large datasets, also tackling common problems such as mixed data and the identification of hidden interactions between features. [19, 37, 50, 60, 75] Unfortunately, most of these methods can not be applied anymore on datasets containing missing values as e.g. distances are undefined between incomplete instances. Further, statistical measures and correlations between features can only be estimated from observed data. However, missing data could follow a different distribution so that the estimations based on observed data can be misleading.

Many researchers handle missing values in a dedicated preprocessing step by either deleting incomplete samples or filling the gaps with estimations (imputation). Deletion leads to a significant loss of information and is only applicable to small missing rates up to 5%. [23] On the other hand, imputation uses all available information and complements the data. However, imputed values can significantly differ from their real counterparts and can thus introduce bias or even artificial structures to the data. [20] Multiple imputation techniques have been developed to account for the uncertainty of estimations by repeatedly imputing the data. [10] However, classical data mining pipelines require single scores, e.g. for distance calculations, so that multiple estimates for a missing value are replaced by the mean erasing the uncertainty. Imputation further suffers from irrelevant and uncorrelated features, e.g. kNN-imputation [17], becomes inefficient with increasing dataset dimensions and lacks the identification of features for which the missingness of values provides knowledge about the class.

This work addresses data mining on incomplete datasets by integrating the uncertainty of missing value estimations into a concrete feature selection approach rather than relying on single estimations. The work is guided by the following research questions:

- What are the limitations of current imputation techniques? How do they cope with irrelevant features, high-dimensional datasets and different missing mechanisms?
- Does feature selection perform better when it is applied directly on incomplete data rather than on imputed data? What is the best strategy to integrate the handling of missing values into feature selection?
- Does feature selection generally improve the classification task on incomplete datasets?

1.2 Proposed Solution

Feature selection on incomplete data is addressed by only few works, e.g. [20] and [66], and is further mentioned as side-product of missing value aware classifiers, such as Random Forests [8] and XGBoost [12]. However, all these methods lack at least one of the following requirements: identification of high order features and redundancy, handling of mixed data or efficiency.

This work builds on a recently proposed framework for feature ranking based on Relevance and Redundancy (RaR). [7, 60] RaR efficiently handles large and mixed datasets while considering multi-feature interactions and redundancy. The authors provide experimental results showing that RaR outperforms several state-of-the-art feature selection techniques on synthetic and real-world datasets. During this work, we extend their approach by integrating robust handling of missing values in the form of a weight function. We further analyze the different parameters of RaR and provide a configuration which applies to various datasets. Finally, we introduce an active sampling approach improving both quality and efficiency.

RaR uses an adapted version of a contrast measure initially used to identify high contrast subspaces (HiCS) for density-based outlier mining [35] which quantifies the conditional dependence between a subspace and the class. RaR uses this contrast to determine the relevance of a subspace, and similarly the redundancy of a feature towards a subspace. The contrast is estimated by creating multiple conditional distributions for the class, or feature, and comparing it to the marginal distribution using a divergence measure, such as Kullback-Leibler divergence (KLD) [40] or Kolmogorov-Smirnov (KS) test statistic. [45] KLD is chosen for the relevance estimation as it converges to mutual information which is a commonly used measure for feature importance. [60] In a high-level overview, RaR estimates relevances in randomly drawn subspaces and deduces single feature relevances by solving an optimization function. Finally, RaR ranks features in an iterative process which also incorporates redundancy.

We integrate the handling of missing values into the estimation of conditional distributions by proposing a weight function. This function models the probability of a missing value to fulfill a condition on the subspace rather than making a binary decision on whether the condition is fulfilled or not. This accounts for the uncertainty of missing values and excuses for small errors in missing value estimations. During this work we propose two categories of weight functions: the first assigns probabilities uniformly to all missing values in a feature whereas the second relies on imputed values to assign probabilities to each missing value individually.

We provide extensive evaluations of the different instantiations on synthetic and real-world datasets and compare them to state-of-the-art feature selection and imputation techniques. Therefore, we first present methods for synthetic data generation and missing value simulation and describe how related methods need to be adjusted for incomplete and mixed datasets. The evaluation is finally based on ranking and classification performances and shows that RaR outperforms its competitor methods in most of the cases.

1.3 Outline

The remainder of this work is structured as follows: Chapter 2 provides a formal problem setting including the notation of this work, definitions of missing mechanisms, mathematical foundations of RaR as well as essential definitions and goals of feature selection. Shortcomings of traditional solutions to missing data, such as deletion and imputation, are presented in Chapter 3 leading to the necessity of integrating the handling of missing values into feature selection approaches. Further, the chapter discusses related works on feature selection according to five criteria with a particular focus on the integration of missing values. Chapter 4 starts with a review of RaR and highlights significant changes and extensions used to make the algorithm more robust, efficient and less dependent on parameter tuning. The chapter further introduces the concept of active sampling and the novel weighting strategies together with a discussion of their advantages and shortcomings. An extensive experimental evaluation in Chapter 5 supports the theoretical discussions from previous sections. The experiments section starts with a description of the experimental setup including related methods, synthetic data generation, missing value simulation, and evaluation metrics. The results on synthetic and real-world datasets are presented next and prove that RaR is a scalable and robust method which outperforms state-of-the-art feature selection and imputation techniques. To the best of our knowledge, the experimental part is the first and most comprehensive evaluation of feature selection methods in the presence of missing values. Final conclusions summarize and discuss the contributions of this work in Chapter 6.

2 Framework for Feature Ranking on Incomplete Datasets

This chapter introduces mathematical notations and concepts used throughout this work. It reviews essential mathematical foundations of the RaR framework and presents details on how we adjust the HiCS contrast measure to handle missing values. Finally, criteria for the comparison of feature selection methods are formally defined.

2.1 Notation

Let $DB = \{X, Y, M\}$ be a dataset in a supervised scenario consisting of a DIM-dimensional feature matrix X with N instances, a target vector Y with the same number of N instances and a binary indicator matrix M with the same dimensions as X. A pattern or sample $x_n \in X$ can be interpreted as an object, event or process which is observed at a specific time represented by the DIM feature values at row n formally described by $x_n = [x_{n1}, x_{n2}..., x_{nDIM}]^T$. Each pattern x_n is associated with a target or class label $y_n \in \{c_1, c_2..., c_C\}$ where C is the number of discrete classes. The features $f_i \in X^T$ define the columns of X and can be interpreted as the attributes of samples, e.g. the age or gender of patients. Each non-empty set of features $S \subseteq X^T$ is considered as a subspace throughout this work. The set of all features F is given by the union of nominal/categorical features Z with values from a discrete domain, e.g. gender, and numeric/continuous features Q with values from a continuous domain, e.g. age or weight. Mixed data is represented by a set of features $F = \{Z \cup Q \mid Z \neq \emptyset \wedge R \neq \emptyset\}$.

The binary indicator matrix M, or generally known as the missing data indicator matrix, is represented by ones and zeros where $m_{ij} = 1$ describes that the value x_{ij} is missing. Encoding each value in X with "?" where the corresponding value in M equals to zero leads to $X_{observed}$. Similarly, we define $X_{missing}$ as the matrix containing the missing values. When applying data mining tasks, the real values behind the missing values are not known and cannot be reconstructed anymore. Instead, robust ways are necessary to apply data mining on the observed feature matrix only. During this work missing values are only allowed in $X_{observed}$ meaning that all elements of the target vector Y are known. The missing rate λ of a dataset is given by the percentage of missing values $\lambda = \frac{1}{DIM*N} \sum_{m_{ij} \in M} m_{ij}$.

2.2 Missing Mechanisms

The best way to handle missing values depends on the underlying missing value assumption commonly known as missing mechanism. [23] Little and Rubin define three different mechanisms which are characterized by the conditional distribution of M given X: [44]

$$p(M|X_{observed}, X_{missing}, \xi) = p(M|X, \xi) \ , \tag{1}$$

where ξ denotes the unknown parameters which define the missing data mechanism. [23]

Missing completely at random (MCAR) describes a situation in which the missingness of a value neither depends on its real value nor any other value expressed by:

$$p(M|X_{observed}, X_{missing}, \xi) = p(M|\xi) \ . \tag{2}$$

MCAR means that there is no observed or hidden reason why a value is missing, e.g. a questionnaire accidentally got lost or a doctor simply forgot to run a test on a patient.

Missing at random (MAR) differs from the MCAR mechanism as the missingness of a value is related to observed values in other features in the dataset and can thus be predicted. MAR missingness can be formulated by:

$$p(M|X_{observed}, X_{missing}, \xi) = p(M|X_{observed}, \xi) \ . \tag{3}$$

For missing values following the MAR mechanism the missingness still does not depend on the hidden value but on an external influence which is observable. In an example male participants of a survey do not answer questions about pregnancy or a sensor cannot measure data during a power outage. In a particular form of MAR the data can be *informatively missing* which means that the missingness is different among classes and thus has a predictive meaning. From our previous example, the gender of a participant of a survey is predictable by the missingness of values for gender-related questions. A standard solution to this kind of missing data is to create a new category for missing values. [16]

Missing not at random (MNAR) characterizes the hardest missing mechanism as there is no general method of handling it properly. [23] In MNAR the missingness of a value depends on its hidden value which makes it impossible to predict. MNAR can be represented by:

$$p(M|X_{observed}, X_{missing}, \xi) = p(M|X_{missing}, \xi) \ . \tag{4}$$

MNAR thus means that only certain values are lost, e.g. a sensor can only measure values in a specific range. García et al. further mention that the reasons for MNAR missingness cannot be ignored and that the distribution of missing data has to be modeled. [23] In contrast, MCAR and MAR mechanisms are termed ignorable and suited for solutions such as imputation.

2.3 Foundations of RaR

This subsection explains the mathematical foundations of RaR introduced by [7] and [60] and adjusts them to the notation of this work. RaR makes use of the concept of conditional independence to decide whether features are relevant to the class or redundant to other features. Binary versions of relevance and redundancy are given in definitions 2.1 and 2.2.

Definition 2.1 (Absolute Subspace Relevance)**.** *A non-empty subspace $S \subseteq X^T$ is defined as absolutely relevant w.r.t. Y iff:*

$$P\big(p(Y|S) = p(Y)\big) \neq 1 \ .$$

This definition implies that a subspace is relevant if it contains at least one feature which provides predictive power about the target. Redundancy, in contrast to relevance, is defined between a feature and a subspace.

Definition 2.2 (Absolute Feature Redundancy)**.** *A feature $f \in X^T$ is called redundant w.r.t. a non-empty subspace $S \subseteq X^T \setminus \{f\}$ iff:*

$$P\big(p(f|S) = p(f)\big) \neq 1 \ .$$

The definition describes a feature to be absolutely redundant to a subspace when they are conditionally dependent. Unfortunately, both definitions only represent a binary view of relevance and redundancy and do not allow a fine-grained ranking of features. Relevance and redundancy can be redefined using a statistical divergence function, also referred to as contrast function in other works, to overcome this issue. [35]

Definition 2.3 (Divergence)**.** *For two probability distributions P and Q, a function $D(P \parallel Q)$ is called a divergence function iff:*

(i) $D(P \parallel Q) \geq 0$ *for every* P *and* Q*,*

(ii) $D(P \parallel Q) = 0 \iff P = Q$ *.*

A divergence function describes how distinct or similar two probability distributions are. RaR takes advantage of the divergence to quantify the conditional dependence between a subspace towards the class (relevance) and the dependency of a feature to a subspace (redundancy). Therefore, RaR instantiates the divergence function with the Kullback-Leibler-Divergence (KLD) for nominal targets and the Kolmogorov-Smirnov (KS) test statistic for numerical targets. KLD has the advantage of converging to mutual information which is a well-established metric for feature relevance. [60] Using a divergence function, subspace relevance and feature redundancy can be redefined according to definitions 2.4 and 2.5

Definition 2.4 (Subspace Relevance)**.** *The relevance of a non-empty subspace $S \subseteq X^T$ w.r.t. Y is quantified by the divergence between the marginal and conditional class distribution measured by an invariant divergence function D:*

$$rel_D(S) = D\big(p(Y|S) \parallel p(Y)\big) \ .$$

Following this definition of relevance, subspaces conditionally independent of the class obtain a relevance score of zero whereas subspaces obtain a high relevance when their conditioning changes the distribution of the class. Similarly, the redundancy of a feature w.r.t. a subspace can be defined by the divergence of the marginal and conditional distribution of the feature when the subspace is conditioned.

Definition 2.5 (Feature Redundancy)**.** *The redundancy of a feature $f \in X^T$ w.r.t. a non-empty subspace $S \subseteq X^T \setminus \{f\}$ is defined by:*

$$red_D(f, S) = D\big(p(f|S) \parallel p(f)\big) \ .$$

Defining relevance and redundancy using invariant divergence functions, such as KLD and the KS test statistic, comes with the advantage of information monotonicity. [7]

Definition 2.6 (Information Monotonicity [14])**.** *Let $P = (p_1, ..., p_n)$ and $Q = (q_1, ..., q_n)$ be two discrete probability distributions over the same sample space $\Omega = \{x_1, ..., x_n\}$. A divergence function D is called invariant if it holds that any partition $A = (A_1, ...A_m)$ of $\{1, ..., n\}$ with notation $P^A = (p_1^A, ..., p_m^A), p_i^A = \sum_{k \in A_i} p_k$ leads to:*

$$D(P^A \parallel Q^A) \leq D(P^A \parallel Q) \leq D(P \parallel Q) \ .$$

It can be shown that the KS test statistic, used by RaR as divergence measure between continuous probability distributions, is also invariant and fulfills the information monotonicity. [14] This property of a divergence function expresses the intuition that the relevance of a subspace S never decreases when a feature is added to S. The same is valid for the redundancy of a feature w.r.t. a subspace. Additionally, it is intuitive that removing a feature from a subspace S never increases its relevance or the redundancy of a feature to S. These observations are useful as they enable RaR to estimate single relevance scores of features based on all subspaces they are part of. The monotonicity of relevance and redundancy is defined below.

Definition 2.7 (Monotonicity of Relevance and Redundancy)**.** *Let $f_i \neq f_j \in X^T$ be two distinct features and $S \subseteq X^T \setminus \{f_i \cup f_j\}$ a non-empty subspace which contains none of the two features. The information monotonicity of relevance and redundancy guarantees that:*

(i) $rel_D(S) \leq rel_D\big(\{S \cup f_i\}\big)$,

(ii) $red_D(f_j, S) \leq red_D\big(f_j, \{S \cup f_i\}\big)$.

Whether the definitions presented in this subsection give an idea of how RaR quantifies relevance and redundancy, it is still unclear how the conditional distributions are estimated and where the algorithm breaks when being confronted with missing values. The next subsection focuses on these parts and presents a formal solution for incomplete datasets.

2.4 Contrast Estimation

RaR uses an adapted version of the HiCS [35] contrast measure which estimates the conditional dependence between a subspace and the class to quantify subspace relevance. Therefore, multiple conditional distributions of the class are obtained by slicing a subspace, e.g. $20 \leq age \leq 25$ and $gender \in \{"female"\}$. A robust estimate for relevance can be obtained by repeating the slicing procedure and measuring the average contrast between the conditional and marginal class distributions. Unfortunately, it is impossible to determine if a missing value falls into a specific value range and thus fulfills a slice condition. A widely used solution is to impute missing values and to operate on the completed dataset. However, relying on imputed values is risky and can result in large errors in the contrast calculation when the imputed values differ significantly from their real counterparts. Instead of operating on an imputed dataset, we propose to use a weighting approach for missing values which models the probability for an unknown value to fulfill a slice condition.

Definition 2.8 (Feature Slice)**.** *Let $f \in X^T$ be a feature containing missing values and SL a slice on f defining which samples should be accounted in estimating the conditional distribution of Y. We define a function $feature\_slice : (\mathbb{R} \cup "?") \rightarrow [0, 1]$ which assigns a value to each sample $f_i \in f$ as in the following:*

$$feature\_slice(f_i) = \begin{cases} 1 & \text{if } f_i \in SL, \\ 0 & \text{if } f_i \notin SL, \\ weight(f_i) & \text{if } f_i = "?", \end{cases}$$

where $weight : "?" \rightarrow [0, 1]$ is a function which models the probability of a missing value to be inside the slice.

For subspaces containing more than one dimension, we need to define a new combination function as we allow values different from zero and one inside a slice.

Definition 2.9 (Subspace Slice)**.** *Let $S \subseteq X^T$ be a subspace of dimension $k \geq 1$. We define a function $slice : \mathbb{R}^k \to [0,1]$ which assigns a value to each sample $x_n \in S^T$:*

$$slice(x_n) = \prod_{i=1}^{k} feature\_slice(x_{ni}) \ .$$

Finally, the conditional class distribution given a subspace S can be defined.

Definition 2.10 (Conditional Class Distribution)**.** *Let $S \subseteq X^T$ be a subspace of dimension $k \geq 1$. The empirical conditional probability mass function is defined for all $c_i \in \{c_1, ..., c_C\}$:*

$$pmf(c_i, S) = \frac{1}{\sum_{n=1}^{N} slice(x_n^S)} \cdot \sum_{n=1}^{N} \mathbb{1}(y_n, c_i) slice(x_n^S) \ ,$$

where $\mathbb{1}$ is the identity function.

This definition allows samples to have an impact on the subspace contrast even though they contain missing values. The uncertainty which comes with missing values is modelled by the weight function for which we provide different strategies in the main part of this work. With all the previous definitions in mind, the goal of feature selection on incomplete datasets can finally be formalized.

2.5 Formal Problem Setting

Feature selection aims to select a feature subset which maximizes relevance to the class while at the same time reducing redundancy between features. In a slightly different setting, feature ranking orders all features based on their importance, usually a value in the range $[0, 1]$. Such a ranking provides more information about the importance of features than a selection which only presents a binary view.

Definition 2.11 (Feature Ranking)**.** *Let $f_i \neq f_j \in X^T$ be two distinct features of a DB and $score : \mathbb{R}^N \to [0,1]$ a function which assigns importance scores to each feature. A function $ranking : DB \to X^T$ is defined to build the order of features based on their importance:*

$$ranking(DB) = \{(f_i, f_j) \mid score(f_i) \geq score(f_j)\} \ .$$

Our primary research goal is to compute rankings which stay consistent when the number of missing values in a DB increases, e.g. simulated by a deletion process on X. The consistency or robustness can be evaluated by comparing rankings, produced by a specific method, over multiple runs on the same DB.

Definition 2.12 (Consistent Feature Ranking)**.** *Let λ be a real value in the interval $(0, 1)$, $deletion : (DB, \lambda) \to DB$ a function which removes a percentage λ of all values from X following MCAR and $distance_{ranking} : X^T \times X^T \to [0,1]$ a function which computes the distance between two rankings. Consistent feature ranking aims to minimize the following:*

$$\forall \lambda \in (0,1) \ minimize: \ distance_{ranking}(ranking(DB), \ ranking(deletion(DB, \lambda))) \ .$$

Further, ranking qualities can be assessed by comparing to a "gold ranking" under the assumption that the true relevances are known. In cases where no gold ranking is available or where missing mechanisms other than MCAR are observed, ranking qualities can be assessed based on classification accuracies. The goals here are obviously to rank features with the highest relevances first and to achieve the highest classification accuracies.

Besides the robustness towards missing data, feature selection is challenged by:

- **Relevance and Redundancy:** Selected features should correlate with the class but not with each other.
- **Feature Interactions:** Feature interactions, referred to as high-order features and characterized by $rel(f_i) + rel(f_j) << rel(f_i, f_j) \mid f_i \neq f_j$, should be considered.
- **Mixed Data:** The feature space X^T consists of numerical and nominal features.
- **Efficiency:** The runtime should not fall prey to exponential search space growth.

The next chapter takes a detailed look at these challenges and presents related work in this field including their strengths and weaknesses.

3 Related Work

This section reviews common state-of-the-art methods for feature selection and puts particular emphasis on the problem of missing data. Feature ranking, as a special case of feature selection, assigns a score to each feature indicating its relevance to the class and thus provides more insights about the data. By selecting features until a cut-off point, feature ranking can reduce the dimensionality of datasets without erasing predictive information towards the class.

Another method for dimensionality reduction is called feature extraction which has Principal Component Analysis (PCA) [73] as one of its best-known representatives. The principal components are linear combinations of the original features and explain most of the variance among the data. Even though feature extraction methods such as PCA can lead to better classification accuracies and more information features, they are excluded from this work as their results are harder to interpret and cannot be used to reduce measurement costs. [29, 58, 63, 74]

Feature selection consists of two parts: a search function to explore different subspaces and an evaluation function to compare these sets. [29, 66] Much research has focused on finding efficient algorithms for these two parts, but unfortunately most of them cannot be applied when facing datasets with missing values. [20] With the ever-growing number of high dimensional datasets with large missing rates robust feature selection techniques, which can handle missing values, become necessary. After providing a more global view on feature selection in the presence of missing values, we present and discuss concrete algorithms based on five properties.

3.1 Missing Data

According to Doquire the most severe problem with missing values is that most approaches cannot be applied anymore. [20] Reasons for the non-applicability depend on the specific algorithms; a common reason, however, is the lack of distance functions for incomplete instances. To apply feature selection on incomplete datasets two general ways exist:

1. Deletion or Imputation transform the datasets into complete ones.
2. Approaches are adjusted to handle missing values internally.

The first approach decouples the missing data and feature selection problem and enables the usage of all standard approaches for feature selection as well as other data mining tasks, e.g. classification. Deletion makes poor use of the data and imputation can add bias or artificial patterns to the data when the imputed values differ significantly from their real counterparts. Thus the second approach could lead to better results when robust procedures to handle missing values are developed and integrated into existing methods.

Complete and available data analysis

Complete data analysis removes all incomplete samples from a dataset whereas available data analysis only eliminates samples with missing values in the variables of interest. Both approaches are prevalent in practice but strongly discouraged as they lead to a significant loss of information and can only be applied to datasets with low missing rates up to 5% and many samples. [23] Further, the approaches cannot cope with all missing mechanisms and fail when the distribution of missing and available data are different.

Imputation of missing values

Instead of deleting incomplete samples, imputation strategies try to find good estimates which replace missing values. Imputation itself forms an active research area producing different approaches which have been shown to improve data mining tasks, such as classification. [3, 22, 68] Garcia et al. [23] group imputation approaches into statistical, machine learning and model-based methods. The most natural statistical technique is mean imputation which replaces each missing value of a feature with the mean of observed values in the feature. However, this method ignores the variability of data and does not consider relations between attributes. [23]

A very common imputation method relying on machine learning is k-nearest-neighbor (kNN) imputation. [17] The idea of this technique is to use the average values among the k closest samples for missing value replacement. To find the nearest neighbors a distance function needs to be defined. A common choice is the heterogeneous euclidean overlap metric. [72] Batista et al. show that KNN imputation leads superior classification accuracies compared to a direct classification using C4.5 and CN2 classifiers. [3] Further, they demonstrate the effectiveness of kNN imputation on datasets with high missing rates. More sophisticated methods include regression imputation, expectation-maximization algorithms, multiple imputation (e.g. MICE) as well as approaches relying on neural networks. [10, 26, 44, 47, 54, 57, 59]

Imputation makes full use of the available data and complements it by replacing missing values with promising estimates. However, imputation is computationally expensive for high dimensional datasets and can introduce bias and artificial patterns to the data which distort the feature selection results. [20] Doquire and Verleysen also state that irrelevant features harm imputation procedures so that imputation should be applied after feature selection. [20] This claim especially holds for distance-based methods such as kNN imputation. Another drawback of imputation is that it only works well for non-informatively missing values. [23] Finally, traditional data mining pipelines integrate imputation by replacing missing values with fixed single scores. Giving the same importance to estimations as to observed values is risky as the estimates could be wrong. Further, the uncertainty of missing value estimation gets lost even though it is modeled by multiple imputation techniques when missing values are replaced with their average estimate.

Embedded handling of missing values

To overcome the problems of complete-case analysis and imputation techniques some approaches have been developed to handle missing values more naturally. Most of these works are from the research field of classification. Among the classifiers which can handle missing values internally, the most popular ones are tree-based such as ID3, C4.5 and CN2. [13, 51–53] ID3 handles missing values by generating additional edges for them whereas CN2 fills missing values with the mode before entropy calculation. The popular C4.5 learning algorithm proposed by Quinlan is an extension of ID3 and uses a probabilistic approach. In the training stage, attributes obtain weights based on the fact if it is observed or missing. During testing, the most probable class is selected by exploring all branches. Another successful tree-based method which gained attention in recent years is XGBoost, proposed by Chen and Guestrin. [12] In their approach, missing values are excluded from the split calculation and the optimal direction (left or right) is learned from the data beforehand.

The development of robust classifiers in the presence of missing values eliminates the strong need for deletion or imputation and led to the creation of feature selection methods relying on classifier estimates. We present some of these works in the next section. For methods relying on different evaluation methods than classifier estimates we explicitly mention how they can handle missing values or how we adjust them to work on incomplete data.

3.2 Evaluation Methods

Feature selection methods are broadly categorized into filter-based, wrapper-based and embedded methods depending on the method used for evaluation of a feature subset. [20,29,58,60,63] During this work a new category - *subspacing* - is opened. The following subsections present and discuss the different categories and give concrete examples.

Wrapper

Wrapper-based methods evaluate feature subsets by measuring classification performances. This usually leads to the highest accuracies but comes with many drawbacks. [20, 32] Due to the multiple classifier evaluations, wrapper methods tend to be highly inefficient, especially when the dataset size is large. Furthermore, the selected features are optimized on a specific classifier and hence, lack generality. [11] When facing incomplete datasets, the only adaptation of traditional wrapper approaches is to choose a missing value aware classifier.

Despite the choice of a classifier, wrapper methods differ by their search functions. There are mainly three different approaches: sequential forward selection (SFS), recursive backward elimination (RBE) and evolutionary methods. [66] In a sequential forward selection, the feature which adds the most value to the current set of features is chosen next. SFS eliminates redundancy among features but lacks feature interactions between pairs of currently unselected features. [60] In contrast, recursive backward elimination starts with the full feature set and recursively removes the one which leads to the smallest performance loss. This way, redundancy and feature interactions are handled. However, both SFS and RBE lack efficiency and can only be applied on datasets of low dimensionality. [60] Further, SFS is better suited for cases where small feature sets should be obtained rather than a complete ranking. In recent years evolutionary approaches have been introduced as more efficient search functions. [66] Two examples of this type are genetic algorithms and particle swarm optimization (PSO) [21]. Both of them can handle redundancy and feature interactions but also lack efficiency when classifier evaluations are used to assess the fitness of individuals.

Tran et al. propose a wrapper-based method for incomplete datasets in [66]. The authors use a C4.5 classifier wrapped into PSO and claim that this approach leads to better classification accuracies and less complex models. In [65] and [67] the authors extend their study by testing on a larger set of datasets and by the concept of bagging. Tran et al. compare the classification accuracies of three different approaches: classification of incomplete data, imputation + classification and feature selection + classification of incomplete data. [66] However, this does not answer whether their methodology is superior to the traditional approach of imputing data in the first step and then applying feature selection and classification on the completed dataset. Furthermore, they do not include a runtime evaluation and only rely on simple imputation techniques such as mean and kNN imputation. Using PSO and C4.5

classifiers can be inefficient for large datasets depending on the PSO parameters. Additionally, decision trees tend to be unstable and their performances can differ significantly due to slight changes in the data. [42] This could be even worse in the presence of missing values.

To complement the wrapper-based methods, a sequential forward selection using a kNN classifier with a partial euclidean distance [20] is investigated. According to Li et al., kNN classifiers are more stable than trees [43] and hence, better suited for feature selection. However, kNN classifiers are inefficient on large datasets due to many distance calculations and further suffer from the curse of dimensionality when evaluating high dimensional subspaces. [33]

Filter

Filter-based methods evaluate feature subsets using a criterion independent of a classifier. They are thus more efficient than wrapper methods and provide a better generality of features. [36] Filter also rely on search functions to select feature subsets for evaluation. Most straightforwardly, a filter evaluates each attribute individually which is fast but ignores interactions and redundancy between features. [50] Many criteria exist for such a bivariate evaluation with Pearson's correlation coefficient and mutual information (MI) being among the most popular ones. [29] MI has the benefit over Pearson's correlation that it can detect non-linear dependencies between attributes and can also be used as multivariate criterion. [4]

To directly compute the mutual information shared between two groups of random variables, the probability density functions (pdf's) must be known. As these pdf's are not known in practice, much work has focused on accurate estimations including histogram and kernel-based methods as well as estimations based on k nearest neighbors. [39, 48, 56, 61] The first two approaches suffer from the curse of dimensionality and lead to inaccurate results in higher dimensions. [20] Kraskov has introduced a MI estimator using k nearest neighbors for continuous [39] and Ross for categorical targets [56]. Both estimators use the Kozachenko-Leonenko estimator for entropy proposed in [38]. These estimators rely solely on distances between samples and do not require any estimation of distributions. Doquire and Verleysen use the Kraskov estimator with a partial distance function to apply feature selection on incomplete datasets for regression problems. [20] The proposed partial distance function computes the distance between two samples by normalizing the euclidean distance between complete attribute pairs with the number of complete attribute pairs. The MI estimator is finally wrapped into a sequential forward selection to obtain a feature set. However, this SFS lacks important feature interactions among unselected features and is prone to errors in the first steps, especially as the distance is undefined for samples without observed values. Finally, relying on k nearest neighbors makes the MI estimation inefficient and inaccurate for high dimensional data.

Due to their efficiency many filter based methods have been developed. Correlation-based feature selection (CFS), an extension of Pearson's correlation coefficient, and the minimal-redundancy-maximal-relevance (mRmR) criterion incorporate redundancy but ignore feature interactions. [32, 50] ReliefF is a filter method which can identify these interactions but lacks redundancy and becomes unreliable for datasets with many features. [37, 55, 70] When facing missing values, all of these methods cannot be applied anymore without dedicated adjustments. However, ReliefF can handle incomplete data when using a missing value aware distance function. In the case of CFS and mRmR, we replace missing values by mean imputation.

Embedded

Embedded feature selection utilizes the information during the training of a classifier to score features. The most prominent classifiers integrating feature selection are tree-based. In a concrete example, ID3 uses the information gain to determine the best split. [51] The information gain thus serves as a measure of feature importance. Unfortunately, features selected later in the tree construction can explain less impurity of the data and are hence, scored as less relevant. This drawback is reduced when using a random forest [8] which averages the feature importance over all trees in the forest. However, the feature scores obtained by embedded methods are highly optimized towards the specific learning algorithm and tend to do not generalize well. Furthermore, embedded approaches inherit the drawbacks of the model such as instability and lack of feature interactions in the case of tree-based methods. Random Forests further tend to assign higher importance measures to variables with more categories and thus lead to unreliable rankings in the case of mixed data. [62]

Subspacing

When dealing with high dimensional data, many algorithms suffer from the curse of dimensionality. This also holds for a variety of feature ranking algorithms, e.g. distances between samples converge to very similar values and cannot be used for accurate estimations anymore. [35] Recently developed subspacing approaches tackle this problem by evaluating randomly drawn subspaces utilizing a classifier or multivariate evaluation criterion. Finally, subspacing methods provide a function to deduce single feature scores from the subspace evaluations.

Li et al. propose Random KNN (RKNN) feature selection as a fast and stable alternative to random forests. [43] RKNN randomly selects subspaces, partitions them into a training and test set and finally assesses classification accuracies using a kNN classifier. The feature importances are obtained by averaging the classification accuracies of subspaces where the features are part of. The authors mention that their approach is well-suited for the case of missing values even though they do not elaborate specifically on this topic. RKNN can be adjusted to handle incomplete data by applying kNN imputation inside the subspaces or by using a missing value aware distance metric as in [20]. As RKNN relies on multiple evaluations of a kNN classifier, it comes with the same drawbacks as wrapper methods including inefficiency and lack of generalization. Furthermore, the averaging of scores over subspaces underestimates high order interactions and redundancies between features.

Shekar and Bocklisch present a smarter method of deducing single feature scores from multiple subspace evaluations. [7,60] Their framework for feature ranking based on Relevance and Redundancy (RaR) consists of three stages: sampling, deduction, and ranking. In a first stage, relevances of multiple subspaces are estimated based on conditional dependencies between the subspaces and the class. An adapted version of the contrast measure presented in HiCS [35] quantifies the conditional dependence. RaR instantiates the contrast with the Kullback-Leibler Divergence (KLD) [40] for categorical and with the Kolmogorov-Smirnov test statistic [45] for continuous targets. KLD serves well as a measure of relevance as it converges to mutual information. [60] Instead of averaging the scores from the sampling stage, the scores in RaR represent lower bounds for the individual feature relevances. Concrete individual scores are the results of the deduction stage which minimizes an objective function while fulfilling the

constraints from the sampling stage. The last step ranks features based on their relevance scores in an iterative manner while also considering the redundancy between features. The redundancy of a feature with regards to a subspace is estimated by measuring their conditional dependence. The authors of RaR demonstrate that their approach outperforms several state-of-the-art feature selection algorithms on synthetic and real-world datasets. Unfortunately, RaR is not applicable when values are missing as the conditional distributions cannot be estimated anymore. This work introduces essential changes to overcome this limitation.

3.3 Challenges in Feature Selection

Traditional challenges for feature selection include efficiency, mixed feature types and the identification of interactions and redundancies among features. This work opens a new evaluation criterion which is the handling of missing data. Table 1 presents an overview of related methods and their abilities to cope with the challenges of feature selection.

Approach	Missing data	Mixed data	Redundancy	Interactions	Efficiency
PSO + C4.5	✓	✓	✓	✗	✗
SFS + KNN	✓	✓	✓	✗	✗
MI	✓	✓	✗	✗	✓
mRmR	✗	✓	✓	✗	✓
CFS	✗	✓	✓	✗	✗
FCBF	✗	✓	✓	✗	✓
ReliefF	✓	✓	✗	✓	✓
Random Forest	✓	✓	✗	✗	✓
XGBoost	✓	✓	✗	✗	✓
RKNN	✓	✓	✗	✓	✗
RaR	✗	✓	✓	✓	✓

Table 1: Comparison of related work based on five challenges of feature selection.

Handling of missing data

Missing data is traditionally handled by deleting incomplete samples or imputing data before further processing. As already discussed these approaches do not make the best use of available data, introduce bias and artificial patterns compromising feature selection, and cannot deal with all missing mechanisms. Imputation techniques also suffer from irrelevant features which should be removed before imputing the data. [20] This, however, leads to the necessity of robust feature selection approaches which operate directly on incomplete datasets.

CFS, FCBF, mRmR and RaR do not handle missing data and other methods such, as MI, ReliefF, SFS + KNN and RKNN, are only able to handle it by using dedicated distance functions. In the case of high missing rates, the partial distance measure defined by Doquire is inaccurate as only few attribute pairs are complete and similarities between instances become more and more unreliable. Further, distances between samples become incomparable as they

rely on different numbers of attribute pairs depending on the location of missing values. This causes large errors for the methods relying on distances between samples. Besides the need for robust similarity measures, many information theory based evaluation criteria require robust procedures for estimating conditional and marginal probability density functions. The case of tree-based classifiers throws up different questions for split calculation, node assignment, and classification of test samples which are already covered in Section 3.1.

Due to its benefits on complete datasets, this work focuses on adapting RaR to incorporate missing values. RaR uses a slicing-based procedure for MI estimation which measures the contrast between conditional and marginal distributions using the Kullback-Leibler divergence. To estimate conditional distributions accurately, we propose a weight function estimating the probability of a missing value to fulfill a slicing condition. As no binary decision is required, this procedure excuses small errors in the missing value estimation and is more robust than traditional methods relying on fixed estimates. Further, the weight function can incorporate the modeled uncertainty obtained by multiple imputation techniques. Finally, the optimization and regularization part of RaR accounts for underestimations of subspace relevances and excuses for small errors in the contrast estimation.

Handling of mixed data

Datasets usually contain features of two types: continuous and discrete. Unfortunately, some methods only work on one of them. In an early integration scheme, techniques such as CFS and ID3 convert features into the type they support. [32,51] A conversion between feature types can cause a loss of information or order and is thus discouraged. [64] Among our competitors, CFS, FCBF and mRmR follow this approach by discretizing continuous features. In an intermediate scheme, features of different types are evaluated separately using dedicated measures and combined in a later step which is the case for ReliefF and several mixed data feature selection algorithms. [19, 37] Several works have also focused on distance measures which incorporate both continuous and discrete features. The algorithms presented in this work, assign a distance of one to a pair of discrete values when they do not match and zero otherwise. However, such a distance metric generally assigns lower distances to numeric features which adds a bias to the importance of different feature types. Further, tree-based methods apply different measures as split criteria for the two types of features. RaR also follows an intermediate integration scheme but uses very similar approaches for the slice creation on continuous and discrete feature.

Feature correlations and redundancy

Only a few approaches can identify interactions between features. These interactions, or high order features, mean that features are only relevant in combination with other features. According to Domingos and Pazani [18], many datasets contain these interactions making their detection an essential part of feature selection algorithms. Sequential search procedures and classifier constructions are not able to identify these interactions. The same holds for filter-based methods which evaluate features independently. Only ReliefF and the subspacing approaches can identify these interactions. However, RKNN has the problem that it underestimates them by averaging the results during the construction of the final ranking. Random

Forest and XGBoost incorporate interactions in the classification task but their feature importance measure ignores them by e.g. simply counting how often a feature is used for splitting.

Wrapper-based and embedded methods handle redundancy as the classification accuracy does not benefit from the addition of redundant features and can even decrease. Furthermore, some filter-based techniques can incorporate redundancy by following search functions such as SFS or by adding an extra term to the calculation of feature importance. Random Forests, RKNN and XGBoost are sensitive to redundancy in their subspaces but ignore it in the final ranking by averaging the results over subspaces. In contrast, RaR performs a dedicated stage for redundancy estimation and incorporates it explicitly in the formula of feature importance.

Efficiency

The efficiency of a feature selection algorithm is mainly dependent on its evaluation method. Generally, evaluations using a classifier are much slower than the computation of information theoretic or statistical measures. When wrapper methods further use extensive search functions such as SFS or RBE, they become even more inefficient due to high numbers of required evaluations. When inspecting Table 1 more closely it might not be obvious why CFS and the MI estimator are marked as inefficient even though they are filters. The reason is that the MI estimation is based on a k nearest neighbor computation which has quadratic complexity in the number of samples when a naïve implementation is used. Further, CFS scales poorly with the dimensionality of data and is thus considered inefficient. Random Forest and XGBoost stand out as efficient method which rely on the evaluation of multiple classifiers. The computation is still efficient as Random Forests follow a bootstrapping approach to select only some samples for the generation of subtrees. Further, the maximum number of considered features in subtrees can be limited and the split computation can be randomized. Observations for XGBoost are similar and it further increases performance by excluding missing values from classifier construction by learning a default path beforehand. Finally, RaR gets its efficiency due to fast evaluations of subspaces and parallelization of the sampling phase.

In summary, RaR is a method which efficiently explores the dataset while considering mixed data, redundancy and interactions between features. Furthermore and most importantly for this work, it has characteristics which make it well-suited for analyzing incomplete datasets.

4 Missing Value Aware Feature Ranking

This chapter reviews the RaR feature ranking framework based on [7] and [60]. It explains the motivation behind the different stages, gives insights about implementation details and illustrates specific parts of the algorithm. We further extend the previous works by a robust handling of missing values and an active sampling of subspaces. We also analyze essential parameters of RaR and provide a general setting removing the strong need of parameter tuning which is an issue of the original implementation. All extensions and modifications to the RaR framework get dedicated subsections or are explicitly stated as our contribution.

4.1 Overview

The goal of RaR is to assess relevance and redundancy scores for all features in a dataset. Evaluating features on their own is clearly an efficient method but lacks relevant interactions formed by multiple attributes. In contrast, an exhaustive search covering all possible subspaces is unfeasible due to the exponential amount of subspaces $S \subseteq 2^{DIM}$. Instead, RaR uses a heuristic approach of selecting only a certain number of subspaces which allows identifying high order features within a reasonable time. Single scores can be derived from the sampled subspace scores and combined into a final ranking which also considers redundancy between selected features. Figure 3 illustrates an overview of these different stages of RaR.

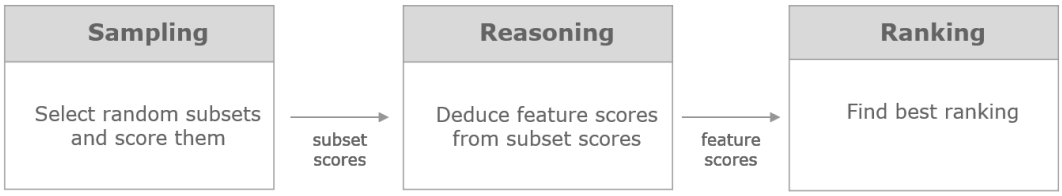
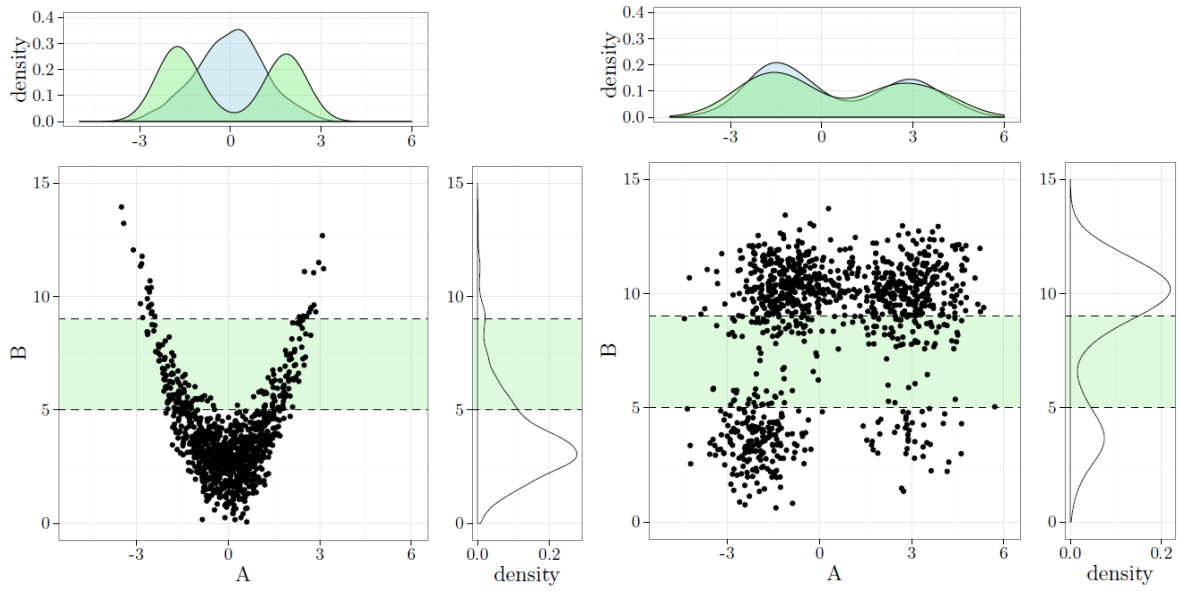


Figure 3: Overview showing the different steps implemented by RaR.

In a first step, RaR computes relevance scores for randomly selected subspaces by comparing conditional distributions of the class to its marginal as explained in Definition 2.4. RaR uses an adapted version of a subspace slicing based contrast, first described by Keller et al. in [35], to estimate conditional distributions for numerical and nominal features. Figure 4 gives an example of how to compute a conditional distribution and illustrates the idea behind contrast. The example contains two features where the contrast from B to A is estimated. One possible conditional distribution is obtained by only considering samples where $x_{nB} \in [5, 8]$.

The figure visualizes that the conditional and marginal distributions look different for correlated features whether they are almost identical in the case of uncorrelated features. Averaging the divergence between the distributions in Figure 4 for m randomly drawn slices can e.g. serve as a measure of redundancy between A and B. Similarly, relevances can be assessed in the sampling phase by applying the conditioning to the target vector. To get comparable estimates for the subspace contrast, the number of samples inside a slice, n_{select}, should be kept constant. [35] A slice on a numerical feature can be computed by sorting its values and randomly selecting a value as the lower bound. The upper bound is given by the value which is n_{select} elements larger. On sparse datasets, containing many zeros, a slice $[0, 0]$ can contain more than



(a) Correlated joint pdf

(b) Uncorrelated joint pdf

Figure 4: Example, taken from [7], showing two datasets with correlated and uncorrelated features. The green area includes all samples fulfilling the slice condition on B. The green density function at the top shows the distribution of values for these samples in A . The blue density illustrates the marginal distribution of A.

n_{select} elements. In this case, we randomly select samples which fall into the slice. Similarly, a slice on a nominal feature can be computed by randomly selecting categories until the number of samples inside the slice equals or exceeds n_{select}. When the number of samples exceeds n_{select}, the exact amount of samples is selected randomly from the lastly added category. As the slices for numerical and nominal features are created independently and combined in a unified way, this procedure is well-suited to subspaces with mixed feature types. The slice creation procedure also tackles missing values. Strategies to model the uncertainty of a missing value falling into a slice are explained later in Section 4.6.

The second phase of RaR deduces individual relevance scores from the sampling results while still incorporating the relevance hidden in interactions. This reasoning is crucial as the final ranking depends on individual feature ratings. Finally, the ranking phase iteratively ranks features based on their relevance and redundancy. The most relevant feature serves as a starting point with zero redundancy. For all remaining attributes, redundancies towards the first feature are computed and combined with the relevances. Finally, the most relevant and least redundant feature is selected next and the procedure is repeated until no feature is left.

4.2 Sampling Phase

RaR uses a sampling phase to assess relevance scores for a predefined number of subspaces, as it is not feasible to evaluate the whole search space which consists of 2^{DIM} possible subspaces. Further, RaR only rates subspaces until dimension k. The reason is that the contrast vanishes

for higher dimensions as the slice ranges on each feature in a subspace need to be increased. [35,60] The intuition behind this is that you end up with no samples which fulfill all conditions when the subspace dimension is large and the slice range is small. In an example dataset with $N = 1000$ samples and $DIM = 100$ features, we aim for a conditional distribution which contains $\alpha = 10\%$ of the samples. Without increasing α we would only select approximately 1% of all samples in a subspace of dimension $k = 2$. To keep the number constant α needs to be adjusted according to the dimension of a subspace as described in the following: $\alpha_k = \alpha^{1/k}$ which is already 31% for $k = 2$ and 95% for $k = 50$ dimensions. By almost selecting all samples over each dimension the conditional and marginal will be very similar and thus α and k need to be kept small. In [7] and [60] k is at maximum five which prevents from finding high order interactions formed by more features. However, they conclude that the limitation of $k \leq 5$ allows RaR to identify most of the high order features while keeping the runtime feasible.

The relevance scoring of each subspace S follows Definition 2.4 and is illustrated in Figure 4. Slices for each feature in S are created and combined following Definition 2.9. Finally, Kullback-Leibler divergence (KLD) measures the divergence of conditional, $p(Y \mid S)$, and marginal, $p(Y)$, distributions. This procedure is repeated m times and the contrasts are averaged to obtain the relevance of S. KLD is defined in the following

$$KLD(P, Q) = \sum_i P(i) * log\left(\frac{P(i)}{Q(i)}\right), \tag{5}$$

where P and Q are two discrete distributions. Shekar et al. show that KLD converges to mutual information and thus serves well as a measure of feature relevance. [60]

Boosting features with contrast in extreme values

During our work, we notice that RaR underestimates the relevance in some features in real-world datasets. We observe this underestimation in features where the contrast is hidden in samples with extreme value. An exemplary boxplot for such a feature is presented in Figure 5.

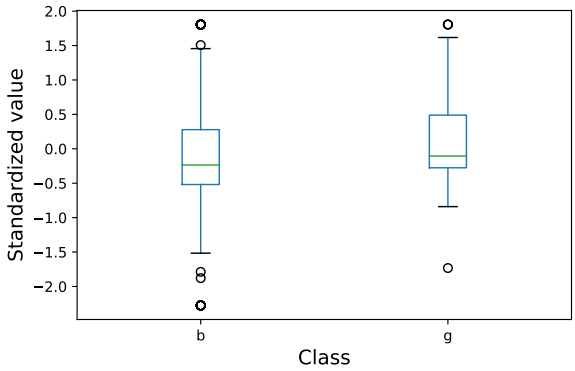


Figure 5: Boxplot showing the distribution of feature *a06* in the Ionosphere dataset.

The Ionosphere dataset contains 34 features and two classes *b* and *g*. The boxplot above shows the distribution for feature *a06* grouped by the class label. The distributions for the two classes look similar in general, but there are some outliers for the feature which strongly

indicate for class b. The feature thus is relevant and should be identified by RaR. Unfortunately, samples with extreme values are less likely inside the slice than samples with a value closer to the mean. Consider an example where 80% of the feature values should be inside the slice. For such a large slice, the 60% centered around the mean are always in the slice while samples with the highest or largest values are very unlikely and maybe never in the slice. An idea is to have a circular slice, e.g. with the following condition: $a06 > 1 \vee a06 < -1$. Following this idea, each sample gets the same probability of being inside the slice. Still, this idea does not make sense as the slice is not cohesive and contains very different values.

Instead, we implement an approach which additionally calculates the contrast for samples with extreme values for each feature independently. In an example, we estimate the relevance by slicing the ten lowest values. Further, the slice slides in the direction to the mean value while keeping the slice size constant. By following this procedure also for the highest values, we obtain slices for extremal regions. We hold the number of these slices small to only consider the extremal regions and compute the final feature boost by averaging the divergence between the conditional and marginal class distributions. Similarly, we compute a boost for nominal features. Here we observe the problem that the relevance is underestimated in features with contrast in rare categories. When a slice contains a category with few observations, it is extended by further categories until enough samples are available. By adding more categories the contrast vanishes. Thus we implemented a boosting which selects samples out of each category ignoring the others. Finally, we combine the relevances in extremal regions with the overall relevance and redundancy of a feature to compute the ranking.

Adjustments for missing values

We fix the issue of missing values in RaR by a modification of the slice creation. Therefore, we need to adjust parameters based on the missing rate λ and to implement a *weight* function as in Definition 2.8. In our implementation, we compute the slice range exclusively on observed values. This makes sense as we only know these values and the usage of imputed values could lead to biased slices. When the slice range is defined, the weight function is used to estimate the likelihood of a missing value to fall into a given slice.

In a complete dataset with N samples, a slice for a feature in a subspace S with dimensionality k contains $n_{select} = \alpha_k \cdot N$ samples. For a large k and a high λ, the number of observed samples for the feature can be lower than n_{select} and the algorithm breaks. Further, with fewer samples for a feature, the slice range increases when n_{select} is kept constant leading to the same issue of a too large α where the contrast vanishes. To tackle both problems n_{select} scales down with the missing rate in our approach and can be redefined as:

$$n_{select} = \alpha_k \cdot (1 - \lambda) \cdot N \; . \tag{6}$$

The slice creation is then performed equally to the one on complete datasets with the only modification that missing values are ignored. Based on the created slices, the *weight* function complements the slice vector by assigning weights to missing values. Most straightforwardly, the function sets the weights to zero. This means that a missing value never falls in a slice and that incomplete samples never have an impact on the conditionals. Section 4.6 presents more sophisticated instantiations of the *weight* function including their strengths and weaknesses.

Further, we note that the contrast estimation for subspaces containing missing values is less robust. We thus increase the probability for features with a high λ to be part of a selected subspace. In consequence, these features are evaluated more often and analyzed in different local contexts. Also, we find it helpful to form subspaces of features which have a positive correlation of missingness meaning that when a value exists in one attribute, it also exists in the other. This leads to fewer ignored samples within the naïve *weight* strategy. The correlation can be estimated using Pearson's correlation coefficient applied to the indicator matrix M. The correlation falls into $[-1, +1]$ where $+1$ means a perfect positive association.

In the case of a high λ and a small N, the conditional distribution for each slice is estimated on very few samples which can be even lower than the number of classes. This leads to highly inaccurate estimations of contrasts and should be avoided. We tackle this problem by specifying a minimum number of samples included in each subspace slice. Finally, we estimate subspace relevance by averaging the contrasts from all slices containing at least the specified amount of samples. When too few slices exceed the threshold, the relevance becomes inaccurate and we ignore the subspace. We present a discussion on the thresholds in Section 4.7.

Implementation details

The sampling phase makes up a large portion of the total runtime of RaR and thus needs to be implemented with performance in mind. There are two notable parts concerning the performance of the sampling phase. First, the slice creation must be fast for all features and second, the contrast calculation using KLD must be efficient.

We achieve a significant performance improvement for the first part by slice caching. Thus we can reuse slices in different contexts and eliminate the need of recalculating them. Before the sampling phase, we initialize a cache with m slices for each of the DIM features and k dimensions. Unfortunately, the caching is only feasible for small and medium-sized datasets due to memory constraints. However, we only notice this as a limitation for one of our datasets (Isolet). Using a cache should also be avoided for small numbers of sampling iterations. There it is very likely that many slices in the cache are unused. Nevertheless, we observe that our cache implementation greatly improves the runtime also in those cases by further optimizations to the cache initialization. The slice creation is more performant when its values are already in order. Based on the sorted feature, all m slices for a feature can be calculated in $O(N)$. During cache initialization the sorting is moved one layer upwards and thus all m·k slices for a feature can be computed in $O(\mathrm{k}{\cdot}N)$ after an initial sorting. In our implementation, we dynamically decide if a cache is used based on specified RaR parameters and memory constraints.

We achieve a performance improvement of the second part by optimizing the computation of KLD. Equation 7 shows how the relevance of a subspace S can be assessed from the conditional and marginal class distribution:

$$relevance(S, T) = KLD\big(p(Y|S) \parallel p(Y)\big) \ . \tag{7}$$

The relative frequencies of classes over all samples give the marginal distribution. Similarly, we estimate the conditional distribution by the distribution among instances inside a slice. Besides the obvious improvement of caching the marginal distribution, the more significant improvement comes with a smarter estimation of conditional distributions. Therefore we also

cache a vector of indices which sorts the target Y and use them to sort the matrix which contains all m slices for a subspace S. The slice matrix has the dimensions $\mathrm{m} \times N$ where a column represents if a sample falls into a slice. After sorting, the matrix matches the sorted target vector Y and a vectorized sum along the axis with dimension N computes the occurrences of all classes in each slice. Thus we only need to calculate $O(C)$ sums to obtain conditional distributions for all slices after an initial sorting of Y. Finally, the next phase deduces individual feature relevances based on the results of the sampling phase.

4.3 Reasoning Phase

During the sampling phase, RaR evaluates subspaces up to a dimension k. The final ranking, however, requires individual scores for each feature. Evaluating each feature independent of other features produces these scores but does not incorporate possible interactions and thus underestimates the relevance of a feature. A naïve way for deriving single relevance scores from results of a subspace S of dimension $k \geq 2$ is to define the relevance of S as the sum of the relevances of all contained features, $rel_D(S) = \sum_{f_i \in S} rel_D(f_i)$. Shekar [60] gives an example of why this decomposition of subspace relevance still underestimates feature interactions. Therefore consider a DB which consists of four features $X^T = \{f_1, f_2, f_3, f_4\}$ where the first three features form an interaction. During the sampling phase two subspaces have been evaluated leading to $rel_D(\{f_1, f_2, f_3\}) = 0.9$ and $rel_D(\{f_1, f_2, f_4\}) = 0.1$. The first subspace contains all features which form the interaction whether the second subspace only a part of it. Based on the second subspace the relevance of the first two features cannot exceed a value of 0.1 even though they should be much more relevant according to the result from the first subspace. The naïve decomposition thus leads to underestimations and can even lead to an unsolvable set of equations. Consider that another subspaces has been evaluated leading to $rel_D(f_3) = 0.1$. The inflexible formulation causes that the relevances for the first three features can nether sum up to 0.9 when the other two constraints should hold. Instead, the relevance of a subspace gives a lower bound for the sum of single relevance scores. [60]

Definition 4.1 (Feature constraint)**.** *For each non-empty subspace $S \subseteq X^T$ a feature constraint is derived as follows:*

$$rel_D(S) \leq \sum_{f_i \in S} r(f_i) \ ,$$

where $r(f_i)$ is a function which returns the decomposed value of f_i.

The set of m feature constraints does not provide a unique solution for individual scores. The works in [7] and [60] propose to find a reasonable solution fulfilling all constraints while at the same time minimizing the sum of individual relevances. Further, they add a regularization term which splits the relevance equally to all features in S when there are not enough constraints for a better decomposition. Their objective function is defined as follows:

$$\min_{r(f)} \left[\sum_{f \in X^T} r(f) + \sum_{f \in X^T} \big(r(f) - \mu\big)^2 \right] s.t. \forall i \in [1, M] : rel_D(S_i) \leq \sum_{f \in S_i} r(f) \ , \tag{8}$$

where $\mu = (1/DIM) \sum_{f \in X^T} r(f)$ is the average individual relevance. The regularization is motivated by the lack of a sufficient number of feature constraints. Without regularization

our former example with $rel_D(\{f_1, f_2, f_3\} = 0.9$ can lead to $r(f_1) = r(f_2) = 0;\ r(f_3) = 0.9$ or $r(f_1) = r(f_2) = r(f_3) = 0.3$. Both solutions are possible but without any further knowledge the second solution is clearly fairer.

During our work, we notice that the relevance of subspaces can exceed the value of one. This has two possible reasons: first, KLD is not normalized to the range of $[0, 1]$ and second, the optimization can deduce scores greater than one even though the constraints are all smaller. We solve the first by normalizing the relevance of a subspace to the interval $[0, 1]$ by:

$$normalized\_rel_D(S) = 1 - e^{-rel_D(S)}\ . \tag{9}$$

Second, we normalize the results of the optimization by dividing by the maximum relevance if it exceeds one. Finally, the reasoning phase provides individual scores, still incorporating feature interactions, based on sampling results as required by the ranking phase.

4.4 Ranking Phase

The single feature scores so far only consider relevance. This means that two identical features are ranked the same even though one feature is sufficient. A demanding challenge in feature selection is to identify redundancies and to incorporate their magnitudes into the ranking. A traditional measure for redundancy is Pearson's correlation coefficient. Unfortunately, it cannot be applied to estimate the redundancy of a feature f to a subspace S. In contrast, RaR uses the divergence between conditional and marginal distributions of f to estimate the redundancy. RaR instantiates the divergence function with the Kolmogorov-Smirnov (KS) test statistic in case of numerical features and with KLD for nominal features. Its faster calculation on continuous features and applicability for distributions with few samples, from 30 instances on [24], motivates the usage of the KS test statistic for redundancy estimation on a numerical feature f. The KS test statistic measures the maximum vertical distance between two cumulative probability distributions. A large score thus means that two distributions are very different and that the slicing on S affects the distribution of f. To penalize redundancy in the final ranking, we need to inverse the KS test statistic. Using Definition 2.5 for redundancy, the harmonic mean of relevance and redundancy describes the individual ranking scores as formalized by:

$$score(f, S_i) = \frac{2 \cdot r(f) \cdot \big(1 - red_D(f, S_i)\big)}{r(f) + \big(1 - red_D(f, S_i)\big)}\ , \tag{10}$$

where S_i is the subspace for which the redundancy of f is estimated to. As it is not clear how to initialize S_i, the authors in [60] propose an iterative construction of the ranking. The ranking starts with the most relevant feature which obtains a redundancy of zero as there are no features selected before. Now let R_n be the ranking which contains n already chosen features. To find the next the score of all non-selected features need to be computed using Equation 10 where S_i equals R_n. However, this means that for large n the redundancy needs to be estimated with regards to a high dimensional subspace which cannot be computed accurately. Thus Shekar proposes to compute redundancies to $\binom{n-1}{k}$ subspaces of dimension k randomly selected without replacement. [60] The maximum redundancy among these computations serves as the redundancy of the feature with regards to R_n.

Further, we suggest to integrate the relevance estimated in extremal regions of a feature $b(f)$ into Formula 10 by a linear combination where β_{boost} defines the impact of the boost:

$$score(f, S_i) = (1 - \beta_{boost}) \cdot \frac{2 \cdot r(f) \cdot \big(1 - red_D(f, S_i)\big)}{r(f) + \big(1 - red_D(f, S_i)\big)} + \beta_{boost} \cdot b(f) \ . \tag{11}$$

Adjustments for missing values

When estimating redundancies, we face the problem that the conditional and marginal distributions are estimated on features which can contain missing values. However, KLD and the KS test statistic only operate on distributions without missing values. To overcome this problem, we simply ignore all samples with missing values in the feature which is the subject of redundancy estimation. However, we notice this only to be a limitation for very high missing rates where this procedure leads to few samples from which no reliable results can be obtained anymore. The intuition why this approach works is that we only ignore instances which have missing values in a single dimension f and thus remove a relatively small amount of samples.

We further like to extend the ranking score by a value which incorporates the missingness correlation between features. Without considering this correlation, the dimensionality reduction could lead to many samples without any observed value in a particular subspace. We actively prevent this scenario by adding a new component to the ranking, similar to the feature boost. The correlation boost is computed for a feature f by determining its minimal missingness correlation to all already selected features in R_n. Note that only negative correlations are helpful for the ranking as a negative correlation means that when a value is missing in one attribute, it is likely that a value is present in another. By also optimizing for high negative correlations we can lower the risk of ending up with many samples without any observed value. We finally redefine the ranking score as:

$$\begin{aligned} score(f, S_i) = (1 - \beta_{boost} - \beta_{correlation}) \cdot \frac{2 \cdot r(f) \cdot \big(1 - red_D(f, S_i)\big)}{r(f) + \big(1 - red_D(f, S_i)\big)} \\ + \beta_{boost} \cdot b(f) + \beta_{correlation} \cdot \max_{f_i \in S_i} (1 - (1 + c(f, f_i)/2)) \ , \end{aligned} \tag{12}$$

where $c(f, f_i)$ defines the missingness correlation between two features computed by Pearson's correlation coefficient on M. The last term first normalizes the Pearson correlation and then calculates the maximum of its inverse which results in the maximal negative correlation.

Implementation details

During our implementation, we face two performance problems of the ranking phase. First, the KS test statistic is computed in $O(N \cdot log(N))$ which becomes a bottleneck due to its repeated computations. Second, the iterative calculation of the ranking leads to score computations in $O(DIM^2)$ limitting the efficiency to datasets with low or medium dimensionality.

Based on the knowledge that the values used in the computation of each conditional distribution are a subset of the marginal distribution, we can compute the KS test statistic for each slice in $O(N)$ after an initial sorting of the feature. Table 2 shows the original feature consisting of eight samples and two possible slices. Similarly to the relevance computation, we first find the sorted indices and use them to sort the feature and its slices resulting in Table 3.

Sample	1	6	2	0	3	5	4	7
Value	1	5	2	1	3	4	3	6
Marginal	1	1	1	1	1	1	1	1
Slice1	1	0	0	1	0	0	1	0
Slice2	0	0	0	0	1	1	0	1

Table 2: Example containing eight samples and two slices in one feature.

Sample	0	1	2	3	4	5	6	7
Value	1	1	2	3	3	4	5	6
Marginal	1	1	1	1	1	1	1	1
Slice1	1	1	0	0	1	0	0	0
Slice2	0	0	0	1	0	1	0	1

Table 3: Result of sorting the table based on the feature values.

Based on the representation shown in Table 3, we compute the KS test statistic for each slice in $O(N)$ by iterating over the samples and comparing the cumulated probability mass of the marginal and conditional. We first determine the number of elements in each distribution used to obtain values on how to increase the cumulated probability at position i. The mass added equals 1/8 for the marginal distribution and 1/3 for each conditional. This can further be generalized for cases in which slices include values other than zero and one. Whenever a sample is inside a slice, we increase the cumulated probability mass by this value. On each last occurrence of a feature value, the probability masses are compared to determine the distance between the cumulative probability distributions. Finally, the maximum distances determine the KS test statistics. Table 4 exemplifies this procedure when reading it from left to right.

Marginal	1	1	1	1	1	1	1	1
Slice1	1	1	0	0	1	0	0	0
Slice2	0	0	0	1	0	1	0	1
Cumulated mass	0.125	0.25	0.375	0.5	0.625	0.75	0.875	1
Cumulated mass	0.333	0.667	0.667	0.667	1	1	1	1
Cumulated mass	0	0	0	0.333	0.333	0.667	0.667	1
Is last?	no	yes	yes	no	yes	yes	yes	yes
Distance1	-	**0.417**	0.292	-	0.375	0.25	0.125	0
Distance2	-	0.25	**0.375**	-	0.292	0.083	0.208	0

Table 4: Example visualizing the iterative construction of the cumulative probability function with a direct computation of the KS test statistic.

The example shows how the KS test statistic is computed in $O(N)$ after an initial sorting of the feature. Both slices lead to conditional distributions which look different from the marginal as the KS test statistic is high with 0.417 and 0.375. This means that the feature is redundant to the subspace used for the slicing. In contrast to KLD, we cannot vectorize this procedure easily as sums would need to be calculated for each of the many unique values in a continuous feature. We thus decided to run this procedure on each slice individually and speed up our implementation using Cython which basically is Python with C or C++ data types.

The second improvement comes with a limitation of redundancy computations to the first x steps and making reuse of prior redundancies. Let X be a matrix containing 100 features. When adding the feature at position 51, $\binom{50-1}{k}$ subspaces of dimension k or lower need to be

evaluated for each open feature according to [60]. The chances are high that previous steps already provide results for some of these subspaces. By keeping the maximum redundancy of unselected features over the first 50 steps, this information helps in determining the 51^{st} feature and can already be sufficient for a large n. This allows reducing the number of subspaces evaluated at a specific step and enables a scoring even though no additional redundancies are estimated. Further, we can ensure that the redundancy of an open feature incorporates estimations towards all previously selected features by assuring that the subspaces conditioned for redundancy estimation include the latest feature. By keeping the maximum redundancy, the redundancy stops increasing at some point. Thus we can end our iterative ranking calculation after step x and sort the rest based on the maximum redundancies and relevances from the reasoning phase. By limiting the number of redundancy calculations, the estimations get a bit more uncertain. However, for features added after step x a correct ranking is less important than for the first x features which represent the dataset after dimensionality reduction. To summarize, the runtime is improved by reducing the number of redundancy estimations in each step and by limiting them to the first $x = \sqrt{DIM}$ steps. Further, we note that this procedure can even improve the quality as it ensures that the redundancy of a feature considers all features selected before. The pseudocode in Algorithm 1 shows our iterative ranking.

Algorithm 1 Ranking Procedure

INPUT: *relevances*: dict, DIM: int, X: matrix
OUTPUT: R - ranking of features

```
1:  procedure RANKING(relevances, DIM, X)
2:      best = max_{f∈X^T} [relevances(f_i)]          ▷ Find most relevant feature
3:      lastF ← best; R ← [best]
4:      maxRedundancies ← {f : 0 ∀f ∈ X^T}
5:      for i = 2 → √DIM do
6:          newS ← {lastF ∪ S}, S ⊆ R          ▷ Select random subspace containing lastF
7:          for f ∈ openFeatures do
8:              maxReds[f] ← max(maxReds[f], red_D(f, newS))
9:              calculate score according to Eq. 12 with red_D(f) = maxReds[f]
10:         end for
11:         best ← feature with highest score
12:         lastF ← best; R.append(best)
13:     end for
14:     calculate scores for openFeatures according to Eq. 12 with red_D(f) = maxReds[f]
15:     sort openFeatures according to scores and append to R
16:     return R
17: end procedure
```

First, the algorithm determines the most relevant feature and initializes a ranking with it as well as a variable containing the latest feature. Further, the algorithm initializes a dictionary which stores the maximum observed redundancy for each feature. The outer loop fills the first $\sqrt{DIM}$ positions by applying Formula 12 to all open features and choosing

the one with the highest score in each step. This feature also updates the latest feature at the end of each iteration and is appended to the ranking. The latest feature forms a new subspace in the next iteration towards which redundancies are calculated. This assures that all previously selected features are incorporated in the redundancy estimation of open attributes until iteration $\sqrt{DIM}$. From there on, the dictionary directly provides the scores eliminating further evaluations which are costly. Further, the dictionary enables a direct computation of scores for all open features so that the ranking can easily be completed. It is also possible to create multiple subspaces to which the redundancies are evaluated to, but we note that one subspace is already sufficient and provides a good trade-off of performance and quality.

4.5 Active Sampling

One major requirement for feature selection is stability. This means that a ranking should not change significantly over repeated runs on the same dataset. Stable rankings are especially important in the domain of bioinformatics. [63] Otherwise, doctors lose their trust in an algorithm and start to use other solutions. The same holds for other fields, e.g. where feature selection is integrated into a pipeline for classification and regression. The final quality of models derived from a data mining pipeline should not depend on luck during feature selection and should be stable as well. As RaR heavily relies on randomized parts, i.e. subspace sampling and slice creation, it lacks stability in many cases. This is especially true when the number of explored subspaces or created slices are low. Increasing both values leads to more stable results, but a lot of computation is wasted on low-contrast subspaces so that the algorithm becomes inefficient. Instead, we improve the stability of RaR by introducing an active sampling approach which evaluates subspaces in a more focused way. Our method follows two paradigms to achieve robustness: it actively resamples subspaces with uncertain contrasts, and it explores additional subspaces required for a better deduction in the reasoning phase.

The first idea of active sampling is to increase the number of slices only for a subspace S with high uncertainty in its contrast. To quantify the uncertainty of the estimated contrast in S we use the standard deviation of contrasts among all slices for S. When the standard deviation reaches a threshold, we increase the number of slices and update the relevance estimation for S. The update is essential as we cannot simply add another contrast as the reasoning phase only considers the constraint with the highest estimate. As the contrast is normalized into the interval $[0, 1]$ we know that the maximum standard deviation is 0.5.

Proof. Let $X \in [0, 1]$ be a random variable and μ its mean. The variance is maximal when $p(X = 0) = P(X = 1) = 0.5$, and hence, μ is 0.5. The variance of X is then 0.25:

$$\begin{aligned} E\big((X-\mu)^2\big) &= P(X=1)(1-\mu)^2 + P(X=0)(0-\mu)^2 \\ &= \mu(1-\mu)^2 + (1-\mu)\mu^2 \\ &= 0.5 \cdot 0.5^2 + 0.5 \cdot 0.5^2 = 0.25 \end{aligned} \tag{13}$$

so that the standard deviation becomes $sdev(X) = \sqrt{0.25} = 0.5$. □

In our implementation, we define a threshold of 0.1 from which on we increase the contrast iterations by 100 additional slices. We further increase this number with each step of 0.1

until we reach the maximum of 400 extra iterations. In a standard configuration, RaR uses 100 contrast iterations. Increasing this number slows down the algorithm but comes with the advantage of more robust rankings. Still, a deviation in contrast always exists due to the nature of the contrast which is mostly visible in specific ranges of a feature. However, we notice in our experiments that the dynamic resampling increases the robustness of our rankings and that the efficiency does not suffer dramatically as the resampling only applies to a few high contrast subspaces. Further, we implement the resampling in a way that we do not need to create new slices for subspaces with more than one dimension. We instead derive new slices by shuffling the feature slices which by their combination leads to new slices for the subspace.

The following describes the second paradigm by first explaining which subspaces are of great importance to the reasoning and should be actively sampled. Two of RaR's biggest strengths are the identification of high order features and the ability to derive individual relevances from multi-dimensional subspaces. Unfortunately, we note that this deduction fails when the sampling phase misses some subspaces. An example gives an intuitive explanation. Let X be a dataset with four features $X^T = \{f_1, f_2, f_3, f_4\}$ where the first is relevant on its own and where $\{f_1, f_2\}$ form a high order feature. RaR now evaluates the subspaces as in Table 5.

Subspace	Relevance
$\{f_1, f_4\}$	0.4
$\{f_2, f_3, f_4\}$	0.6

Table 5: Exemplary results obtained by the sampling phase.

The following constraints can be derived: $r(f_1)+r(f_4) \geq 0.4$ and $r(f_2)+r(f_3)+r(f_4) \geq 0.6$. From the first constraint, it is not clear which of the two features creates the contrast or if both together form an interaction. Without further knowledge, regularization is the fairest option leading to scores of 0.2 for both f_1 and f_4. For our ranking, this is highly problematic as feature f_4 is not relevant at all. In our second constraint, we also do not know where the contrast comes from. Even more problematic is that f_4 is observed again in a relevant subspace which could lead to more relevance for f_4. The only way to prevent such a wrong deduction is to collect more subspaces. As already stressed, an increased number of subspaces lowers the efficiency of RaR, and not all subspaces are helpful in solving the issue for our toy example.

To create a good trade-off between quality and efficiency we collect 10% and at most 100 of the most relevant subspaces with more than one feature for further investigation. The exploration of these subspaces happens at the end of the sampling phase. For each of them, we assure that we know relevance scores for each included feature. This prevents from incorrectly assigning relevance to a noisy feature when it is combined with a relevant feature as in the first constraint from our example. Second, for subspaces with more than two dimensions, we further check their 2-dimensional subspaces to correctly identify high order features. By following this approach, we still increase the number of evaluated subspaces and lose some efficiency. However, as we only focus on subspaces which are highly relevant, we need much fewer evaluations than in an approach where we merely draw more random subspaces.

Another advantage which comes with our active sampling approach is that we can collect interactions rather than only incorporating them in the relevance scores. Let $S = \{f_2, f_3\}$ be

one of the subspaces which we actively sample after obtaining the results in Table 5. S can be considered as a high order feature when the summed relevances of single features are very low whereas the combined relevance is high: $\sum_{f \in S} rel_D(f) << rel_D(S)$. When we observe such a case we can speak of S as a high order feature and can even increase their importance by updating the constraint for S with a higher value. This makes sense as the relevance of a high order feature is split equally to all of its contained features. Additionally, we store the identified interactions in a variable for further investigations. The pseudocode in Algorithm 2 summarizes how we implement our active sampling for additional subspaces.

Algorithm 2 Active Sampling

INPUT: K: Knowledgebase from sampling phase
OUTPUT: K: Extended Knowledgebase, I: List of interactions

```
1:  procedure ExtendKnowledgebase(K)
2:      I ← [ ]
3:      best ← collect top 10% of multi-dimensional subspaces from K
4:      for S ∈ best do
5:          for f ∈ S do                          ▷ New constraints for single features
6:              if (f, rel_D(f)) ∉ K : K.append((f, rel_D(f)))
7:          end for
8:          if rel_D(S) ≥ 1.5 · Σ_{f∈S} rel_D(f) then      ▷ New constraints for interactions
9:              K.append((S, rel_D(S) · √|S|))
10:             I.append(S)                       ▷ Collect interactions
11:         end if
12:         if |S| > 2 then                       ▷ New constraints for 2-d subspaces in S
13:             for F ⊆ S and |F| = 2 do
14:                 if rel_D(F) ≥ 1.5 Σ_{f∈F} then
15:                     K.append((F, rel_D(F) · √2))
16:                     I.append(F)               ▷ Collect interactions
17:                 end if
18:             end for
19:         end if
20:     end for
21:     return K, I
22: end procedure
```

First, the algorithm identifies the highest scored multi-dimensional subspaces in the knowledgebase created during the sampling phase. From them, all missing 1-dimensional and 2-dimensional subsets are evaluated, and interactions are collected based on a defined threshold. Finally, the procedure returns the extended knowledgebase and a list of interactions.

Another possible idea of active sampling is to prune the search space and thus improve efficiency. When we observe a multi-dimensional subspace with a low contrast, we can prune all of its subsets due to the information monotonicity of relevance. However, we did not study this interesting path and leave it open to future works.

4.6 Weighting Strategies

In this section, we finally present our weighting strategies to tackle missing values inside RaR. First, we describe the general idea of a weighting strategy and differentiate it from traditional solutions such as deletion and imputation. Second, we propose different approaches for the weight function from Definition 4.6 and discuss their strengths and weaknesses.

By our weighting strategy, we model the uncertainty of a missing value to be inside a specific slice. This comes with two benefits: even partly observed samples can have an impact on the conditional distributions and no fixed estimation for a missing value is needed. A classical deletion approach removes all incomplete samples even though most of the attributes of these samples are available. Such a deletion can potentially eliminate all samples and is inferior to our weighting approach which makes better use of the observed data. In contrast, imputation completes the dataset by finding estimates for missing values and thus makes full use of the observed data. However, imputation creates single substitutes for missing values which can differ significantly from their real counterparts and thus lead to significant errors. Further, imputation makes assumptions on the data and missing mechanisms and requires features to be related to each other. Imputation influences global statistical measures, e.g. mean and variance, and can potentially add bias. Additionally, imputation can introduce artificial patterns or make them disappear. There exist techniques such as MICE [10] which overcome many of these weaknesses of single imputation techniques by modeling the uncertainty of missing values using multiple imputations and thus deriving various estimates for each missing value. However, these estimates are usually combined by the mean to obtain single values required by most feature selection algorithms, e.g. to measure distances between samples. By this compression, the uncertainty of imputation gets lost.

In our strategies, the weight is either assigned uniformly to all missing values of a feature or imputation techniques are used to estimate the weights for samples independently. The difference to classical imputation is that we do not use the imputed values to construct the slices but only to assign weights. We expect this to be more robust than treating imputed values equally to observed values. Further, our methods do not require fixed estimations for missing values. Instead, we only use the imputations to model the probability of a value being in a slice. The next subsections describe concrete weighting strategies.

Zero-weighting

As already mentioned in the reasoning section, the simplest weighting strategy is to assign a weight of zero to all missing values. This means that we ignore all incomplete samples in the estimation of conditional distributions. However, a deletion in subspaces, where only a part of all dimensions is selected, removes much fewer samples than a global deletion. The following formula expresses the probability that a sample does not contain any missing values:

$$P(\text{"complete sample"}) = (1 - \lambda)^{DIM} \ . \tag{14}$$

Thus the probability of a sample being complete decreases exponentially with the dimensionality. Let us consider an example dataset with 500 samples and 20 features. The maximum subspace size considered by RaR equals three in our work. Table 6 describes how many samples are expected to be still available after removing incomplete samples.

	0.00	0.05	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90
1-d	500.0	475.0	450.0	400.0	350.0	300.0	250.0	200.0	150.0	100.0	50.0
2-d	500.0	451.2	405.0	320.0	245.0	180.0	125.0	80.0	45.0	20.0	5.0
3-d	500.0	428.7	364.5	256.0	171.5	108.0	62.5	32.0	13.5	4.0	0.5
20-d	500.0	179.2	60.8	5.8	0.4	0.0	0.0	0.0	0.0	0.0	0.0

Table 6: Table showing the expected number of samples left after removing incomplete samples for different missing rates and dimensions.

Whether a global deletion leads to a loss of nearly all samples for missing rates from 20% on, the zero-weighting strategy still keeps enough instances for robust estimations of conditional distributions. Further, this strategy benefits from the general subspacing idea of RaR which evaluates each feature in multiple subspaces. In consequence, an incomplete sample in subspace S_1 can still be complete in another subspace S_2. Another difference to a traditional deletion approach is that we also consider target values for incomplete samples in the estimation of the marginal distribution. This allows detecting features where the missingness is related to a specific class. In such a scenario, incomplete instances never influence the conditional distributions which always leads to high contrast towards the marginal. The detection of such informatively missing features is impossible for traditional deletion or imputation techniques. However, the zero-weighting strategy cannot be applied anymore for datasets with high missing rates and only a few samples as too few instances will form the conditional distribution. This becomes even worse for higher dimensional subspaces in which we need to select more samples.

Fault-tolerant-weighting

In such cases, robust weighting strategies are required to assign weights to missing values and hence, allow incomplete samples to have an impact on the conditional distributions. Our fault-tolerant weighting strategy follows the idea of [25] which presents a definition of fault-tolerance for subspace clustering with missing values. In our approach, we assign a weight of one to all missing values and define an additional function which checks if the number of missing values for a sample exceeds a maximum. In this case, we ignore the instance in the estimation of conditional distributions. The function for fault-tolerance is defined below.

Definition 4.2 (Fault-tolerance)**.** *Let $S \subseteq X^T$ be a subspace of dimension $k \geq 1$. We define a function $fault\_tolerance : \mathbb{R}^k \to \{0, 1\}$ which describes if a sample $x_n \in S^T$ contains a reasonably small amount of missing values and can thus be considered,*

$$fault\_tolerance(x_n) = \begin{cases} 1 & \text{if } \sum_i^k \mathbb{1}(x_{ni}, \text{"?"}) \leq \lfloor \frac{k}{2} \rfloor, \\ 0 & \text{otherwise.} \end{cases}$$

The final slice for a subspace is created by a logical *and* between the subspace slice following Definition 2.9 and the binary vector for fault-tolerance computed as described above. The big advantage of the fault-tolerant strategy over zero-weighting is that incomplete samples are used under the promise that all observed values fulfill the slice condition and that the number of missing values is reasonably small. The fault-tolerant strategy makes thus better use of

observed values and allows the estimation of conditional distributions for larger missing rates and fewer samples. However, the fault-tolerance only improves subspaces with more than one dimension and gives much weight to missing values without considering their uncertainties.

Probabilistic-weighting

The problem with both approaches presented before is that they cannot keep the number of samples inside a slice constant. This is problematic in two different ways. In the zero-weighting strategy, the number of samples considered for the estimation of conditional distributions can be deficient leading to large contrasts and deviations so that the ranking becomes unstable. For the partial strategy, the number of samples can increase as the conditions only need hold for observed values. This usually leads to a smaller contrast for higher dimensional subspaces as the conditional and marginal are more similar. The contrast estimated in one dimension, however, is high as the fault-tolerance does not allow missing values here. This underestimates relevance in higher dimensions and prevents from identifying feature interactions.

With our probabilistic weighting, we aim to find a fair way of splitting weight to all missing values. Therefore we weaken our binary slice definition and model the uncertainty of a missing value following into a slice. As we aim to select $\alpha_k \cdot N$ elements in a k-dimensional subspace, a natural choice is to assign a weight of α_k to each missing value. This selection comes with the benefit that the number of samples used for the estimation of conditionals is kept constant over all subspaces. The number of samples selected for each feature in a subspace equals:

$$n\_samples = \alpha_k \cdot (1 - \lambda) \cdot N + \alpha_k \cdot \lambda \cdot N \tag{15}$$

$$= \alpha_k \cdot N \ . \tag{16}$$

As $\alpha_k = \alpha^{1/k}$ is chosen so that the final number of samples selected for subspaces of different dimensions is constant, assigning a weight of α_k to each missing value in a k-dimensional subspace keeps the slice size constant. Thus the probabilistic strategy does not over- or underestimate relevance as both strategies presented before. Further, it does not rely on any assumption on the data and does not add computational complexity. However, based on the formula it can be seen that a lot of the total weight is split equally to missing values. This leads to similar conditionals and marginals and finally a lower contrast. Nonetheless, a lower contrast makes sense for features with a high λ as there is simply not enough data.

One possible disadvantage of assigning α_k to missing values is that the probability of falling into a slice is not constant under the assumption that the data follows a normal distribution. This means that it is more likely for missing values to fall into a slice around the mean than in an extremal region. Therefore, we propose a second instantiation of our probabilistic weighting strategy which assumes a normal distribution for continuous features and uniformly assigns weight to unknown values of a feature considering the specific slice range. The weight equals the difference between the cumulative distribution function for the feature at the maximum and minimum value of the slice. For discrete features, the probability of falling into the slice equals the relative frequency of the number of observed samples for the categories selected by the slice. This distribution-based weighting has the advantage that it considers the slice location and does not weaken the contrast for slices far away from the mean. However, it makes assumptions on the data and adds computational complexity.

Single-imputation-based-weighting

All previous approaches assign weights uniformly to missing values given a specific slice. This has the disadvantage that it generally leads to more similar conditionals and marginals so that the contrast vanishes. We propose a different set of strategies which assign weights to unknown values individually by considering the observed attributes of the samples. The most intuitive way to split the weights is to use an imputation function to obtain estimates for each missing value and then to check if the estimations fall into the slice. This comes with the advantage that the estimated values do not need to perfectly match the real value as long as they are similar. The imputation is performed on all dimensions at once as imputation requires a reasonable amount of features and does not work on low-dimensional subspaces. Further, this is more efficient than performing imputation in every sampled subspace.

The total weight assigned to missing values follows the same idea as the probabilistic slicing which keeps the slice size constant. Thus the weight assigned by the approaches relying on single imputation equals $\alpha_k \cdot \lambda \cdot N$ for each feature. In the following, we present two instantiations of the single-imputation-based weighting strategy. The first strategy splits the whole weight equally among samples whose estimates are closest to the center of a slice. The center is given by the average of the maximum and minimum of a slice. Further, we only consider samples inside a certain radius around the center and limit the weight for each feature to a maximum of one. In the case where we observe fewer samples inside the radius than needed, we split the rest of the weight equally to all other missing values of the feature. This approach has the advantage that it splits more weight to samples which are likely inside the slice. This is more focused than previous approaches but requires the definition of a radius and can be unreliable as a binary decision is made. A fairer approach is to assign weight to missing values based on the distance of their estimations to the center. In this approach, small changes in the estimation have only a small impact on the weight assigned which is more robust than the radius approach. In our implementation the weight for each missing value x_{nf} in a feature f is assigned based on the imputed values $\hat{x}_{nf}$ as follows:

$$distance_{normed}(x_{nf}, center) = 1 - \frac{distance(\hat{x}_{nf}, center)}{max_{i=[0,N]}\big(distance(\hat{x}_{if}, center) \cdot \mathbb{1}(x_{if}, \text{"?"})\big)} \tag{17}$$

$$weight(x_{nf}, center) = \frac{distance_{normed}(x_{nf}, center)}{\sum_{j=0}^{N} distance_{normed}(x_{if}, center)} \cdot \alpha_k \cdot \lambda \cdot N\ . \tag{18}$$

We calculate the distance of an estimate to the slice center and normalize it by the maximum distance. We further invert the distance as higher weights should fall to values whose estimates are closer to the center. Finally, we normalize the distances by their sum so and multiply them with the weight we want to split to all missing values. Thus the slice size is kept constant. This approach does not suffer from slightly incorrect estimates but can lead to more similar conditionals and marginals compared to the radius-based strategy. Similar, we assign weights to discrete features a distance of one if an estimated class is in a slice and zero otherwise.

The main advantage of the single-imputation-based strategies is that they assign weight more focused to samples which are more likely to fall into a slice. However, this requires the choice of an imputation method and adds computational complexity. Further, single imputation can produce wrong estimates propagating to a biased ranking of poor quality.

Multiple-imputation-based-weighting

More reliable imputation techniques perform multiple imputations to the data and obtain a set of estimates for each missing value. However, the modeled uncertainty gets lost when the estimates are combined using the mean to obtain single scores as required by most algorithms. However, running RaR on different imputed versions of a dataset is not an option considering efficiency. To overcome these issues we propose a multiple-imputation-based weighting strategy. In a first step, multiple imputation, e.g. using MICE, is performed to obtain different estimates for each missing value. Similar to the radius approach, we assign weights based on the relative frequency of estimations falling into the area around a slice center. The difference here is that we weaken the strict decision of falling into a slice or not by considering multiple estimations which is more robust. However, this approach is even more complex and requires sophisticated multiple imputation techniques. Further, this approach consumes a lot of memory for large datasets as it needs to store many completed versions of a dataset.

Example & Discussion

All of the weighting strategies described before are able to assign weights to incomplete samples in a subspace. The weighting strategies are applied to each feature independently, except the fault-tolerance strategy which requires the number of missing values for a sample over all subspace dimensions. The independent creation of slices is a basic requirement of RaR enabling appropriate handling of mixed data and is hence, contained by our weighting strategies. The strategies either uniformly assign weights to missing values or independently based on imputation. Table 7 and Table 8 show toy datasets which clarify how the strategies operate. We further note that we keep the tables as simple as possible and only use them to exemplify the weight assignment. However, they do not represent a real scenario in which α and the slice range would change with the dimension and where the weights in the *alpha* column would be constant. Still, the tables serve well in explaining the strategies while hiding complexity.

	Dataset			$\{f_1\}$				$\{f_1, f_2\}$			
n	f_1	f_2	f_3	zero	fault-t.	alpha	distr.	zero	fault-t.	alpha	distr.
1	?	0.5	1	0	0	0.5	0.68	0	1	0.5	0.68
2	1.5	?	0.7	0	0	0	0	0	0	0	0
3	1	0	2	1	1	1	1	1	1	1	1
4	-1.2	2	-0.5	0	0	0	0	0	0	0	0
5	-0.7	?	0	1	1	1	1	0	1	0.5	0.68
6	?	0.2	?	0	0	0.5	0.68	0	1	0.5	0.68
7	?	?	?	0	0	0.25	0.46	0	0	0.25	0.46

Table 7: Example showing a dataset containing seven samples, three features and first group of strategies. The weights which would be assigned to each sample are given for a slice $[-1, 1]$, $\alpha = 0.5$ and a probability of 0.68 of falling into $[-1, 1]$.

Table 7 shows that the zero-weighting strategy ignores samples containing missing values. When adding f_2 to feature f_1 this becomes worse as even more samples are ignored and only

one sample is left. The benefit of subspacing is visible in sample five which is ignored in the subspace $\{f_1, f_2\}$ but considered in the subspace $\{f_1, f_3\}$. In contrast to zero-weighting where the slice size decreases with more dimensions it increases for the fault-tolerance strategy, e.g. sample one is considered in $\{f_1, f_2\}$ even though its value for the first feature is unknown. For 1-dimensional subspaces it shows the same results as the zero-weighting strategy as the fault-tolerance only applies for higher dimensional subspaces. However, this strategy selects almost all samples in higher dimensions and thus leads to small contrasts. The probabilistic approaches do not have the issue of ending up with too few samples and can even assign weights in one dimension. The difference to the fault-tolerance strategy is that the missing values have a weight lower than one and thus less impact on the conditionals. The only change between the instantiations based on alpha and the slice values is the numeric value assigned to missing values. All these strategies are easy to compute and do not add much complexity to RaR. Nonetheless, the uniform splitting of weight can lead to vanishing contrast for high missing rates. Table 8 illustrates the imputation-based approaches.

	Dataset			Single Imp.			Multiple Imp.	$\{f_1\}$		
n	f_1	f_2	f_3	$\hat{f}_1$	$\hat{f}_2$	$\hat{f}_3$	$\hat{f}_1$	radius	distance	multiple
1	?	0.5	1	**0.5**	0.5	1	**[0.5, 1.2]**	0	0.46	0.5
2	1.5	?	0.7	1.5	**-0.5**	0.7	1.5	0	0	0
3	1	0	2	1	0	2	1	1	1	1
4	-1.2	2	-0.5	-1.2	2	-0.5	-1.2	0	0	0
5	-0.7	?	0	-0.7	**1.5**	0	-0.7	1	1	1
6	?	0.2	?	**0.8**	0.2	**1.5**	**[1.1,1.2]**	0	0	0
7	?	?	?	**0.45**	**1.1**	**1.3**	**[0.45, 0.3]**	1	0.54	1

Table 8: Example showing a dataset containing seven samples and three features as well as an imputed version. The weights assigned to each sample based on the imputed values (bold) are shown for a slice $[-1, 1]$ and a total weight of one for the missing values.

The table shows the same dataset as before but adds an imputed version derived by an arbitrary imputation. Further, it shows multiple imputations for feature f_1. The slicing is only illustrated on a single feature as the slice creation is isolated for each feature and the combination just multiplies the feature slices. In the radius-based strategy, the imputed value for sample seven is closest to the slice center located at zero. Thus sample seven gets the full weight. Considering that the estimated value for sample one is very close to the value of sample seven this decision seems risky. The distance-based strategy weakens the risk by assigning weights based on the distance. Still, as the estimate for sample seven is closer to the mean it gets more weight. The farthest estimate, for sample six, does not obtain any weight. The assigned weights are only this high for the distance-based strategy as only a few missing values are present. For larger datasets, the weights are smaller and more similar so that a comparable effect to the probabilistic strategy can be observed, which is that the contrast vanishes. The approach based on multiple imputation checks how often the estimated values fall into the slice. The multiple imputations, in this case, show that the value for sample one is underestimated by single imputation and that the real value is probably higher. However,

it gets less weight than sample seven where we can be surer that it falls into the slice. The multiple-imputation-based strategy seems to be superior as it incorporates the uncertainty of missing value estimations. However, the slice size is not constant and more memory is required to store the multiple estimations. In summary, the imputation-based strategies use more knowledge to predict the probability of missing values falling into a slice. This, however, comes with increased complexity, runtime and the need for an appropriate imputation method.

4.7 Parameter Analysis

RaR uses several parameters which can be tuned to fit datasets of different characteristics. However, hyper-parameter tuning is a time-consuming process and a robust parameter setting is desirable. In [60] the parameters differ significantly between all datasets. The work in [7] provides formulas and discussions which can help to find the right settings. Following these ideas, we continue the work on finding a robust parameter setting to improve the generalization and handling of missing values of RaR. Table 9 shows the most critical parameters.

Parameter	**Description**
α - slice size	Percentage of all samples considered in the conditional distributions during contrast estimation
k - max. subspace size	Maximum dimension of subspaces selected in the sampling or ranking phase
n - sampling iterations	Number of subspaces evaluated in the sampling phase
m - contrast iterations	Number of conditional distributions used to estimate contrast
$\beta_{boost}, \beta_{correlation}$ - Eq. 12	Factors describing the influence of slicing in edge regions and of correlated missingness

Table 9: Overview of the most important RaR parameters and their meaning

The following subsections discuss the impact of these parameters on different datasets and propose default parameter settings.

Slice size - α

We determine α as the parameter with the most crucial impact on the ranking. Further, it is the parameter which differs the most in the work of [60] where it ranges from 1% to 20%. The parameter α determines the number of samples which are used to estimate the conditional distributions. A large α is more robust but leads to lower contrast as the marginal and conditional distributions look more similar. This is especially a problem for subspaces of higher dimensionality as can be seen by the exponential increase in $\alpha_k = \alpha^{1/k}$. For an α of 0.1 already 63% of all samples are selected for each feature in a 5-dimensional subspace. With

such a large slice the contrast vanishes and high order features cannot be detected anymore as the contrast measured in 1-dimensional subspaces is likely to be larger even for weakly relevant features. Further, a large α prevents from sensing contrasts hidden in small or extreme value ranges. A small α, on the other hand, leads to unreliable results as too few samples are used to construct the conditional distributions. However, due to multiple contrast iterations, the contrast becomes more reliable and α should be as small as possible. During our work, we notice that some datasets need a significantly larger α. One reason is that some real-world datasets contain samples sorted by the target vector. This leads to very high contrast in each slice of RaR as it is very likely that all samples are of the same class. We solve this issue by shuffling the samples before applying RaR on the data. The second observation comes from multi-class problems. The number of samples inside a slice needs to be significantly larger than the number of classes. Otherwise, each slice leads to high contrast. To incorporate the number of classes we suggest to select α as defined below:

$$\alpha = \frac{n\_samples\_per\_class \cdot n\_classes}{n\_samples} \,, \tag{19}$$

where $n\_samples\_per\_class$ is a constant which we set to five in our experiments. Following this definition, we obtain an α which works well among a great variety of datasets. We further adjust α for 1-dimensional subspaces by $\alpha_1 = \alpha^{1/1.5}$ which slightly reduces the contrast there. We notice this to be helpful in the detection of high order features as contrasts get more comparable over different dimensions.

When being confronted with missing values, we need to adjust the number of samples selected in a slice according to the missing rate λ. By down-scaling $\alpha \cdot N$ with λ we assure that the required amount of samples can be selected. However, this can lead to a minimal number of samples inside the slice e.g. for the zero-weighting strategy. When the number of samples becomes lower than a threshold, we ignore the contrast derived from the slice as it is too unreliable. Also, we need a minimum number of slices to compute a robust estimate of relevance. When this fails, we ignore a subspace and do not add it to the knowledgebase. The minimum number of samples needed in a slice equals the number of classes and 30 slices need to pass this threshold. Note that approaches other than the zero-weighting strategy assign weights to missing values. Thus the number of samples in a slice is identical to its weight.

Maximum subspace dimensionality - k

The parameter k defines the maximum dimensionality of subspaces evaluated during the sampling phase. Obviously, limiting the maximum dimensionality to k prevents from finding high order features consisting of $k+1$ features or more. However, the contrast estimation for large subspaces is inaccurate and we expect most of the high order features being formed by a small number of features. Still, a large k is beneficial as it helps to cover more of the search space and makes it more likely to cover high order features. So we recommend increasing k with the dimensionality of a dataset. In [7], Bocklisch analyzes the impact of k on a synthetic dataset. The experiment shows that $k \in \{2, 3\}$ leads to the smallest classification error of a Random Forest classifier on their dataset. We thus decide to set $k = 2$ for smaller datasets with less than 15 features and $k = 3$ for larger ones.

Number of sampling iterations - n

The increase of k for datasets of higher dimensions is also motivated by the number of sampling iterations n. The parameter n defines how many subspaces RaR evaluates during the sampling phase. With a larger k the search space is covered better and fewer iterations are needed. The selection of n is difficult as it is hard to estimate if and how many high order features are present in the dataset. A small n is preferable due to efficiency and can achieve comparable results when there are no high order features. However, the presence of high order features is not known and n should be large enough to cover potential interactions between features. A solution for the setting of n is provided in [7]. The work presents a formula based on the probability of missing an s-dimensional subspace in a DIM-dimensional dataset when subspaces of dimensionality $k \geq s$ are sampled:

$$n \geq \frac{ln(\beta)}{ln\left(1 - \frac{\binom{DIM-s}{k-s}}{\binom{DIM}{k}}\right)} \,. \tag{20}$$

To keep the runtime reasonable, we set $s = 2$ which means that the probability β defines the chance to miss a 2-dimensional subspace in the sampling phase. We further set $\beta = 0.05$ for datasets with $DIM \leq 15$ and $\beta = 0.01$ otherwise. We use a lower β for higher dimensions as we increase the maximum subspace size there and thus are more likely to cover 2-dimensional subspaces. Figure 6 shows the number of subspaces for an increasing dataset dimensionality.

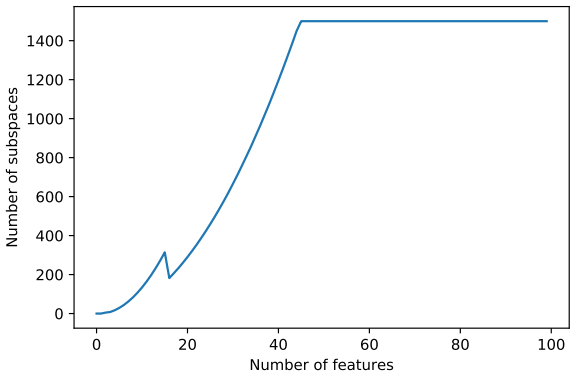


Figure 6: Number of subspaces evaluated in the sampling phase for a specific dimensionality.

The plot shows that the number of subspaces increases exponentially with the dimensionality. The number of sampling iterations is further limited to 1500 to ensure efficiency. The cut from feature 15 to 16 shows the point where we increase k. When dealing with missing values we find it helpful to increase n based on the missing rate λ. This makes sense as features with missing values can be evaluated better when they are part of multiple subspaces. The final number of subspaces is calculated by formula 20 multiplied by $\lambda/2$.

Number of contrast iterations - m

The number of contrast iterations determines how many slices are used to estimate conditional distributions. As this follows a random process, a larger m leads to more robust results. This

robustness comes with the issue of inefficiency and m should lead a good trade-off between both. Bocklisch further [7] conducts an experiment to assess the influence of m on the ranking quality. The analysis shows that up to an m of 100 the quality increases significantly. For larger values, the quality stagnates even though the runtime further increases linear in m. The setting of $m = 100$ makes sense for an α of 0.01 as most of the samples should be covered by at least one slice. Assuming that α is small, we propose to use 100 contrast iterations m as the default. Further, we suggest increasing m similarly to the number of sampling iterations to be more robust in the presence of missing values.

Boosting factors - β_{boost}, $\beta_{correlation}$

The last two parameters describe the factors for the relevance measured in extremal regions and for the missingness correlation as described in equation 12. The sum of both parameters should be significantly lower than 0.5 which means that relevance and redundancy have the largest impact on the final ranking scores. Setting β_{boost} to value larger than zero can be useful when contrast is hidden in the extremal regions missed by the general slicing approach. However, the contrast in extremal areas should not dominate the relevance of a feature and we thus decided to keep this parameter small with a value of 0.1. For datasets containing many classes, we notice an unreliable estimation of the feature boost so that the boost should either be dismissed or measured over a larger slice. The second parameter $\beta_{correlation}$ is useful when the missingness of a feature is correlated to the presence of another feature. In contrast to the other boost parameter, we can safely set $\beta_{correlation}$ to 0.1 or similar as the correlation only has an impact on the ranking if there really is a correlation. Otherwise, it has an equal effect on all features without changing the ranking.

4.8 Runtime Analysis

Bocklisch [7] and Shekar [60] show superior scalability of RaR compared to wrapper-based and similar to filter methods which do not incorporate high order features. This work extends RaR by introducing weighting strategies for missing values, active sampling, an improved ranking phase as well as some smaller changes. We discuss the runtime complexity of our extended version of RaR in this section and compare us with the complexity in [60] to analyze how our extensions change the runtime. Therefore, we split the runtime into four phases: data preparation, sampling, reasoning, and ranking. We perform our theoretical runtime analysis with regards to the complexity of operations and make assumptions about vector and matrix operations which e.g. allow calculating m operations at once. The number of performed operations is of course still m but as they can be executed in parallel the factor m is removed in our analysis for such cases. First, we develop the runtime of the zero-weighting strategy which serves as a basis for the analysis of all other weighting strategies.

Data preparation

The data preparation phase includes the creation of the slice cache as well as index structures which help to improve the efficiency of the following phases. In particular, the check for missing values is costly and thus we compute a binary missing value indicator matrix beforehand.

Therefore each value needs to be checked if it is missing or not which requires $N \cdot DIM$ evaluations. However, it is possible to perform all these evaluations at once using a vectorized matrix operation on X, and thus the detection of missing values can be performed in $O(1)$. The preparation also includes the caching of sorted indices for each feature. These are used to improve the efficiency of the slice creation and the calculation of statistical tests. We sort all features in X at once leading to a complexity of $O(N \cdot log(N))$. Finally, we initialize the cache and fill it with slices for each feature and each dimension, as α changes with the dimension. The slice creation takes constant time when the sorted indices are available. Therefore we select samples of the original feature based on a consecutive number of sorted indices. Further, a weighting strategy needs to be applied to complement the slice. In the case of zero-weighting, we set all missing values to zero in constant time as the position of missing values are known. Initializing the cache with m slices for DIM features and k dimensions leads to a complexity of $O(DIM \cdot k \cdot m)$. The total complexity of the preparation phase is thus:

$$O(N \cdot log(N) + DIM \cdot k \cdot m) \; . \tag{21}$$

Sampling phase

During the sampling phase, RaR evaluates n randomly selected subspaces. Therefore, each subspace needs to be sliced to obtain the conditional distributions. The cache already provides a matrix containing m slices for each feature and dimension. The only cost comes with combining the slices using matrix multiplication. As there are at most k features in a subspace, the cost for the combination is limited by k. RaR further calculates the contrast during the sampling phase based on KLD as described in section 4.2. Therefore RaR sorts the slice matrix of a subspace by using the sorted indices for the target vector Y. Then, a vectorized sum over all slices is performed to count the weights for each class of Y. Thus RaR only needs to calculate C sums, assuming that Y contains C unique classes. The total complexity of the sampling phase, without active sampling and boosting, is thus given by $O(n \cdot k \cdot C)$.

After the sampling phase, active sampling collects further constraints to improve the reasoning. First, RaR sorts all results based on their estimated relevances before selecting the top 10%. The sorting takes time in $O(n \cdot log(n))$ and the selection in $O(1)$. The active sampling evaluates single and 2-dimensional subspaces contained in the most relevant subspaces. Considering that many of these additional subspaces are already rated and that at most 100 subspaces are examined, the active sampling does not add a higher complexity to the sampling phase so that the first n evaluations dominate the complexity. The resampling also adds no significant complexity to the runtime as it only applies to highly relevant subspaces and only increases the constant factor m by a further constant factor which is at most five.

The sampling phase also measures contrast hidden in extremal regions which are likely to be missed by the general slicing approach. RaR calculates the feature boost for all DIM features based on $b << m$ slices whose evaluations take time in $O(C)$ as for regular slices. This leads to a complexity of $O(DIM \cdot b \cdot C)$ and a total complexity of the sampling phase in:

$$O(n \cdot k \cdot C + n \cdot log(n) + DIM \cdot b \cdot C) \; . \tag{22}$$

Considering that $n >> DIM$ the runtime of boosting is neglectable as the sampling phase

dominates it. Intuitively the evaluation of n subspaces is more expensive than sorting the n results even though the complexity could lead to different expectations. This is because calculating sums on a matrix is more expensive than comparing two objects based on a single number. Based on this observation, the final complexity of the sampling phase is given by:

$$O(n \cdot k \cdot C) \ . \tag{23}$$

The final complexity is very similar to the complexity of $O(n \cdot k \cdot N)$ in [60]. However, based on their paper it is not clear how they construct a slice for a feature in linear time without any preparation. Further, with our approach, we can obtain m conditional distributions by performing C sums which significantly improves the efficiency.

Reasoning phase

The reasoning phase deduces single relevances based on the n constraints created during the sampling phase. During our work, we do not make any changes to the optimization formula and obtain the same time complexity of $O(n + \sqrt{DIM} \cdot ln\frac{1}{\epsilon})$ as presented in [60]. The solution obtained by the optimizer converges to an ϵ-accurate solution in a reasonable time. The runtime of the reasoning phase is neglectable as the sampling phase takes much more time.

Ranking phase

RaR iteratively creates the final ranking. In each step, RaR estimates the redundancy of open features towards the already selected features leading to a quadratic complexity as in [60]. In our approach, we only select the first $\sqrt{DIM}$ features in an iterative procedure while storing and reusing the maximum redundancies of previous iterations. This allows to estimate the redundancy of features in later steps without performing any additional redundancy calculation. The approach of Shekar [60] further requires $\binom{n-1}{k}$ redundancies for each open feature at step n. By reusing former redundancies, we reduce this number to one and further assure that the subspace used for redundancy estimation in the next iteration contains the latest feature. By following this approach, we need to perform $O(\sqrt{DIM} \cdot DIM)$ redundancy estimations to select the first $\sqrt{DIM}$ features. Each redundancy calculation requires a slice creation for the subspace performed in $O(k)$ using the slice cache. The KS test statistic is computed for each of the m slices in $O(N)$ leading to a redundancy complexity of $O(k \cdot m \cdot N)$. Assuming that the calculation of the final score using equation 12 is in constant time we observe a complexity of $O(\sqrt{DIM} \cdot DIM \cdot k \cdot m \cdot N)$ for ranking the first features.

RaR calculates the scores for remaining features based on the stored redundancy estimations in $O(DIM)$. To complete the ranking, RaR needs to sort these scores in $O(DIM \cdot log(DIM))$ and append them to the current ranking. As the logarithm increases slower than the square root, the iterative selection of the first features dominates the ranking phase expressed by:

$$O(\sqrt{DIM} \cdot DIM \cdot k \cdot m \cdot N) \ . \tag{24}$$

Assuming that the sampling phase dominates the reasoning phase, that k, m and C are constant factors, and that $n >> N \cdot log(N)$ the final time complexity is:

$$O([N \cdot log(N) + DIM \cdot k \cdot m] + [n \cdot k \cdot C] + [\sqrt{DIM} \cdot DIM \cdot k \cdot m \cdot N]) \quad (25)$$

$$= O(N \cdot log(N) + n \cdot k \cdot C + \sqrt{DIM} \cdot DIM \cdot k \cdot m \cdot N) \quad (26)$$

$$= O(n + \sqrt{DIM} \cdot DIM \cdot N) \; . \quad (27)$$

Thus the ranking phase dominates the final complexity followed by the contrast estimations for the n subspaces during the sampling phase.

Weighting strategies

This subsection explains how different weighting strategies influence the time complexity of RaR. The fault-tolerant weighting strategy assigns a weight of one instead of zero and adds a check for the maximum number of missing values. This requires to count all missing values in each sample in a subspace which can be performed with a single query to the missing value indicator matrix. The comparison to a maximum and the multiplication of the fault-tolerance vector and the subspace slice take constant time. Thus the fault-tolerance strategy does not add further complexity and should result in an almost identical runtime.

The probabilistic strategies assign a constant weight to all missing values. Assigning alpha thus does not change the complexity. When the slice region influences the weight, RaR needs to estimate the probability of missing values falling into the slice from a distribution or by using the known relative frequencies for categorical features. As both instantiations only add tasks of constant complexity, they do not change the overall complexity.

When relying on imputed values for the weight assignment, RaR needs to perform an imputation step in the data preparation phase. Depending on the imputation method this can be costly, especially for multiple imputation methods such as MICE. Comparing the runtime complexities of imputation techniques is out-of-scope of this work which only discusses the extra complexity added to the weight assignment here. In the single-imputation method with radius instantiation, the weight splits equally to the w samples whose estimates are closest to the slice center. RaR calculates the distances for all estimates in a single operation. Finding the w closest estimates takes almost linear time when w is small and a smart data structure such as a heap is used, $O(N + w \cdot log(w))$. The distance instantiation does not need to find the w closest estimates and directly calculates the weight based on the distances in constant time. It thus does not alter the complexity, except for an extra imputation step. The radius approach, however, adds complexity to the slice creation during the cache creation. Adding the factor N to it leads to $O(DIM * k * m * N)$ which is still dominated by the ranking phase.

In the multiple-imputation-based strategy, RaR assigns weight by counting the relative frequency of an estimate falling into a specific slice. Therefore RaR calculates distances from estimations to the slice centers and finally compares them with a specified radius. This operation takes constant time using vectorization. Checking for each of the multiple estimates leads to a complexity linear to the number of multiple imputations. Also for this strategy, the total runtime is dominated by the redundancy phase, ignoring the complexity of imputation. Due to the many parts of RaR, we further analyze its runtime throughout various experiments.

5 Evaluation

This section provides experimental results about the quality and efficiency of RaR and other established feature selection techniques with a focus on the handling of missing values and the impact of imputation. Further, the experiments verify if our adjustments such as active sampling improve the quality of RaR. Based on the experimental results, we review the theoretical discussions from previous sections and provide answers to our initial research questions.

The previous works on RaR in [7] and [60] have already compared their implementations to well-known techniques, such as ReliefF, CFS and mRmR, using artificial and real-world datasets. They conclude that RaR outperforms many of them when facing mixed feature types and high order features. The authors further demonstrate the high scalability and quality of RaR on large datasets. Our experimental evaluation aims to show that our extensions further improve the quality and efficiency of RaR and that our weighting strategies properly handle missing values without interfering efficiency. In particular, we are interested in evaluating how the rankings change with increasing missing rates, how imputation influences the ranking and which impact the missingness mechanism has on the quality of selected features.

We implement RaR in Python and provide it on Github[1] together with implementations of competitor methods as well as code for downloading and generating datasets. For reproducibility, we provide the scripts to run the experiments together with all results and configurations. To achieve a fair evaluation, all parameters of RaR, competitors, and classifiers are kept constant during the experiments, unless stated otherwise, and multiple runs are performed to increase the statistical significance of results. We conduct all experiments on the same machine with an Intel Core i7-4720HQ CPU on a clock rate at 2.6 GHz and 16GB RAM.

To the best of our knowledge, we are the firsts to conduct an extensive comparison of feature selection in the presence of missing values. Our experimental evaluation includes various synthetic datasets, ten datasets taken from OpenML, missing rates from $0-90\%$, four missing mechanisms and a list of ten approaches including our version of RaR. We present the different competitors, datasets and metrics used for evaluation in the next subsection. Finally, we expose and analyze our results on synthetic and real-world datasets in Sections 5.2 and 5.3.

5.1 Experimental Setup

Traditionally, extra preprocessing steps are used to tackle incomplete data and only a few implementations for algorithms which can cope with missing values directly are available. In an example, the widely used machine learning library scikit-learn [49] for Python does not allow missing values in its classifiers and feature selection approaches and only provides basic implementations for imputation techniques. Also, other libraries do not support missing values out-of-the-box so that we need to adjust them to include them in our experiments. This subsection presents the competitors including their adjustments as well as the data generation process and the metrics used for comparison.

[1] `https://github.com/SebastianRehfeldt/missing-value-aware-feature-selection`

Competitors

The competitors include traditional preprocessing steps, **deletion** and **imputation**, used to transform an incomplete dataset into a complete one. The deletion is implemented as a row deletion removing all incomplete samples. As imputation techniques, we select mean, kNN and soft imputation as well as MICE imputation which is a popular technique for multiple imputation. [10,17,46] Implementations for these approaches are taken from fancyimpute.[2] In our experiments, we perform deletion and imputation before running RaR which allows us to evaluate if the integrated handling of missing values is superior to an extra preprocessing step.

As competitors from the paradigm of embedded feature selection, we select **XGBoost** [12] and a **Random Forest** [8] which are both tree-based. XGBoost handles missing values by learning the optimal direction of missing values for each feature whereas incomplete samples are ignored or filled with the mean/median in Random Forests. The implementation for XGBoost comes from its official website[3] and for Random Forest from the Orange data mining library [15]. The importance of a feature in XGBoost equals the number of times it is used for a split and in a Random Forest usually the average information gain it has over all trees.

The Orange data mining library also provides missing value aware implementations for the filter methods **FCBF** [75] and **ReliefF** [55]. FCBF uses an entropy-based measure for feature selection, and ReliefF evaluates the discriminative power of a feature between classes on similar samples. ReliefF handles missing values in a modified probabilistic distance function and FCBF by adjusting the information gain used for the calculation of symmetrical uncertainty. We further use implementations for the filter methods **mRmR** [9] and **CFS** [76] from the scikit-feature Python module [41]. CFS selects features which are correlated with the target but not with each other and mRmR selects features based on the minimal-redundancy-maximum-relevance criterion. Both implementations rely on histograms so that we add discretization of numeric features to apply both methods on mixed data. Further, we replace all missing values with unique values. Based on the paper of Doquire et al. [20], we implement a filter method based on mutual information (**MI**). We use their partial distance strategy and extend it to handle mixed data by defining the distance between discrete values as zero if the categories match and one otherwise. In contrast to Doquire, we focus on classification and use the partial distance function in the mutual information estimator presented by Ross. [56] Even though the mutual information estimation applies to feature subspaces with more than one dimension, we only use it to assess single features due to efficiency reasons.

Tran et al. propose a wrapper-based method which uses a C4.5 classifier in particle swarm optimization (PSO) to select features. Instead of a C4.5 classifier, we use a decision tree implementation from Orange which can handle missing values and mixed data. Instead of PSO, we use sequential forward selection (**SFS** + **Tree**) in our experiments which allows to create a ranking and is more efficient than PSO with the parameters of Tran on small datasets.

To complement our competitors, we add a subspacing approach inspired by **RKNN**. [43] Li et al. propose an alternative to Random Forests by training and testing kNN-classifiers on randomly drawn subspaces. The importance of a feature equals the average classification accuracy among subspaces which include the feature. We do not implement their recursive

[2] `https://github.com/iskandr/fancyimpute`
[3] `https://xgboost.readthedocs.io/en/latest/index.html`

selection process as we aim for feature ranking and as we lack an efficient implementation of a missing value aware kNN-classifier. However, we provide a simple implementation using the partial distance strategy introduced by the MI estimator. Altogether, our competitors cover a broad range of feature selection techniques as well as methods for handling missing data.

During this section, we name different instantiations of RaR by the weighting strategy they employ. **RaR-deletion** and **RaR-partial** refer to the zero-weighting and fault-tolerant weighting strategy. **RaR-alpha** and **RaR-proba** belong to the probabilistic strategies which assign weights according to alpha or the slice position. **RaR-radius** and **RaR-distance** describe single imputation based strategies and **RaR-multi** the one based on multiple imputation. For small datasets, we rely on MICE imputation and for larger ones on soft imputation, erasing RaR-multi, due to efficiency. We further evaluate **RaR-category** similar to RaR-deletion but with the difference that we create a new category for missing values in discrete features in this case. All instantiations use the same parameter setting unless stated otherwise.

Datasets

Analyzing the strengths and weaknesses of algorithms requires a setup which covers a broad range of datasets with varying characteristics. As in the original works of Bocklisch [7] and Shekar [60], we rely on the generation of synthetic data as well as on datasets from OpenML. Synthetic datasets have the advantage that specific properties of a dataset such as the number of features, samples or feature types can be controlled. Further, the relevances of features, as well as high order features, are known which facilitates the evaluation of feature selection. We extend their data generation process, which itself is based on a NIPS [27] workshop on variable and feature selection, by the simulation of different missing rates and missing mechanisms.

In a short wrapup of [7], we generate complete datasets by first creating a feature matrix, assigning relevances to features and then computing the target vector based on the features and defined relevances. The feature matrix X contains independent features following a normal distribution $\mathcal{N}(0, 1)$. We further create dependent features as a linear combination of all independent features with randomly drawn coefficients. We further add gaussian noise from $\mathcal{N}(0, 0.1)$ to the features to simulate measurement inaccuracies. We compute the target vector as a linear combination of independent and dependent features where the coefficients determine feature relevances. The factors are zero for irrelevant and drawn uniformly from the range $[0.2, 1]$ for relevant features. The range $[0.2, 1]$ assures that relevant features have at least a minimal impact on the target. The result of the linear combination is binarized by applying the signum function to construct the final target vector. To account for labeling errors, a specific percentage of targets, e.g. 1%, is flipped. We simulate high order features, which mean a minimal set of features which are only relevant in their combination, by creating a hidden feature. We incorporate this hidden feature in the target computation but remove it before applying the feature selection. We compute such a high order feature as an xor of independent and irrelevant features. This assures that these features are irrelevant by themselves but relevant combined. We uniformly draw the relevance of a high order feature from the range $[0.2, 1]$ and split the relevance equally to features which form the interaction. Further, features can be part of multiple high order features to increase complexity. Finally, we provide discretization to simulate different feature types.

We further extend the data generation process by simulating missing values as deletion process on the complete feature matrix X. We provide MCAR, MAR and MNAR missing mechanisms as well as an approach where the missingness between features is correlated. We simulate MCAR by randomly removing a defined fraction λ of values from X leading also to an equal expected number of missing values for each feature. However, we cannot apply this procedure to gradually increase λ during an experiment as it does not assure that values removed for a small λ are also removed for a higher λ. We tackle this by creating a dummy matrix with the same dimensions as X and consisting of random values taken from $\mathcal{N}(0, 1)$ and finally remove the values in X with the λ smallest values in the dummy matrix. As the values in the dummy matrix are taken randomly from a normal distribution, the missingness of values in X follows MCAR. The MAR mechanism is used to simulate informatively/predictive missingness. In this scenario, only values for samples belonging to a specific class, the majority class, are removed. Figure 7 shows such a simulation on the Ionosphere dataset using the Python package missingno for missing value visualization. [6]

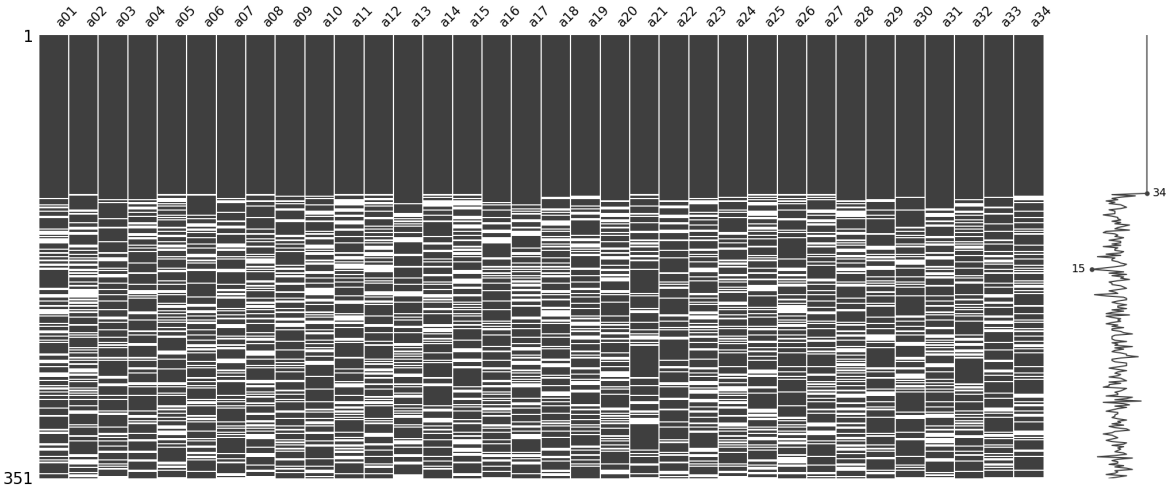


Figure 7: Bar plot showing the location of missing values according to MAR and $\lambda = 0.3$ in the Ionosphere dataset sorted by the class.

We first sort the dataset by its target so that we can observe that missing values are only present in one class, represented by holes in the bars. Thus the missing mechanism is an easy version of MAR. We further simulate MNAR by removing values based on their actual value. Thus the λ largest or smallest values are removed for each feature. However, this procedure only makes sense for relatively small values of λ. Let us assume $\lambda = 0.5$ and a feature f which follows the distribution $\mathcal{N}(0, 1)$. By our MNAR procedure, we end up with only positive or negative values for $\lambda \geq 0.5$. Inspired by the missingno package, we simulate correlated missingness. Therefore we create a binary dummy vector containing N values and whose ones indicate that a value will be removed. For each feature, we shuffle a random percentage ofvalues in the dummy vector and use the shuffled version to remove values in the feature. A large shuffle percentage leads to small missingness correlation of a feature whether it is high for small shuffle percentages. For negative correlations, we invert the vector and flip ones to contain a fixed λ over features. Figure 8 shows an example for this scheme.

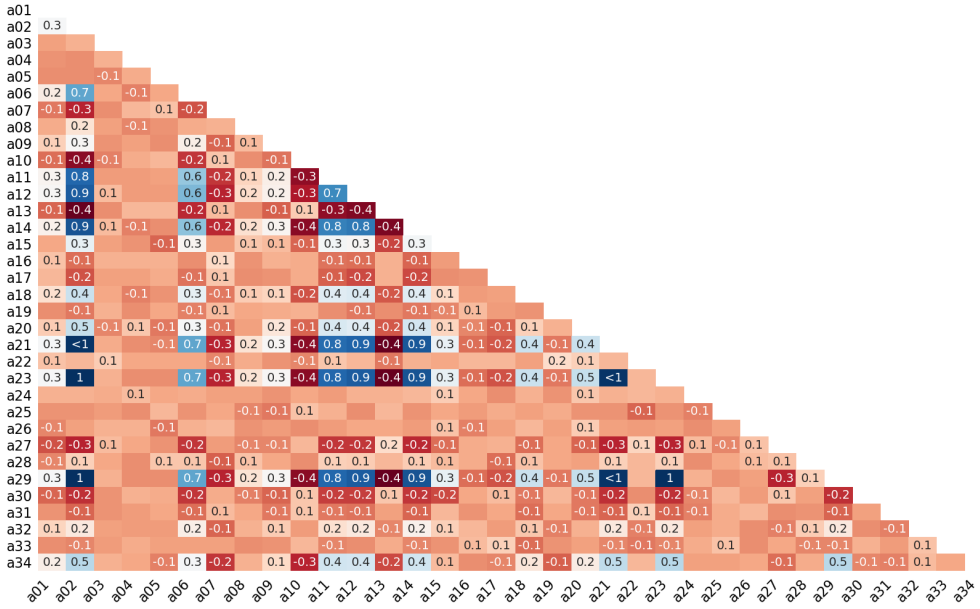


Figure 8: Correlation plot for Ionosphere visualizing the correlated missingness for $\lambda = 0.3$.

Dark blue colors represent high positive correlations while dark red colors indicate the opposite. We calculate the missingness correlation by Pearson's correlation coefficient applied to the missingness indicator matrix. Finally, we simulate a different form of predictive missingness, which is harder to identify, by integrating it into the data generation process. First, we create a hidden binary dummy vector whose ones are used to remove values in an irrelevant feature. Similar to high order features, we use the dummy vector in the target computation and remove it afterwards. By removing values in an irrelevant feature based on the dummy vector, the feature becomes relevant solely based on the missingness of its values.

To complement our experiments and to demonstrate applicability of the algorithms on real-world datasets, we evaluate them on ten datasets taken from OpenML as listed in Table 10.

	N	DIM	Classes	Maj. Class (%)	λ (%)	Inc. samples (%)	Type
Heart-c	303	13	2	54.46	0.17	2.31	mixed
Ionosphere	351	34	2	64.1	0	0	numeric
Semeion	1593	256	10	10.17	0	0	numeric
Madelon	2600	500	2	50	0	0	numeric
Musk2	6598	169	2	84.59	0	0	mixed
Isolet	7797	617	26	3.85	0	0	numeric
Hepatitis	155	19	2	79.35	5.39	48.39	mixed
Schizo	340	14	2	52.06	16.35	67.06	mixed
Vote	435	16	2	61.38	5.3	46.67	nominal
Soybean	683	35	19	13.47	9.5	17.72	nominal

Table 10: Overview listing the OpenML datasets including information about dimensions, classes, missing rates, percentages of incomplete samples as well as feature types.

We select datasets which cover different characteristics including varying dataset dimensions, binary and multi-class problems, with and without class imbalance, different feature types as well as different missing rates. Further, the datasets come from different domains, e.g. Semeion and Isolet from the field of natural language processing as well as Hepatitis, Musk2, Schizo, Soybean, and Heart-c from the medical/biological domain. Ionosphere contains measured radar returns from the ionosphere and Vote includes votes from U.S. congressmen. Finally, Madelon is an artificial dataset which contains high order features and which is widely used to measure the ability of feature selection methods to identify feature interactions. [30]

Metrics

We evaluate the quality of feature selection approaches on synthetic datasets by the cumulative gain (CG) of a feature set. [2] The CG is a popular metric from the field of information retrieval which uses knowledge about ground-truth relevances to score a ranking. The CG is calculated as the sum of relevances for all selected features, assuming that the true relevances sum up to one. On synthetic datasets, the CG calculation is straightforward as both the ground-truth relevances and the number of relevant features are known. On real-world datasets, this information is not available, and we use classification accuracies instead. In detail, we report the macro average F_1 scores for multiple classifiers which are calculated by averaging the harmonic mean of precision and recall for each class. However, the evaluation using the CG is preferable as we lack a perfect classifier. In our experiments, we perform multiple runs of each algorithm to increase the statistical significance of our results. Further, we create multiple versions for a synthetic dataset based on a fixed configuration and perform multiple missing value simulations on each created dataset. This is necessary as both steps rely on randomized processes and might lead to misleading or unreliable results. However, when performing multiple runs, the results get more meaningful and statistically significant. We provide dataset configurations for all experiments in our Github repository together with parameters specifying the number of runs, the missing rates, and algorithm parametrization. The parameters for each feature selection approach are, however, kept constant over all experiments, unless they are the subject of it, to obtain a fair comparison. For the experiments on synthetic data, we create five different datasets for a configuration and further repeat the missing value simulation five times leading to 25 runs for each approach. Missing rates from 0% to 90% are evaluated at 10% intervals to see how the quality changes with the missing rate. On real-world datasets, the data is fixed and we only perform five different missing value insertions. Further, we repeatedly apply three-fold cross-validation for each approach to determine the classification performance for specific classifiers and a number k of selected features. As reference classifiers, we provide a kNN classifier using the partial distance strategy and a decision tree implementation from the Orange data mining library. Further, we use a more efficient kNN implementation from scikit-learn for large datasets as well as a support vector machine and a gaussian naïve bayes classifier from scikit-learn. We apply mean imputation before using the scikit-learn classifiers to handle missing values. The impact of imputation on classification is, however, not the main subject of this work and can be neglected as the classifiers are the same for each feature selection approach. The following sections provide the results and give more detailed information about the experimental setup of concrete experiments.

5.2 Experiments on Synthetic Data

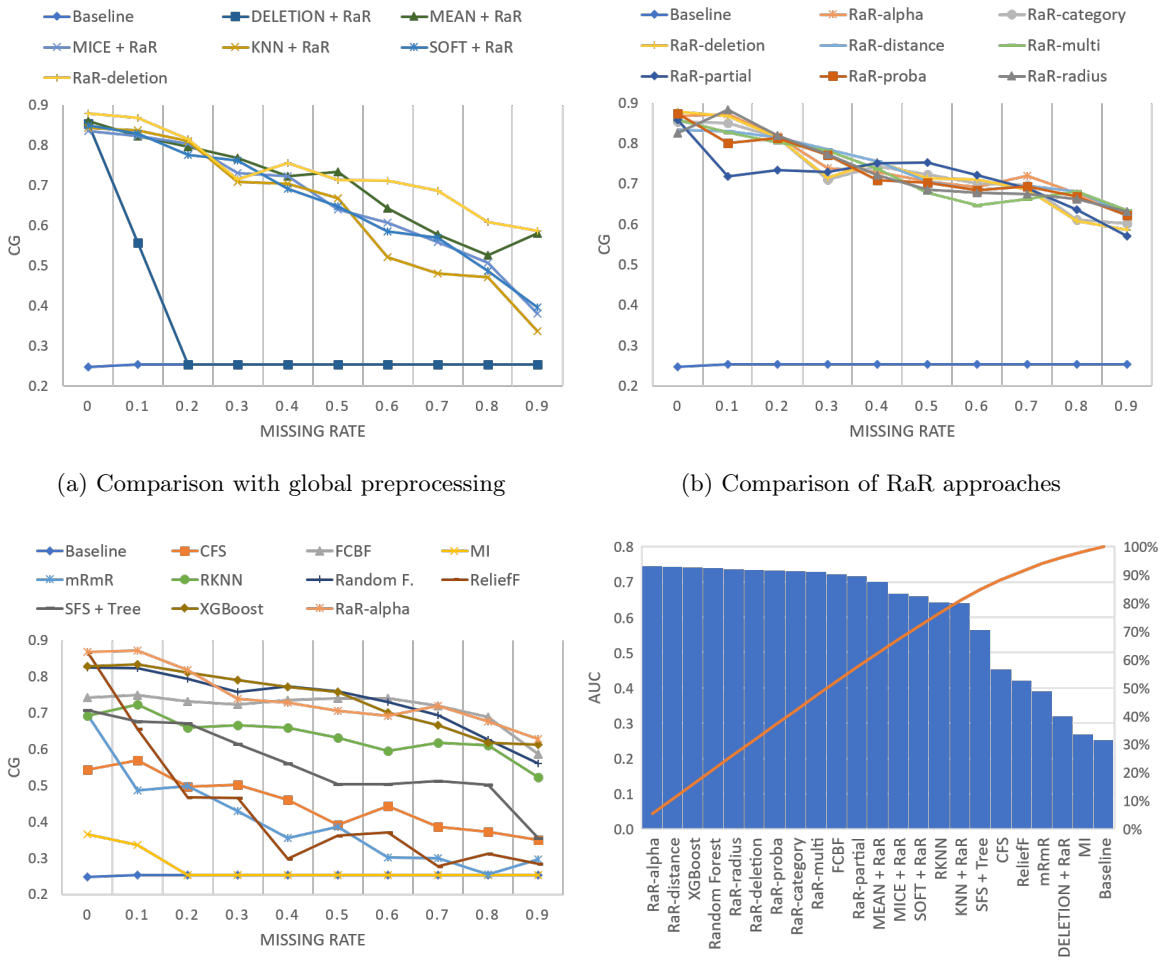
The generation of synthetic datasets allows us to simulate different characteristics of datasets and simplifies the evaluation of feature selection approaches as the true relevances for features are known. In each experiment, we create multiple versions of a dataset based on a configuration and simulate missing values according to the MCAR mechanism. By generating five datasets and performing five missing value simulations we perform 25 executions for each algorithm and each missing rate. This eliminates the effect of randomness on both generation processes and leads to meaningful results. Due to the additional dimension of the missing rate, we perform the experiments on relatively small datasets with 20 independent features and 500 samples. This, however, is a typical size in the OpenML repository. We evaluate ranking qualities by the CG at position k where k is the number of relevant features. We note that the results of different feature selection approaches become more similar when more features are relevant as chances increase to select a relevant feature by accident. We thus decide to provide two configurations: three relevant + one high order feature; and three high order features only. A high order feature hereby consists of two features which are only relevant in their combination. We further provide a configuration in which we discretize half of the features.

Starting with an experiment on how the qualities of rankings change with an increasing missing rate, we compare different weighting strategies and evaluate our active sampling approach in more detail. We further measure the effect of caching on the runtime and check how the different approaches detect informatively missing values. Finally, we evaluate the scalability of the methods with increasing dataset dimensions and use classifiers to determine the quality of the ranking concerning the task of classification.

Evaluation using an increasing missing rate

First, we evaluate the ranking quality over an increasing missing rate on a dataset with 500 samples and 20 independent features which contains three relevant and one high order feature.

Figure 9 plots the CG over an increasing missing rate for different competitors. The figures at the top compare global preprocessing techniques with RaR as well as the different weighting strategies. Figure 9 shows that RaR-deletion outperforms global preprocessing techniques. Especially a global deletion suffers from an increasing missing rate and is not able to create rankings for missing rates from 20% upwards. The baseline shows the quality of a random ranking and thus provides a lower boundary of the quality. This already indicates that a global deletion is inappropriate as it removes too many samples which is, however, not the case for a deletion in subspaces. Interestingly, the imputation techniques already show an inferior quality compared to RaR-deletion and mean imputation performs best among the imputation techniques. We believe that the reason for this is the lack of correlation between features and samples due to the synthetic data generation process. In Figure 9b we compare the different weighting strategies. The strategies seem to work almost identical, except for RaR-partial which shows a large drop for the missing rate of 10%. In comparison to the global preprocessing techniques, all weighting strategies seem to be superior with no clear winner. In Figure 9c we provide the results for the competitor methods which perform worse or at most similar to RaR-alpha. The MI estimator is hardly able to find high-quality features and also histogram-based methods such as CFS and mRmR have problems to create good



(a) Comparison with global preprocessing

(b) Comparison of RaR approaches

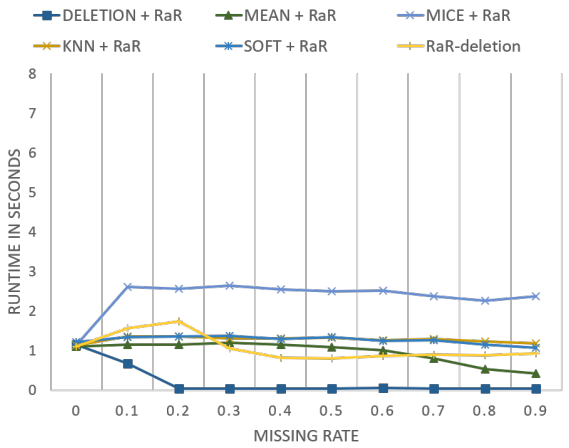
(c) Comparison with competitors

(d) Pareto plot of AUC scores

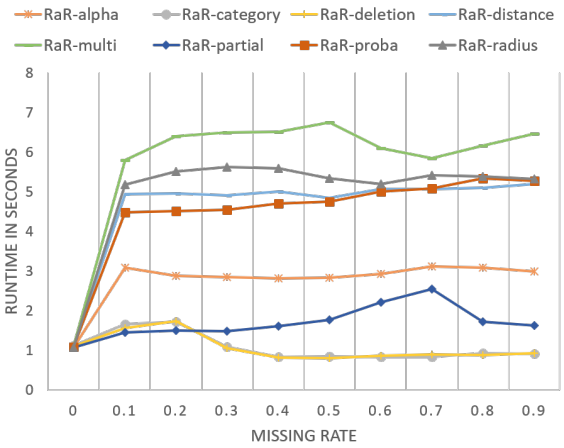
Figure 9: Ranking evaluation displaying the CG over missing rate.

rankings even for small missing rates. For SFS and ReliefF we observe good qualities on the complete data but the largest quality losses with an increasing missing rate. The other methods FCBF, RKNN, XGBoost, and the Random Forest perform similarly to RaR and do not suffer significantly from missing values. In Figure 9d we present the area under the curve (AUC) for the CG over missing rate for all methods. The Pareto plot displays the results sorted by their quality together with the cumulative weight. It shows that our weighting strategies perform equally well and better than preprocessing techniques and many competitors, except for XGBoost, Random Forest and FCBF. This also gets clear by the decreasing slope of the line representing the cumulative total. We further provide runtimes for all methods in Figure 10.

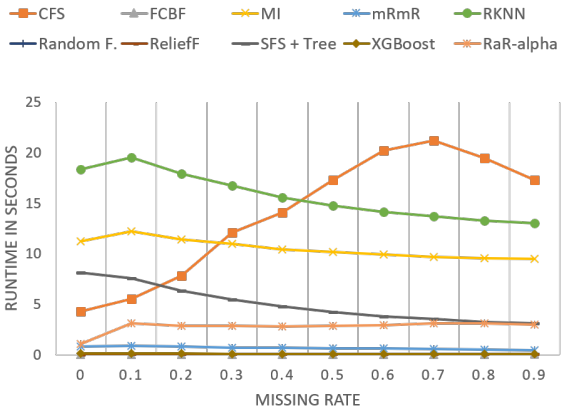
The runtime for the global deletion decreases to almost zero as can be expected from the figure before. In contrast, imputation generally adds complexity to RaR, except for the very simple mean imputation. MICE is the most complicated method and thus needs the most time to complete the dataset. KNN and soft imputation are less sophisticated and only add little overhead to RaR and thus almost show constant runtimes over the missing rate. RaR-deletion performs similar to them showing that the simple weighting strategy does not add significant



(a) Comparison with global preprocessing



(b) Comparison of RaR approaches



(c) Comparison of competitors

Figure 10: Runtime evaluation displaying the execution time over missing rate.

overhead to the missing data handling. The increase in runtime, in the beginning, is due to the additional checking for missing values which is computationally expensive. However, for larger missing rates more subspaces are ignored and the runtime decreases. In Figure 10b we present the runtimes for our weighting strategies. RaR-deletion and RaR-partial are significantly faster than the other methods. This is because the slice vectors for these approaches are binary whereas the other approaches assign real-valued weights to missing values. The operations on binary vectors can be optimized and performed faster during the runtime which explains the difference. However, there seems to be only a constant offset for RaR-alpha and RaR-proba from which RaR-alpha is clearly faster. The methods relying on imputation show the highest runtime explained by the additional runtime of MICE imputation. The plot further shows that the runtime does not increase significantly with the missing rate and that the weighting strategies only add little complexity to RaR, with the small exception for RaR-proba and imputation-based strategies which can be seen more dramatically in later experiments.

Comparing to competitor methods in Figure 10c, RaR shows equal performance to XGBoost, mRmR and the methods provided by Orange on complete data. In the presence of missing

values, RaR-alpha starts to use real-valued weights in the slices and thus its performances decreases in the first step. However, the runtime of RaR is still comparable to the fastest methods and better than the ones of wrapper and histogram-based methods. The runtime comparison with competitor methods is not really fair as some methods have an underlying C or C++ implementation which is much faster compared to Python. Further, the kNN classifier used in RKNN and the MI filter is not optimized and probably not as efficient as it could be. However, trends for the runtimes can still be recognized. The most obvious trend is that wrapper-based methods such as RKNN, MI and SFS are slower than filter-based and embedded methods. Further, it is interesting that the runtime of CFS increases significantly with the missing rate which makes it not suitable for these cases. We present a more detailed analysis of runtime and scalability in a dedicated experiment later in this section.

We repeat the experiment for three more configurations. The second configuration creates three high order features and the third and fourth use the same two configurations but with ten discretized features. Table 11 presents the AUC scores for the CG over missing rate.

	numeric-3-1	numeric-cluster	mixed-3-1	mixed-cluster	$\sum$
Baseline	0.25	0.24	0.31	0.28	1.08
CFS	0.45	0.27	0.29	0.22	1.23
FCBF	0.72	0.31	0.35	0.34	1.71
MI	0.27	0.24	0.54	0.44	1.49
mRmR	0.39	0.28	0.42	0.24	1.33
RKNN	0.64	0.33	0.59	0.32	1.88
Random Forest	0.74	0.39	0.23	0.23	1.60
ReliefF	0.42	0.35	0.42	0.34	1.53
SFS + Tree	0.56	0.34	0.53	0.32	1.75
XGBoost	**0.74**	0.42	0.48	0.24	1.88
DELETION + RaR	0.32	0.28	0.35	0.30	1.26
MEAN + RaR	0.70	**0.46**	0.65	0.44	2.25
KNN + RaR	0.64	0.41	0.57	0.40	2.02
SOFT + RaR	0.66	0.44	0.61	0.44	2.15
MICE + RaR	0.67	0.44	0.58	0.42	2.11
RaR-deletion	0.73	**0.46**	0.65	0.43	2.27
RaR-category	0.73	0.45	0.51	0.39	2.08
RaR-partial	0.72	0.36	0.66	0.36	2.10
RaR-alpha	**0.74**	0.44	**0.67**	**0.46**	**2.32**
RaR-proba	0.73	0.44	0.60	0.40	2.18
RaR-radius	**0.74**	0.41	0.64	0.43	2.22
RaR-distance	0.**74**	0.43	0.66	0.44	2.27
RaR-multi	0.73	0.45	0.56	0.37	2.11

Table 11: AUC scores from CG over missing rate for different synthetic datasets.

The table shows that the ranking qualities for competitor methods drop significantly when

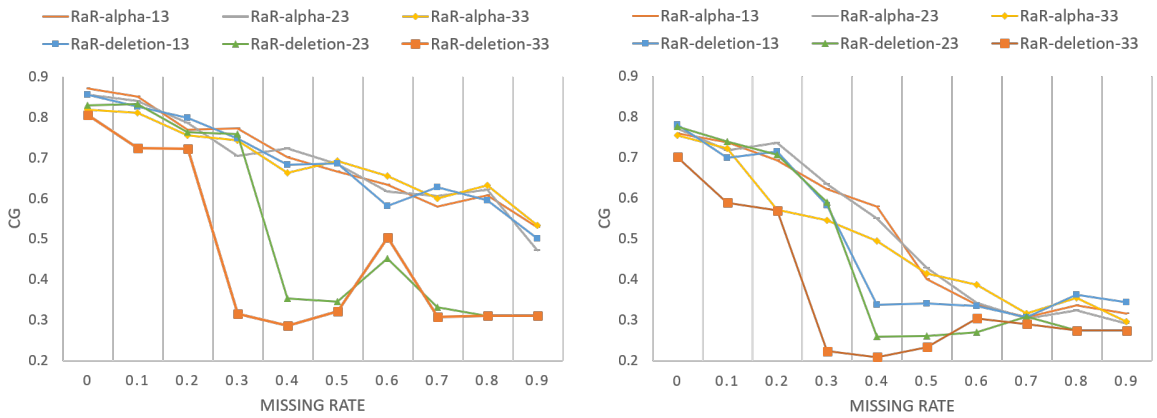
the relevance is hidden in high order features. Especially filter-based methods such as CFS, FCBF, MI, and mRmR are not able to identify these feature interactions so that their quality almost drops to the baseline. Also, RKNN and the SFS cannot identify them correctly. For RKNN we believe that this is due to the averaging of scores which underestimates interactions. For the SFS the reason lies in the sequential construction of the tree-based classifier it uses. Embedded methods achieve better qualities but are still inferior compared to RaR. The same counts for ReliefF where we see the reason for the low AUC in the inappropriate handling of missing values. Further, the qualities of competitor methods generally decrease when facing mixed data and even fall under the baseline. Especially, XGBoost, FCBF, and Random Forests create significantly worse results for the last two configurations.

In contrast, RaR achieves almost equal results on mixed data compared to the first two configurations. The weighting strategies for RaR work comparably with RaR-alpha being the best strategy in most cases. The two most exciting observations among the strategies are the low quality of the fault-tolerant strategy for the second configuration and the low quality of RaR-category for mixed data. RaR-category works the same way as RaR-deletion with the small change that it creates a new category for missing values in discrete features which is a common but here inappropriate technique. For RaR-partial we assume that the lower scores relate to worse results for small missing rates. In these cases, it splits much weight to incomplete samples and the conditionals and marginal become similar which impedes the relevance estimation. Further, RaR-proba performs worse than RaR-alpha, especially for mixed datasets. Compared to the preprocessing techniques, the weighting strategies outperform global deletion as well as kNN, soft and MICE imputation. Mean imputation interestingly achieves the best quality for the second configuration. We believe that mean imputation is better suited on our synthetic data as the values are not correlated and mean imputation thus leads to the lowest imputation error. The success of imputation-based weighting strategies is dependent on an accurate imputation. As we already noticed, imputation does not work well on our synthetic data and we thus expect imputation-based weighting strategies to perform worse in these cases. However, we note that the effect of an incorrect imputation is less severe for these approaches compared to kNN, soft or MICE imputation as global preprocessing step.

Comparison of zero-weighting and probabilistic-weighting strategy

From our initial experiment, we cannot identify substantial differences between weighting strategies. Primarily, RaR-deletion performs surprisingly well even for large missing rates. To get a deeper understanding of RaR-deletion and to identify its limitations, we conduct a dedicated experiment in which we disable our extensions of active sampling and boosting. We further change RaR by setting the minimal subspace size to one, two and three to see if RaR-deletion can still measure contrast in higher dimensional subspaces. For comparison, we add RaR-alpha to the experiment and present the results in Figure 11.

Figure 11a shows the CG over an increasing missing rate for a dataset with three relevant features, one high order feature, and ten discrete features. The instantiations of RaR-alpha perform equally well without being affected by increasing the minimum subspace size. In contrast, RaR-deletion only achieves comparable results when single features are evaluated. Without them, RaR-deletion becomes inapplicable for higher missing rates as it starts to



(a) Evaluation on 3 relevant and 1 high order feature

(b) Evaluation on 3 high order features

Figure 11: Comparison of zero-weighting and probabilistic-weighting strategy with different subspace sizes during the sampling phase evaluated on mixed data.

ignore too many samples. This observation shows that we need to assign weight to unknown values for more substantial missing rates. Whereas the lack of measuring contrast in higher dimensional subspaces is not so severe in the left figure, it becomes problematic when feature interactions are present as in Figure 11b. Here the relevance is hidden in three high order features which can only be identified by measuring contrast in multi-dimensional subspaces. In this configuration, all instantiations of RaR-deletion perform worse than the versions of RaR-alpha for higher missing rates. Thus the weight assignment to unknown values is even more crucial when we want to measure high order features.

Evaluation of active sampling

In this section, we verify if our active sampling of additional subspaces improves the deduction of single relevances in the reasoning phase. Therefore, we perform experiments on two different synthetic datasets containing mixed data. The first one includes three relevant features and one high order feature whereas the second contains three high order features. We evaluate the ranking quality of RaR-alpha using the CG over missing rate for different parameter settings. The suffixes in the column names of Table 12 describe the subspace sizes evaluated by RaR during the sampling phase. Thus *config1-13* means that configuration one is used and that RaR evaluates subspaces of dimensions one to three. The row names describe how many subspaces RaR samples in the first phase and if active sampling is used (AS suffix).

The table shows that the quality of RaR generally increases when using active sampling. This observation is not very surprising as RaR evaluates additional subspaces, 10% or at most 100, and thus explores more of the search space. However, RaR-290-AS achieves comparable and even better results than RaR-580 which evaluates almost twice as many subspaces. The differences become clearer for datasets generated based on the second configuration. Here only high order features are present and the probability that many subspaces cover the interacting features decreases so that the reasoning becomes harder. When evaluating too few subspaces, some interactions are likely to be missed. We can reduce this risk by sampling more or

	config1-13	config1-23	config2-13	config2-33
RaR-145	0.68	0.67	0.39	0.42
RaR-145-AS	0.68	0.67	0.45	0.51
RaR-290	0.70	0.69	0.47	0.46
RaR-290-AS	0.70	**0.71**	0.49	**0.52**
RaR-580	**0.71**	0.69	**0.52**	0.46

Table 12: AUC scores of CG over missing rate for different subspace sizes with and without active sampling during the sampling phase.

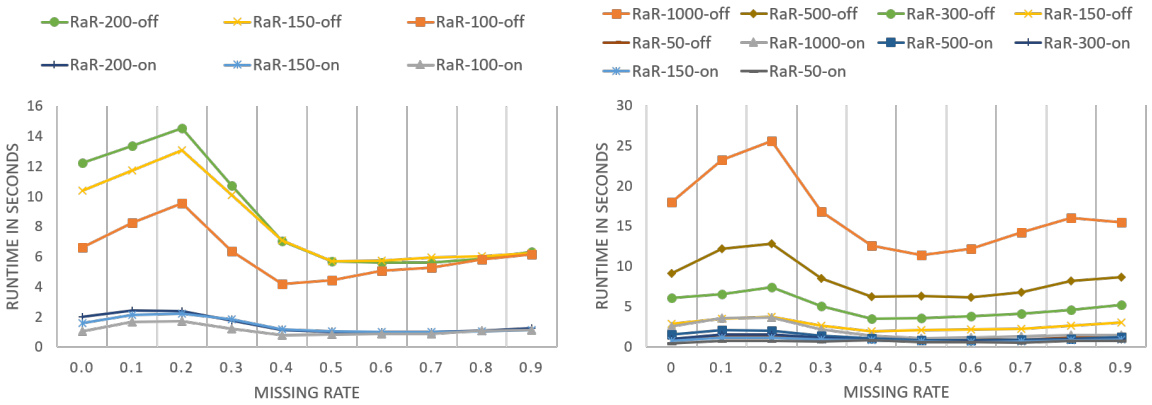
larger subspaces which is visible in rows three to five and the last column. However, this leads to inefficiency and vanishing contrast in higher dimensions, especially for high missing rates. Further, with too few sampling iterations RaR covers the high order features in too few subspaces and cannot deduce individual scores correctly so that it assigns relevance also to irrelevant features included in subspaces containing interactions. This wrong deduction is a possible reason for the low AUCs for RaR-145 in the last two columns. With active sampling, we can prevent this issue and improve the reasoning by sampling a small number of additional subspaces. By doing so, RaR-145-AS achieves significantly better scores than its counterpart RaR-145 and even better scores than RaR-580 in the last column. We note that the interactions must be covered already in the first 145 subspaces as the active sampling only explores subsets of already sampled subspaces. Finally, this experiment demonstrates that active sampling improves the reasoning by preventing relevance assignment to irrelevant features and thus allows using fewer sampling iterations without a significant loss of quality.

Evaluation of cache on runtime

An even better way of improving the performance of RaR is to cache slices. We evaluate the effect of caching on the performance by running RaR with different numbers of contrast and sampling iterations on a synthetic dataset while increasing the missing rate. Figure 12 presents the results where the suffixes *off* and *on* indicate if an approach uses slice caching and where the numbers refer to contrast and sampling iterations. As a primary instantiation, we use RaR-deletion but expect similar results for other weighting strategies.

Figure 12a shows that the runtime increases linear in the number of contrast iterations and that using the cache leads to a performance gain of approximately factor six. Also, we observe that the runtime first increases and then decreases which is due to the overhead of missing value checking in the beginning and the skipping of subspaces in the end. Interestingly, runtimes for different numbers of contrast iterations converge to the same runtime for large missing rates. We explain this by the fast slice creation in cases where only a few values exist.

Figure 12b shows a linear relationship between the number of sampling iterations and the runtime when RaR does not use a cache. In contrast, using a cache leads to significantly lower runtimes and the linear relation vanishes as RaR starts to reuse slices at some point. Thus RaR can evaluate 1000 subspaces instead of 150 within the same time when using a cache. In the other extreme, using the cache is also faster for very few sampling iterations where RaR-50-on creates more slices than it needs. We explain this by the more efficient slice creation during



(a) Contrast iterations

(b) Sampling iterations

Figure 12: Impact of caching slices on the runtime of RaR.

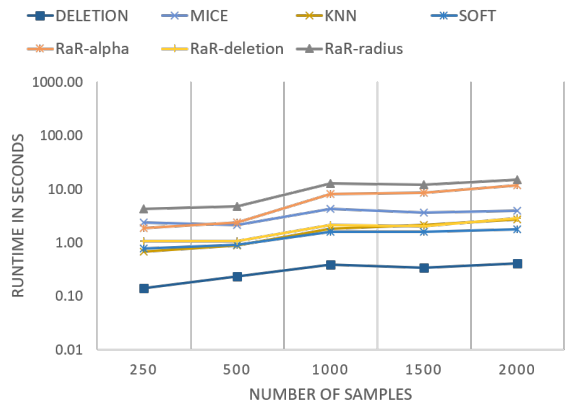
the cache initialization which reuses sorted indices for features. We do not evaluate the effect of the cache on other weighting-strategies but expect an even more significant performance gain there as they add more complexity to the slice creation compared to RaR-deletion.

Scalability analysis for increasing dataset dimensions

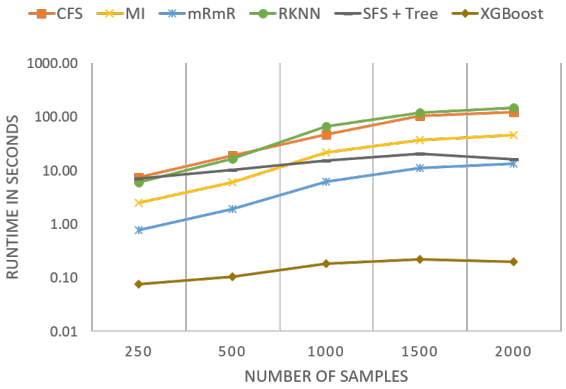
We further analyze the scalability of our weighting strategies and competitor methods with increasing dataset dimensions. Similarly to the experiments before, we create multiple datasets based on a configuration in which we insert missing values of different missing rates. This time, we change the dataset configurations throughout the runs so that we create more samples and more features respectively. Figure 13 shows the runtime of RaR and competitor methods.

The upper figures show the results for an increasing number of samples and the ones below the impact of dimensionality on the runtime. The different weighting strategies of RaR are displayed at the left and the competitor methods at the right side. We only present a subset of RaR and competitor methods to get cleaner plots by hiding approaches with similar runtimes.

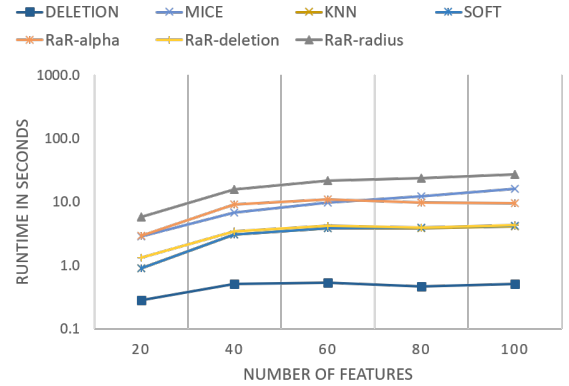
Figure 13a shows that the runtimes of RaR do not dramatically increase with more samples. Noting the logarithmic scale, the runtimes of RaR are in the range of one to approximately 15 seconds and thus much faster compared to many competitor methods displayed at Figure 13b. We further observe a linear increase in runtime for the RaR methods where RaR-alpha has the most significant increase in runtime with a factor of around six when dealing with eight times more samples. RaR-proba has a very similar runtime to RaR-alpha, and the imputation-based methods are comparable to each other with RaR-multi being the slowest strategy. Interestingly, the runtimes for RaR-deletion and RaR-partial only lead to an increase in the runtime of factor two to three and are much faster than the other strategies. We explain this by the binary slice representations in these strategies. The linear increase in runtime supports our theoretical analysis on RaR where the ranking phase described by $O(\sqrt{DIM} \cdot DIM \cdot N)$ dominates the total complexity. Among the imputation methods, only kNN imputation suffers substantially from an increasing number of samples. The scalabilities of competitor methods are clearly worse, except for XGBoost, SFS and the approaches implemented by Orange. The runtimes



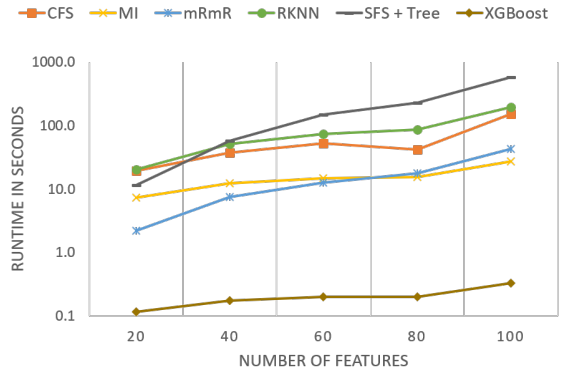
(a) Increasing number of samples (RaR)



(b) Increasing number of samples (Competitors)



(c) Increasing number of features (RaR)



(d) Increasing number of features (Competitors)

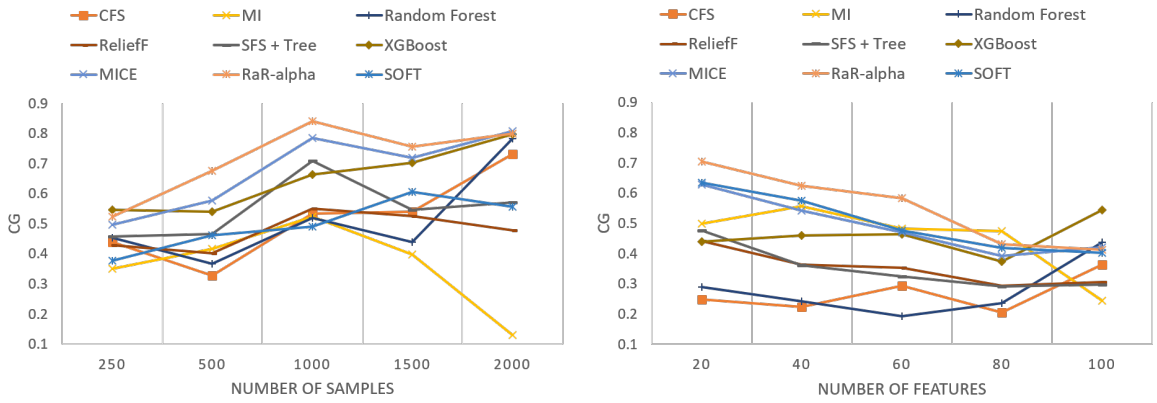
Figure 13: Runtime of different approaches with increasing dataset dimensions.

of the histogram-based approaches, CFS and mRmR, as well as MI and RKNN suffer the most from an increasing number of samples. They show a quadratic increase in runtime and become significantly slower compared to RaR. The SFS does not show such a dramatic increase in runtime due to the efficient decision tree implementation it relies on but is still slower than RaR. For the approaches XGBoost, FCBF, ReliefF, and Random Forest we cannot observe significant changes due to their fast implementations and refer to later experiments which present the runtimes of these methods again for larger datasets.

When increasing the number of features, RaR shows a sharp increase in runtime in the beginning, illustrated in Figure 13c. We explain this observation by the increasing number of sampling iterations needed to cover high order features and by the growing number of redundancy calculations in the ranking phase. However, by limiting the sampling iterations to a maximum of 1500 and by performing the iterative ranking only for the first $\sqrt{DIM}$ features, the runtime does not increase dramatically anymore from around 60 features on. Only MICE as global preprocessing technique, and weighting strategies relying on it, do not scale well with increasing dimensionality. In contrast, many competitors suffer from a quadratic complexity and become very inefficient best seen in SFS, mRmR, and CFS. Univariate filter methods such

as MI only show a linear increase as they do not consider multi-dimensional subspaces. RKNN works similar to RaR by analyzing these subspaces but samples much larger subspaces and thus only needs to add fewer subspaces for higher dimensionalities. Altogether the experiments demonstrate that RaR scales well with increasing dimensions which supports our theoretical analysis.

We further use this experiment to show trends for the ranking qualities with increasing dataset dimensions. Therefore, three relevant features and one high order feature form the basic configuration when using 20 features. When increasing the dimensionality of the dataset, we raise these numbers linearly to keep the percentage of relevant features constant among the settings. Also, we keep a fixed percentage of 50% discrete features. Figure 14 shows the average CG over different missing rates for each configuration and algorithm.



(a) Increasing number of samples

(b) Increasing number of features

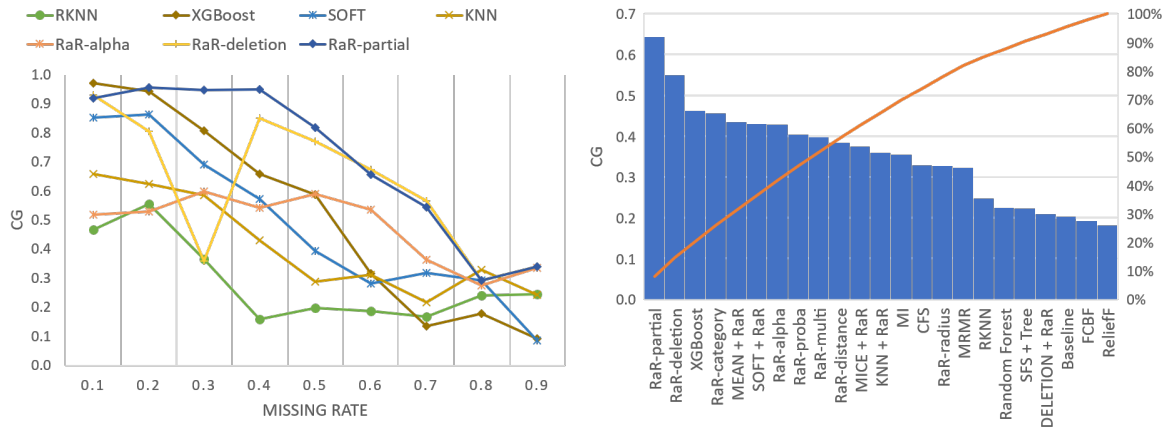
Figure 14: Mean CG over missing rates with increasing dataset dimensions.

The figures show that the qualities generally increase with more samples, except for MI, and decrease with the dimensionality. More samples intuitively lead to better scores as more samples are available to learn from which is especially helpful for more substantial missing rates. In contrast, the qualities decrease significantly with the dimensionality as feature selection becomes harder there and especially high order features are harder to identify. RaR-alpha, however, performs the best in both scenarios and we expect that we can even further improve its quality in Figure 14b by increasing the number of sampling iterations.

Evaluation of informatively missing values

We evaluate the capability of feature selection methods to identify features with informatively missing values in an experiment where we generate datasets with 20 features including four with informatively missing values and 16 with values MCAR. We simulate the predictive missingness by integrating it into the synthetic data generation process using a dummy vector as described in the experimental setup and present the results in Figure 15.

Figure 15a shows that the qualities are high for small missing rates and some approaches and then generally decrease. For small and medium missing rates, the RaR approaches work well, especially RaR-deletion and RaR-partial. RaR-alpha assigns weight uniformly to all



(a) CG over missing rate

(b) AUCs for CG over missing rate

Figure 15: Ranking qualities on synthetic data with informatively missing values.

missing values which leads to vanishing contrasts and lower scores in this experiment. Among the competitors, XGBoost stands out as the only method identifying features with predictive missingness as it learns optimal directions for unknown values. Further, the plot shows that ignoring missing values as in RKNN's distance function or imputing them in a preprocessing step are inappropriate choices here. The Pareto plot in Figure 15b presents the AUC for all competitor and weighting strategies sorted by their quality. RaR instantiations generally perform better than competitor and imputation methods, except for XGBoost. RaR-deletion ignores all incomplete samples and thus always measures contrast in features with informatively missing values. For RaR-partial we observe a similar effect as the chances for incomplete instances to have an impact on the conditionals are higher as only the observed values need to fall in a slice. Uniformly assigning weights to unknown values, as in RaR-alpha or RaR-proba, vanishes the contrast and imputation or deletion strategies do not use the missingness at all. When dealing with informatively missing values, a common path is to create new categories for them in discrete features as also stated in [16] and [23]. We see in RaR-category that this is an appropriate solution when the missingness of values depends on the class variable.

Evaluation of classification performance

In a final experiment on synthetic datasets, we evaluate the rankings based on classification scores measured by various classifiers. In particular, we are interested if feature selection still improves classification accuracies on incomplete datasets. Therefore, we generate a dataset with 20 features from which ten are discrete. The relevance is hidden in three single relevant features and one high order feature. We measure the classification performance by three times introducing missing values to a dataset according to MCAR and missing rates from 0% to 50% and by three times applying three-fold cross-validation to the dataset instantiations, leading to 27 evaluations of each algorithm. As reference classifiers, we provide a kNN classifier using the partial distance strategy, a decision tree implementation from Orange as well as a Naïve Bayes classifier and support vector machine from scikit-learn with preceding mean imputation. As the number of relevant features is known beforehand, we can use it to reduce the dimensionality

to this number. We further provide classification scores over the full set of features, the set containing only relevant features and a randomly selected set. The last two sets serve as upper and lower bounds for the classification scores. The classification performance is described by the macro-average F_1 score which is the average F_1 score over all classes of a dataset. Figure 16 shows the F_1 score of a kNN classifier over an increasing missing rate. For higher missing rates, the scores collapse to 0.5 which equals a random guessing of a class.

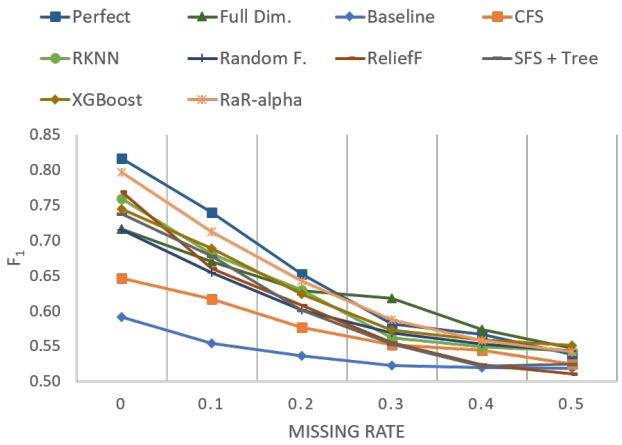


Figure 16: F_1 scores over missing rate on a synthetic data with mixed features.

The plot shows that classification accuracies drop significantly with an increasing missing rate. Further, feature selection generally improves the classification performance over the full set of features for small missing rates up to 20%, except for CFS as well as mRmR, FCBF, and MI which obtain similar scores. For more substantial missing rates, the classification performance is highest when the classifiers use all dimensions. However, even these scores are close to random guessing and thus of poor quality. The plot further shows that RaR-alpha achieves the best ratings among the feature selection methods and almost reaches the perfect quality where features are selected based on the known relevances. Other weighting strategies produce comparable results and are hidden in the plot. For ReliefF we observe the most significant performance drop when introducing missing values to the dataset which makes it an inappropriate choice for feature selection on incomplete datasets. Relying on one classifier for evaluation is, however, risky as we lack a perfect classifier and some methods optimize the selected features towards a specific classifier. We thus repeat the experiment for more classifiers and present the AUC scores for the F_1 score over missing rate in Table 13.

	Full Dim.	Baseline	Perfect	CFS	RKNN	RF	ReliefF	SFS	XGB	RaR-alpha
KNN	0.58	0.53	0.60	0.56	0.58	0.57	0.57	0.56	0.58	**0.59**
GNB	0.63	0.55	0.66	0.60	0.64	0.63	0.61	0.62	0.64	**0.65**
SVM	0.61	0.55	0.67	0.60	0.64	0.64	0.61	0.62	0.65	**0.66**
TREE	0.60	0.54	0.62	0.58	0.61	0.60	0.59	0.60	0.61	**0.62**

Table 13: AUC for F_1 scores of multiple classifiers over missing rate.

The first three columns show the classification performances over the full dimension, a random set and the perfect set containing only relevant features. As expected, the baseline achieves scores around 0.5 which expresses random guessing of classes. The classification performance over all dimensions is significantly higher but the irrelevant hinder from attaining better ratings due to overfitting. When only using the relevant features in the perfect ranking, the classifiers achieve the highest scores. RaR-alpha produces scores almost identical to the ideal scores while all other methods obtain lower ratings. Some methods such as CFS as well as mRmR, FCBF, and MI which look similar, do not improve the classification accuracy at all. The same holds for ReliefF which creates poor quality rankings when facing incomplete datasets. Embedded and wrapper-based methods can enhance classification qualities in general but are sensitive to the classifiers used whereas RaR selects more general features.

Throughout our experiments on synthetic data, we have shown that RaR creates superior rankings on mixed datasets and datasets including high order features compared to competitor methods and global preprocessing techniques such as deletion and imputation. The rankings created by RaR are robust towards an increasing missing rate and incorporate informatively missing values. The efficiency of RaR scales well with the missing rate as well as dataset dimensions. Further, the rankings created by RaR improve classification performances and are not sensitive to a particular classifier. In the next section, we examine if these observations are still valid on real-world datasets and if the integrated handling of missing values in feature selection methods is still better than a global preprocessing using imputation.

5.3 Experiments on Real-World Data

Even though synthetic data generation mimics many facets of real-world datasets, this process does not model all characteristics of datasets and it is crucial to evaluate feature selection and imputation methods on real-world datasets as well. The main difference compared to our synthetic datasets is that features in real-world datasets are more correlated/redundant to each other so that we expect imputation techniques to perform better here. From an experimental point of view, the evaluation of feature selection becomes harder as the true relevances are not known anymore and cannot be used for evaluation. Further, feature selection should eliminate redundancy which is also hard to measure. The assessment on real-world datasets contains two different metrics. In the first experiments, a similar score to the CG is introduced to quantify the robustness of feature selection methods with an increasing missing rate. Finally, we evaluate the quality of rankings based on classification performances. The difficulty for the assessment of classification performances is that the number of relevant features is not known and that the number of selected features thus becomes a new dimension of the experiments.

Evaluation of ranking stability

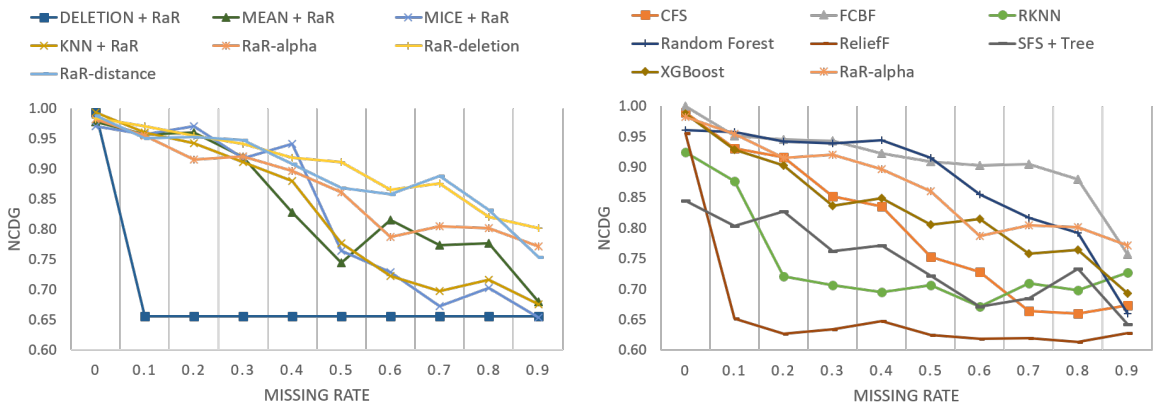
We evaluate the robustness and stability of rankings by a metric similar to the CG which is called normalized discounted cumulative gain (NDCG). [71] The advantage of the NDCG over the CG is that it is defined over all features in a ranking and we thus do not need to define a cut-off point. The NDCG sums up the relevances of selected features penalized by their positions in the ranking. Thus the NDCG is high when relevant features are ranked first. The

sum of discounted relevances (DCG) is finally normalized into the range $[0, 1]$ by dividing by the ideal DCG (IDCG). The following formulas express the scores:

$$DCG = \sum_{i=1}^{DIM} \frac{rel(i)}{log_2(i+1)} \tag{28}$$

$$NDCG = \frac{DCG}{IDCG}\,, \tag{29}$$

where $rel(i)$ represents the relevance of the feature ranked at position i. The difficulty in this setting is that $rel(i)$ is not known. We estimate these relevances independently for each feature selection method by running it multiple times on the complete data and by using the average ranking score for a feature as the true relevance. In a slightly different setting, we gain more meaningful results when we use the average relevance scores to create a final ranking and then use the feature positions to set the relevances. In an example, the feature with the highest average ranking score gets a relevance of $1/2$ and the feature with the p highest scores gets a relevance of $1/(p+1)$. In this scenario, there are clearer differences in the relevances and more importance is given to features at the beginning of a ranking. We give a counter-example to show why these adjustments make sense. The average relevance scores of three features are $0.4, 0.35, 0.45$. Even the worst ranking achieves an NCDG of 0.94 which is almost a perfect score. Figure 17 shows the NDCG over an increasing missing rate on the Ionosphere dataset.



(a) Comparison of RaR and global preprocessing

(b) Comparison with competitors

Figure 17: Stability of rankings towards the ranking on the complete dataset (Ionosphere).

The rankings for the RaR strategies and imputation techniques stay consistent for low and medium missing rates. Imputation techniques perform better on the Ionosphere dataset compared to synthetic datasets and MICE leads to more robust rankings than mean imputation which is contrary to the former observations. KNN and soft imputation also perform better than mean imputation but worse compared to MICE. However, for higher missing rates imputation suffers from larger estimation errors and direct handling of missing values in RaR leads to more robust results. Also, mean imputation performs better when many values are missing showing that other imputation techniques start to create or blur patterns in the data. Interestingly, RaR-alpha performs worse than RaR-deletion and RaR-partial as well as strategies

relying on imputation. Compared to competitors in Figure 17b, rankings produced by RaR-alpha are robust and do not change dramatically as e.g. the rankings computed by ReliefF. The observation that RaR produces more robust rankings than deterministic algorithms such as CFS is surprising as RaR is a heuristic approach which heavily relies on randomization.

We further examine the stability of rankings on additional datasets. Table 14 presents the AUC of the NDCG over an increasing missing rate for all instantiations of RaR as well as preprocessing techniques. We do not show results for competitor methods to focus on the differences between weighting strategies and the impact of imputation. Further, the results on other datasets do not provide additional insights which could not be seen on Ionosphere.

	Heart-c	Hepatitis	Ionosphere	Schizo	Semeion	Soybean	Vote	$\sum$
DEL + RaR	0.72	0.65	0.67	0.77	0.50	0.64	0.74	4.69
MEAN + RaR	**0.93**	0.85	0.84	0.91	0.83	0.82	**0.98**	6.18
KNN + RaR	0.88	0.84	0.83	0.87	0.92	0.88	0.97	6.19
SOFT + RaR	0.92	0.82	0.82	**0.96**	**0.94**	0.80	0.97	6.23
MICE + RaR	0.90	0.87	0.83	0.93	*	0.85	0.95	5.32
RaR-deletion	0.92	0.90	**0.91**	0.87	0.89	0.92	**0.98**	6.39
RaR-category	0.87	0.85	0.90	0.93	0.89	0.81	0.97	6.24
RaR-partial	0.92	**0.91**	**0.91**	0.93	**0.94**	**0.93**	**0.98**	**6.53**
RaR-alpha	0.91	**0.91**	0.87	**0.96**	0.87	0.89	**0.98**	6.39
RaR-proba	0.83	0.82	0.82	0.94	0.93	0.82	0.97	6.13
RaR-radius	0.85	0.87	0.88	0.93	0.79	0.82	0.85	5.98
RaR-distance	0.86	0.78	0.90	0.95	0.93	0.71	0.87	6.00
RaR-multi	0.82	0.82	0.87	0.94	*	0.92	0.94	5.31

Table 14: AUC scores for NDCG over missing rate for multiple UCI datasets

The table shows again that global deletion is discouraged. In contrast, imputation techniques perform well on real-world datasets and produce stable rankings comparable to many of our weighting strategies. Among the imputation techniques, we do not observe substantial differences but note that mean imputation performs worse than others, especially on Semeion. Semeion contains more features than the other datasets which allows imputation techniques to make more accurate estimations for missing values. The fact that mean imputation performs comparably among datasets of small dimensionality supports this hypothesis. On Semeion we further notice that we cannot apply MICE anymore due to lack of efficiency which is why we rely on soft imputation in RaR-radius and RaR-distance for larger datasets. The imputation-based strategies benefit from a more accurate imputation and achieve their highest scores on Semeion as well. However, they are generally less robust than global imputation and our other weighting strategies. Among them, RaR-partial shows the highest robustness followed by RaR-alpha and RaR-deletion which are both still better than global imputation. To continue our runtime and scalability analysis, we provide runtimes measured on Semeion in Table 15.

Semeion contains 256 features and 1593 samples and is thus larger than all other datasets in the table as well as all synthetic datasets during the scalability experiment. The table confirms many points of our previous discussion on the scalability and shows that the histogram-based

CFS	FCBF	MI	mRmR	RKNN	SFS	XGB
*	9	1384	356	4884	20802	28
MICE	RaR-multi	KNN	RaR-radius	RaR-alpha	RaR-deletion	RaR-proba
556	178	20	63	48	20	173

Table 15: Maximum runtimes in seconds on Semeion.

methods CFS and mRmR do not scale well with dataset sizes and that CFS does not even finish a single ranking within a day. Also, MI and RKNN get inefficient with runtimes of around 10 minutes and 1.5 hours due to the quadratic complexity of finding the nearest neighbors. SFS suffers from the quadratic complexity in the number of features and is an inappropriate choice for feature ranking on high dimensional datasets. However, we will see that the runtime of SFS is significantly lower when only a ranking of the first k features is required. Further, the runtimes of FCBF, Random Forest and ReliefF are still very efficient with FCBF being the slowest on Semeion. However, we believe that this is due to the more efficient implementation in C or C++ in the Orange Data Mining library. XGBoost, in contrast, shows a more significant increase in runtime than the methods from Orange and is slower than RaR-deletion or fast global imputation techniques such as soft or kNN imputation. In contrast, MICE does not scale well with the dimensionality and requires around ten minutes to estimate the missing values. RaR-multi is faster than MICE imputation as we reduce the number of multiple imputations for it to defeat its memory consumption when storing all estimations. Soft and kNN imputation are faster with 15 and 20 seconds and can still be applied. Among our weighting strategies, RaR-deletion and RaR-partial are the most efficient ones followed by RaR-alpha which only adds a constant offset. RaR-proba becomes clearly more inefficient than RaR-alpha and is even more inefficient than RaR-radius and RaR-distance relying on soft imputation.

We end our runtime discussion here and conclude that RaR achieves runtimes comparable to the fastest filter-based and embedded methods and shows superior efficiency compared to wrappers. Further, many of our weighting strategies scale well with dataset dimensions and increasing missing rates and thus do not hinder RaR's efficiency. In the following experiments, we mark methods with an asterisk when they do not finish a single ranking within a day.

Evaluation of active sampling and boosting

We complement our analysis of active sampling and relevance boosting in extremal regions by examining their effects on ranking stabilities and classification performances on real-world datasets. First, we provide the impact of both concepts on the ranking stability in Table 16.

The table shows that active sampling improves the robustness of rankings on datasets with more features, e.g. Ionosphere, Semeion, and Soybean. Further it achieves the superior results with the same or even lower numbers of sampling iterations. In contrast, boosting decreases the ranking robustness for many datasets. The suffix *b0.1* describes the factor for the boost of relevance hidden in extremal regions according to Equation 12. Boosting performs especially bad on Semeion and Soybean which represent multi-class problems. In these cases, the estimation of contrast measured in extremal regions is unreliable and overestimated and thus

	Heart-c	Hepatitis	Ionosphere	Schizo	Semeion	Soybean	Vote
RaR-50	0.93	0.90	0.71	0.96	0.51	0.75	0.96
RaR-50-AS	0.90	0.92	0.86	0.96	0.61	0.88	0.98
RaR-145	0.90	0.92	0.87	0.95	0.58	0.86	0.98
RaR-145-AS	0.92	0.91	0.88	0.96	0.77	0.86	0.98
RaR-290	0.94	0.91	0.87	0.96	0.65	0.88	0.98
RaR-290-AS	0.92	0.91	0.88	0.95	0.82	0.88	0.98
RaR-580	0.92	0.92	0.87	0.96	0.83	0.88	0.98
RaR-b0.0	0.94	0.91	0.89	0.97	0.93	0.92	0.98
RaR-b0.1	0.93	0.91	0.87	0.95	0.86	0.86	0.99
RaR-b0.2	0.90	0.92	0.88	0.94	0.83	0.84	0.98

Table 16: Impact of active sampling and boosting on ranking stability for OpenML datasets.

needs to be adjusted or skipped. The problem is that we aim to measure the contrast in extreme values only which leads to a small number of desired samples for the estimation of conditional distributions. However, too few samples always lead to high contrasts when many classes exist. With large contrasts in the extremal regions, the final ranking scores can be dominated by them even for small boosting factors which leads to the unreliable rankings.

However, when evaluating the impact of boosting on classification scores, the situation looks different. To measure classification performances, we follow the same setup as described on synthetic data with the difference that the parameter k of selected features is unknown and thus spans an additional dimension of the experiments. We thus perform evaluations for values of k from one to five and further evaluate multiple classifiers. To get a compact representation of the performances, we use the average F_1 score among the different values of k for each classifier, feature selection approach and missing rate. We further only display the AUC for the F_1 scores over increasing missing rates to evaluate a feature selection approach in a more compressed way. The score reduces a lot of information but still shows if a method can rank highly relevant features first. Table 17 presents the scores for different boosting factors, datasets, and classifiers. For active sampling, we cannot identify significant changes in the classification performances and do not present these results here.

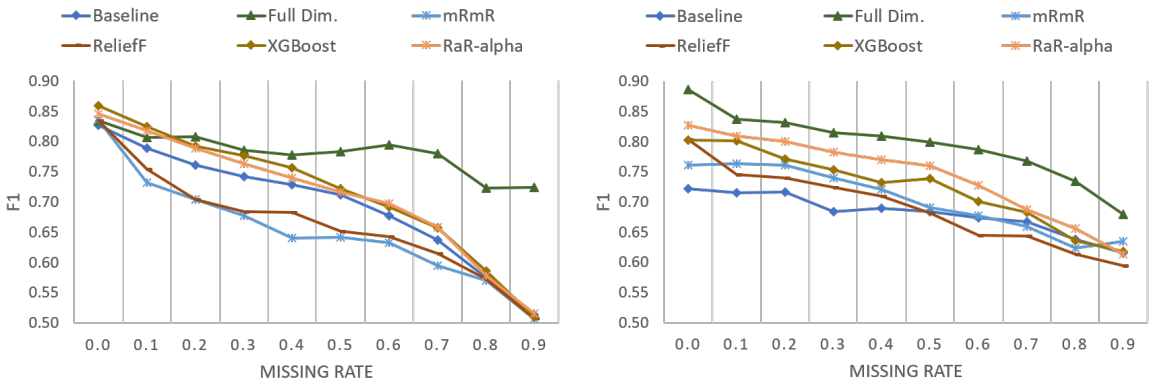
	Heart-c	Hepatitis	Ionosphere	Schizo	Soybean	Vote
RaR-b0.0 (knn)	0.66	0.78	0.71	0.59	0.26	0.83
RaR-b0.1 (knn)	0.66	0.79	0.72	0.63	0.27	0.83
RaR-b0.2 (knn)	0.65	0.80	0.72	0.64	0.28	0.83
RaR-b0.0 (gnb)	0.66	0.79	0.71	0.53	0.19	0.83
RaR-b0.1 (gnb)	0.66	0.79	0.75	0.53	0.20	0.83
RaR-b0.2 (gnb)	0.66	0.79	0.76	0.52	0.20	0.83

Table 17: AUC score for F_1 scores over missing rates for different datasets and different factors for the relevance measured in extremal regions

The table presents scores of different instantiations of the boost factor for a kNN and gaussian Naïve Bayes (gnb) classifier. We observe that boosting improves classification performances, especially for Ionosphere and the gnb classifier as well as for Schizo and the kNN classifier. The differences between the factors 0.1 and 0.2 are not very significant and, due to the negative impact of boosting on ranking robustness and multi-class problems, we suggest to keep the boosting factor as small as possible.

Evaluation of classification performance

We finally evaluate the quality of our competitor methods and RaR instantiations based on classification performances. The experimental setup is identical to the one used to assess boosting in the subsection before. The plots in Figure 18 describe the average F_1 scores of a kNN and a gnb classifier over an increasing missing rate on Ionosphere.



(a) kNN classifier

(b) gnb classifier

Figure 18: Average F_1 scores among $k \in \{1, 2, 3, 4, 5\}$ selected features over missing rate.

We show representatives for different feature selection paradigms and RaR-alpha as the only instantiation of our weighting strategies. Further, we provide results over the full set of features and for a baseline which selects features sorted by their names. However, the comparison with the F_1 score over the entire dataset is unfair to the feature selection methods as their scores are averaged among the $k \in \{1, 2, 3, 4, 5\}$ best features and it does not always hold that the predictive quality of small numbers of features is sufficient for successful classification. Nevertheless, the scores for XGBoost and RaR-alpha exceed the score over the full dataset for low missing rates. For more substantial missing rates, the performances for all methods decrease. Especially ReliefF and mRmR suffer from an increasing missing rate and start to select worse features than the baseline as in the particular example of Ionosphere the relevance lies in the first features. However, proper feature selection methods should still outperform this baseline. After manual inspection of the first features, we observe that also entirely irrelevant features are among the first five features. As the gnb classifier is more sensitive to these features, the baseline performs worse in Figure 18b. From the plots, we further note that RaR-alpha achieves the highest classification performances followed by XGBoost. We do not show instantiations of other weighting strategies as they perform very similar to RaR-alpha.

To get a more compact comparison of all methods and further datasets, we only present the AUCs for the average F_1 scores in Table 18. For high dimensional and sparse datasets, Isolet and Semeion, we collect scores at $k = \{5, 10, 15, 20, 25\}$ as we know that more features need to be selected there. The results are further collected for a gnb classifier as it is more sensitive to irrelevant and redundant features. [5] We provide results for other classifiers in our Github repository as well, but the only noticeable difference there is that wrapper-based and embedded methods optimize the selected feature subsets towards the classifiers they use.

	Heart	Hepa.	Ionos.	Schizo	Soy.	Vote	Isolet	Madelon	Musk	Semeion
Full Dim.	0.71	0.76	0.80	0.56	0.45	0.87	0.79	0.56	0.77	0.75
Baseline	0.62	0.78	0.68	0.53	0.15	0.73	0.29	0.50	0.81	0.33
CFS	0.64	0.78	0.71	0.52	0.21	0.83	*	*	*	*
FCBF	0.62	0.78	0.74	0.53	0.15	0.73	**0.40**	**0.57**	0.71	**0.43**
MI	0.64	0.77	0.70	0.53	**0.22**	0.83	*	0.50	*	0.34
mRmR	0.58	0.74	0.70	0.52	0.21	**0.84**	*	0.53	0.59	0.41
ReliefF	0.65	0.79	0.69	**0.54**	0.19	0.82	0.27	0.50	0.74	0.38
Random F.	0.65	**0.80**	**0.75**	0.52	0.18	0.83	0.38	**0.57**	**0.84**	0.41
RKNN	0.65	0.78	0.71	0.52	0.20	0.82	*	*	*	*
XGBoost	0.63	0.79	0.72	0.52	0.18	0.76	0.38	0.56	**0.84**	0.41
SFS + Tree	0.64	0.78	0.71	0.52	0.20	0.81	*	0.54	0.82	*
RaR-alpha	**0.66**	0.79	**0.75**	0.53	0.20	0.83	0.34	0.54	**0.84**	0.39

Table 18: AUC of F_1 scores for different OpenML datasets and a gnb classifier.

RaR-alpha is chosen as the only instantiation for RaR as it achieves the highest scores for most datasets, except for Isolet and Semeion where RaR-deletion reaches higher scores of 0.38 and 0.41. RaR-alpha leads to the highest scores in three datasets and produces results close to the bests in most other datasets. In the case of Hepatitis and Musk, feature selection improves the classification performance over all dimensions even though it is an unfair comparison to the account of feature selection. Besides RaR-alpha, Random Forest and FCBF stand out as methods leading to the highest scores. The other methods generally achieve lower scores and even fall below the Baseline, e.g. mRmR on Heart or Hepatitis. More detailed observations, such as the detection of high order features and the handling of mixed data, have already been discussed in the section of synthetic datasets in more controlled settings and are omitted here.

For the runtimes, we only summarize the most important observations. The asterisks show that CFS and RKNN cannot finish the experiment in one day for the largest datasets. MI has the same scalability issue as RKNN but is still applicable to create a single ranking for Semeion and Madelon in 10 and 38 minutes. These scores seem to be reasonable, but due to many repetitions, the total execution time of the experiments increases dramatically. In comparison, the fastest methods still achieve rankings in a few seconds on these datasets. Further, we note that mRmR cannot be applied on Isolet anymore and that SFS can only be applied to Musk and Madelon as only the first five features need to be ranked there. For RaR-radius and RaR-distance, we switch the imputation technique to soft imputation for large datasets which eliminates RaR-multi. We further list selected runtimes on Isolet in Table 19.

FCBF	ReliefF	Random F.	XGBoost	RaR-alpha	RaR-del.	RaR-dist.	RaR-rad.
84	20	6	654	181	83	454	577

Table 19: Runtime in seconds for ranking the top k features in one fold of the cross-validation.

We see that FCBF scales worse than RaR as it is slightly slower than RaR-deletion on Isolet. RaR-alpha needs significantly more time than RaR-deletion but is still faster than methods based on single imputation even though they rely on soft imputation. RaR-alpha is also faster than XGBoost which we relate to the high complexity of fitting a classification model on a large dataset containing many classes. This claim is also supported by the observation that XGBoost is slower on Semeion than on Madelon which is larger in both terms of samples and features. Only ReliefF and the Random Forest finish a ranking faster than RaR-deletion. However, these methods show significantly worse classification performances.

Evaluation of the impact of imputation on feature selection

So far the impact of imputation on the quality of selected features has only been analyzed on synthetic data and during the robustness analysis in the first experiment on real-world datasets. However, the application of imputation on synthetic datasets is not promising as no correlations between values and features exist there. This correlation is present in datasets from OpenML which makes them better suited to examine the impact of imputation on feature selection. Further, the stability analysis has only shown how the rankings change over the missing rate but this is not sufficient to conclude on their qualities. We thus analyze the impact of imputation on feature selection and classification in a dedicated experiment which follows the same experimental setup as the experiment before. The only difference is that we record the maximum classification performance among all values for k instead of their average. This is because imputation relies on a sufficient number of features to accurately estimate missing values. We select RaR-alpha as the base method and create six different settings for each dataset. In the first three settings, we apply RaR-alpha directly on the incomplete dataset whereas the last three settings perform an imputation step before running RaR. This allows concluding if feature selection is hindered and misled by imputation. In the first and fourth configuration, we perform classification on the reduced test set without imputation. The second and fifth setting perform an imputation on the entire test set before dimensionality reduction and the third and sixth setting apply imputation on the reduced test set. These settings allow conclusions about the impact of imputation on classification and if irrelevant features hinder imputation. The settings which perform imputation over the entire test set are however not applicable when feature selection aims to reduce measurement and storage costs as still all features need to be measured. The experiments rely on the four classifiers mentioned before and use kNN, soft and MICE imputation. We present detailed results for a kNN classifier and kNN imputation in Table 20 and more compact results for additional classifiers and imputation techniques later in this experiment.

The table shows that feature selection in most cases improves classification performances. The results between the first and last three settings are minimal with a positive tendency to the configurations which do not impute data before feature selection. More significant

	Heart	Hepa.	Ionos.	Schizo	Soy.	Vote	Isolet	Madelon	Musk	Semeion	$\sum$
Full Dim.	0.72	0.81	0.78	0.56	0.53	0.87	0.59	0.52	0.88	0.70	6.95
RaR-1	0.74	0.81	0.82	**0.68**	0.43	0.89	0.48	0.60	0.89	0.64	6.97
RaR-2	**0.76**	**0.82**	**0.83**	0.66	**0.57**	**0.91**	0.72	0.60	**0.92**	**0.75**	**7.53**
RaR-3	**0.76**	0.81	0.81	0.66	0.54	0.90	0.58	0.59	0.90	0.66	7.22
RaR-4	0.73	0.80	0.81	0.64	0.41	0.88	0.53	0.60	0.88	0.63	6.93
RaR-5	**0.76**	**0.82**	0.82	0.63	**0.57**	0.90	**0.77**	**0.61**	**0.92**	**0.75**	**7.53**
RaR-6	0.75	**0.82**	0.80	0.62	0.53	0.89	0.62	0.60	0.89	0.66	7.18

Table 20: AUC scores for the maximum F_1 scores of a kNN classifier among the best k features over an increasing missing rate. The first three versions apply feature selection on incomplete data, the others after kNN imputation. RaR 1 and 4 do not apply imputation on the test data. RaR 2 and 5 apply kNN imputation on the test data before dimensionality reduction, RaR 3 and 6 after.

differences exist for the impact of imputation on classification. Imputing the test set improves the classification, especially on datasets with many features. Isolet and Semeion benefit the most from imputing the test set which could also be related to data sparsity, e.g. Semeion consists of binary features with few ones. On these datasets, feature selection also benefits from imputation so that the last three settings perform better there. Applying imputation on the entire test set gains the highest scores which supports that imputation needs a reasonable amount of data to estimate missing values accurately. However, imputation over many features can be inefficient and all features still need to be measured preventing cost savings. Also, many real-world datasets only contain a few features so that imputation cannot show its strengths.

To examine the impact of irrelevant features, we repeat the experiment on small datasets each extended by ten features with values randomly drawn from $\mathcal{N}(0, 1)$. We do not add noisy features to large datasets as they already contain a large number of features from which we expect many to be weakly or not relevant. Table 21 displays the results.

	Heart	Hepa.	Ionos.	Schizo	Soy.	Vote	$\sum$		Table 20	
Full Dim.	0.69	0.79	0.75	0.50	0.45	0.84	4.03		4.26	
RaR-1	0.73	0.81	**0.82**	**0.65**	0.42	0.89	4.32		4.36	
RaR-2	**0.74**	**0.82**	**0.82**	**0.65**	**0.55**	**0.90**	**4.48**		**4.53**	
RaR-3	**0.74**	**0.82**	**0.82**	**0.65**	0.53	**0.90**	4.46	**13.26**	4.48	**13.37**
RaR-4	0.72	0.81	0.81	0.59	0.41	0.89	4.23		4.29	
RaR-5	0.73	0.81	**0.82**	0.59	0.53	**0.90**	4.38		4.49	
RaR-6	0.73	0.81	0.80	0.58	0.52	0.89	4.33	12.93	4.42	13.19

Table 21: Table showing the scores for the same experiment as in Table 20 but with ten additional noisy features in each dataset. The sums in the last columns are aggregated results for each approach and include the aggregated results from Table 20.

The table shows that classification using the full set of features suffers significantly when we add noisy features. Feature selection averts the risk of overfitting on irrelevant features and leads to more similar scores compared to Table 20. However, irrelevant features hinder imputation so that the scores for the last three settings decrease notably as inferior features are selected. Also, differences between settings applying imputation on the entire or reduced test set become more similar due to the negative impact of uncorrelated features on imputation. Nevertheless, the ranking of settings stays the same and imputation over the whole set of features still achieves the highest scores followed by imputation on the reduced test set.

We further provide results for additional classifiers and imputation techniques. To facilitate their comparison, we only display aggregated results for a kNN and gnb classifier and the three imputation techniques. kNN, soft, and MICE. We further provide more detailed results also including a decision tree and a support vector machine in our Github repository. Table 22 shows the aggregated results for datasets without additional noisy features.

	KNN			GNB			
	KNN	SOFT	MICE	KNN	SOFT	MICE	$\sum$
Full Dim.	4.26	4.26	4.26	4.14	4.14	4.14	25.22
RaR-1	4.36	4.36	4.36	4.20	4.20	4.20	25.69
RaR-2	**4.53**	**4.54**	**4.55**	**4.24**	**4.29**	**4.26**	**26.41**
RaR-3	4.48	4.49	4.50	4.19	4.23	4.19	26.06
RaR-4	4.29	4.31	4.26	4.16	4.16	4.14	25.33
RaR-5	4.49	4.53	4.51	4.21	4.23	4.22	26.18
RaR-6	4.42	4.46	4.43	4.14	4.13	4.11	25.69
$\sum$	30.83	**30.95**	30.87	29.28	29.38	29.28	180.58

Table 22: Aggregated results as in Table 21 without noisy features.

Feature selection before imputation consistently selects better features resulting in higher classification performances, independent of the imputation strategy or classifier. Further, imputation improves the classification accuracies of a kNN classifier in all cases and most cases for a gnb classifier. Imputation over the entire test set consistently leads to the best scores but does not allow to reduce measurement and storage costs. When these cost savings are the primary goal of feature selection, imputation on the reduced set still improves the quality of classification but with less significant improvement. Among the imputation techniques, soft imputation performs the best but without substantial differences to kNN and MICE.

To get a more comprehensive picture of the impact of uncorrelated features, we present the same aggregated results as in Table 23 but with ten additional noisy features in the datasets. The table also contains the aggregated sums from Table 22 to facilitate their comparison. Noisy features, again, clearly reduce the classification scores when all dimensions are used. When applying feature selection directly on incomplete data, the scores do not drop that much, except for setting two which applies imputation on the entire test set. This shows that irrelevant features hinder imputation and that at least all wholly uncorrelated features should be removed before imputation. Applying imputation before feature selection leads to notably

	KNN	KNN SOFT	KNN MICE	GNB KNN	GNB SOFT	GNB MICE	$\sum$	Table 22
Full Dim.	4.03	4.03	4.03	4.09	4.09	4.09	24.34	25.22
RaR-1	4.32	4.32	4.32	**4.21**	4.21	4.21	25.61	25.69
RaR-2	**4.48**	**4.51**	4.47	4.19	4.28	4.21	**26.14**	**26.41**
RaR-3	4.46	4.47	**4.48**	4.18	**4.29**	**4.22**	26.09	26.06
RaR-4	4.23	4.25	4.22	4.15	4.16	4.13	25.15	25.33
RaR-5	4.38	4.46	4.38	4.14	4.19	4.13	25.67	26.18
RaR-6	4.33	4.41	4.36	4.10	4.14	4.10	25.45	25.69
$\sum$	30.22	**30.45**	30.28	29.06	29.36	29.08	178.45	180.58
Table 22	30.83	**30.95**	30.87	29.28	29.38	29.28	180.58	-

Table 23: Aggregated results as in Table 22 but with additional noisy features.

inferior scores compared to the first three settings. These observations are independent of the classifier and imputation strategy. Among the imputation strategies, soft imputation is the least prone to irrelevant features as it constantly achieves the highest scores.

Throughout this experiment, we have used the MCAR mechanism to introduce missing values to different datasets to investigate the impact of imputation on feature selection. We conclude that a direct feature selection on incomplete datasets leads to superior sets of selected features and that irrelevant features hinder an accurate imputation. Nevertheless, imputation is successful in improving classification performances in the MCAR scenario. The effect of different missing mechanisms is the subject of the final experiment in the next subsection.

Evaluation of different missing mechanisms

We evaluate two additional missing mechanisms on real-world datasets following our descriptions from the experimental setup: MNAR and predictive missingness as special form of MAR. In both cases, the classification performances are higher compared to scores obtained on datasets with missing values following MCAR. Hereby, the scores improve equally among all feature selection methods. We present the classification performances over the full feature set for the MCAR and MNAR missing mechanisms for all four classifiers in Table 24.

	GNB	MNAR	KNN	MNAR	SVM	MNAR	TREE	MNAR
Heart-c	0.71	0.76	0.71	0.63	0.72	0.77	0.65	0.68
Hepatitis	0.76	0.76	0.81	0.78	0.81	0.83	0.78	0.77
Ionosphere	0.80	0.85	0.78	0.77	0.86	0.92	0.75	0.79
Schizo	0.56	0.57	0.55	0.54	0.56	0.57	0.62	0.64
Soybean	0.45	0.59	0.53	0.43	0.45	0.68	0.42	0.46
Vote	0.87	0.87	0.87	0.87	0.87	0.88	0.81	0.82

Table 24: AUC for F_1 scores over missing rate applied on all dimensions.

Only the kNN classifier using the partial distance strategy does not achieve better scores when facing the MNAR mechanism. This shows that ignoring missing values is inappropriate for the MNAR mechanism, and respectively the predictive missingness. We simulate the MNAR mechanism by removing the extreme values of a feature. When the classification accuracies improve, this means that the extreme values provide information about the class. This is further motivation for the need for estimating relevance in the extremal regions of each feature. Surprisingly, the different feature selection methods do not show significant differences in their qualities and all improve equally well. This is in contrast to the results in Figure 15b where the predictive missingness is evaluated on synthetic datasets. The difference is that we have simulated predictive missingness in only a few features and using a generative process on the synthetic dataset. In our current experiment, the missingness is simulated equally in all features so that they all become predictive. In a more careful experimental design, we expect stronger differences between feature selection methods after all. Anyhow, we have already shown that RaR performs better in identifying features with informatively missing values and are thus more interested in examining the impact of imputation on this missing mechanism.

Therefore, we use the same experimental setup as in the experiment before but this time we simulate informatively missing values. The results for a gnb classifier and kNN imputation are presented in Table 25. This time we use the mean F_1 score among the different values for k to account for the fact that all features are getting relevant due to the missing values. The decision on the gnb classifier also accounts for this.

	Heart-c	Hepatitis	Ionosphere	Schizo	Soybean	Vote	$\sum$	
Full Dim.	0.88	0.81	0.86	0.64	0.90	0.95	5.04	
RaR-1	**0.80**	**0.82**	0.84	**0.63**	**0.69**	**0.96**	**4.75**	
RaR-2	0.78	0.79	0.83	0.58	0.64	0.95	4.56	
RaR-3	0.78	0.79	0.83	0.57	0.58	0.92	4.47	**13.79**
RaR-4	**0.80**	0.81	**0.85**	**0.63**	0.67	**0.96**	4.73	
RaR-5	0.79	0.77	0.83	0.58	0.63	0.95	4.54	
RaR-6	0.78	0.77	0.83	0.57	0.57	0.92	4.45	13.73

Table 25: AUC scores for F_1 scores over an increasing missing rate up to 50% for a gaussian naïve bayes classifier and kNN imputation. Values are informatively missing.

The results show that imputation lowers the classification scores in the case of predictive missingness independent of an application before or after reducing the dimensionality of the test set. The decision of applying feature selection before or after imputation on the training set does not get clear out of this experiment. Nonetheless, we are convinced that only a direct feature selection is able to identify predictively missing features as imputation before feature selection erases the information of missingness. This can also be seen in the worse classification performance when the test set is completed using imputation. The similar results are simply explained by the relevance in each feature due to the missing values and more sophisticated experimental designs, similar to the one on synthetic datasets, are required.

We have shown on real-world datasets that the rankings produced by RaR are robust towards an increasing missing rate and even more stable than some of our deterministic competitors. Active sampling further improves the robustness and allows to make use of fewer sampling iterations while at the same time preventing from reasoning errors. Boosting of relevance in extremal regions can lead to less robust rankings but is however useful to improve the classification performances which is often the aim of feature selection. The classification performances for RaR are among the bests and generally improve qualities compared to the full set of features. RaR further shows good scalability and is among the most efficient methods. We have further analyzed imputation on real-world datasets as the results obtained on synthetic datasets cannot be used for a complete evaluation of imputation. Imputation as a preprocessing or weight estimation technique performs much better on OpenML datasets than on synthetic datasets but is still behind many of our other and rather simple weighting strategies. However, imputation shows its strengths in improving classification accuracies when dealing with MCAR missing values. There, imputation performs generally the better the more features are available, except for entirely irrelevant and uncorrelated features. For MNAR or informatively missing mechanisms, we observe a decrease in classification performances when using imputation as the missingness gets lost. Imputation in these cases should also not be applied before feature selection as the detection of features with informatively missing features is beneficial regarding classification performances.

6 Conclusion

With this work, we continue the development of a feature ranking framework relying on Relevance and Redundancy (RaR). We focus on the robust applicability on incomplete datasets and evaluate RaR's performance compared to well-established feature selection methods and traditional missing value handling approaches such as deletion and imputation. We use the mathematical foundations of RaR to spotlight the part which breaks on incomplete datasets and provide a solution to this issue. Therefore, we introduce a weight function enabling a robust estimation of conditional distributions which is essential for the assessment of relevance and redundancy. Using different instantiations of the weight function, we maintain the benefits of RaR also on incomplete datasets without adding much extra complexity.

RaR ranks features by first evaluating the relevance of subspaces using a contrast measure defined between marginal and conditional distributions. The sampling of subspace relevances is followed by the deduction of individual relevances and a ranking procedure which incorporates feature redundancies. We integrate the handling of missing values in the estimation of conditional distributions. Therefore, conditions for each feature in a subspace are built only on the observed values and a weight function is used to estimate the probability of missing values to fulfill these conditions. Whereas we do not necessarily need to assign weight to missing values on small missing rates, this becomes necessary for more substantial missing rates and subspaces of higher dimensionalities. We introduce two groups of weight functions: the first uniformly assigns weights to missing values of a feature whereas the second assigns weights based on imputed values. During our experiments, we show that simple methods are already sufficient and perform generally the best. We further extend RaR by an active sampling approach which helps to prevent reasoning errors in the second stage and by the estimation of relevance hidden in extremal value regions which are likely to be ignored in the general slicing approach. Finally, we develop a robust parameter setting applicable to various datasets.

To evaluate our theoretical discussions, we provide an extensive experimental evaluation containing a large set of synthetic and real-world datasets, many competitor methods as well as a wide range of missing rates and missing mechanisms. To the best of our knowledge, we are the firsts to perform such an extensive evaluation of feature selection on incomplete datasets which also includes the adjustment of many competitor methods as well as the simulation of missing values according to different missing mechanisms. The evaluations on synthetic datasets show that RaR can still identify relevant features when many values are missing. In contrast, the quality of many competitors drops significantly even for low missing rates. This especially holds for methods which do not follow a subspacing approach as in RaR, RKNN or XGBoost and Random Forest feature selection. RaR further shows superior quality on mixed data, on datasets containing features with informatively missing values and on datasets containing high order features. The detection of high order features, however, suffers the most from an increasing missing rate and the quality significantly drops for missing rates larger than 30%. Finally, a global deletion is strictly discouraged and imputation shows poor quality when no relations between features exist. On real-world datasets, imputation performs significantly better but still leads to the selection of inferior feature subsets, especially when the missingness is related to the target variable. Nonetheless, imputation has still its entitlement as it generally improves classification accuracies. Concerning classification performances, RaR is among the

best techniques. For competitor methods, we observe the tendency that approaches which do not incorporate redundancy achieve lower scores, especially on datasets where more features need to be selected. This makes the detection of redundancy an essential aspect of feature selection. Finally, RaR provides good scalability and is among the most efficient methods on our largest datasets. Thus, RaR is the only method in our experiments which can efficiently handle missing values and mixed data while at the same time identifying relevant features, including high order interactions, as well as redundancies.

Future works

Throughout our experiments, we notice certain aspects of RaR which are worthy of future research. The first aspects focus on the general idea of RaR and thus also apply to complete datasets. The second aspects are related to the handling of missing values.

The contrast measure used by RaR works well for binary classification problems. However, we note that the qualities are lower compared to competitors on multi-class problems. This can either be tackled by an adjustment of parameters or more generally by divergence functions which measure a class-wise contrast. The second idea can also reduce the risk of selecting features which only work well in the discrimination of one class. Further, the contrasts measured in subspaces of different sizes are not always comparable and monotone which hinders the reasoning of individual relevance scores. Thus, future work could also develop a new strategy to estimate conditional distributions for a feature subspace while also giving the same importance to all samples including the ones with extreme feature values. Finally, one could evaluate the applicability of RaR on regression problems by exchanging the Kullback-Leibler divergence with a different divergence measure such as the Kolmogorov-Smirnov test statistic.

During our research, we have mainly focused on the accurate estimation of relevance from incomplete datasets while giving little attention to redundancy and its importance on the final rankings. In contrast to complete datasets, we believe that redundancy among features can be beneficial when dealing with high missing rates. Imagine two highly relevant features which are redundant. When a value is missing in one of the features, chances are still there that a value exists in the second feature which then can be used. Future work could thus focus on evaluating the impact of redundancy on the final ranking score or on using observed values in redundant features for estimating the likelihood of a missing value fulfilling the slice conditions. Using observed values from redundant features can further be the starting point for the development of better weighting strategies which assign weight more focussed so that the contrast does not vanish and can be estimated more accurately.

References

[1] Asuncion, Arthur ; Newman, David: *UCI machine learning repository*. 2007

[2] Baeza-Yates, Ricardo ; Ribeiro-Neto, Berthier u. a.: *Modern information retrieval*. Bd. 463. ACM press New York, 1999

[3] Batista, GEAPA ; Monard, MC: Experimental comparison of K-nearest neighbor and mean or mode imputation methods with the internal strategies used by C4. 5 and CN2 to treat missing data. In: *University of Sao Paulo* 34 (2003)

[4] Battiti, Roberto: Using mutual information for selecting features in supervised neural net learning. In: *IEEE Transactions on neural networks* 5 (1994), Nr. 4, S. 537–550

[5] Bermejo, Pablo ; Gámez, Jose A. ; Puerta, Jose M.: A GRASP algorithm for fast hybrid (filter-wrapper) feature subset selection in high-dimensional datasets. In: *Pattern Recognition Letters* 32 (2011), Nr. 5, S. 701–711

[6] Bilogur, Aleksey: Missingno: a missing data visualization suite. In: *Journal of Open Source Software* 3 (2018), Nr. 22, S. 547

[7] Bocklisch, Tom: *Feature Selection for Mixed Data Types by Multi-Feature Correlation Analysis*. 2016. – unpublished thesis

[8] Breiman, Leo: Random forests. In: *Machine learning* 45 (2001), Nr. 1, S. 5–32

[9] Brown, Gavin ; Pocock, Adam ; Zhao, Ming-Jie ; Luján, Mikel: Conditional likelihood maximisation: a unifying framework for information theoretic feature selection. In: *Journal of machine learning research* 13 (2012), Nr. Jan, S. 27–66

[10] Buuren, S v. ; Groothuis-Oudshoorn, Karin: mice: Multivariate imputation by chained equations in R. In: *Journal of statistical software* (2010), S. 1–68

[11] Chandrashekar, Girish ; Sahin, Ferat: A survey on feature selection methods. In: *Computers & Electrical Engineering* 40 (2014), Nr. 1, S. 16–28

[12] Chen, Tianqi ; Guestrin, Carlos: Xgboost: A scalable tree boosting system. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* ACM, 2016, S. 785–794

[13] Clark, Peter ; Niblett, Tim: The CN2 induction algorithm. In: *Machine learning* 3 (1989), Nr. 4, S. 261–283

[14] Csiszár, Imre: Axiomatic characterizations of information measures. In: *Entropy* 10 (2008), Nr. 3, S. 261–273

[15] Demšar, Janez ; Curk, Tomaž ; Erjavec, Aleš ; Gorup, Črt ; Hočevar, Tomaž ; Milutinovič, Mitar ; Možina, Martin ; Polajnar, Matija ; Toplak, Marko ; Starič, Anže u. a.: Orange: data mining toolbox in Python. In: *The Journal of Machine Learning Research* 14 (2013), Nr. 1, S. 2349–2353

[16] Ding, Yufeng ; Simonoff, Jeffrey S.: An investigation of missing data methods for classification trees applied to binary response data. In: *Journal of Machine Learning Research* 11 (2010), Nr. Jan, S. 131–170

[17] Dixon, John K.: Pattern recognition with partly missing data. In: *IEEE Transactions on Systems, Man, and Cybernetics* 9 (1979), Nr. 10, S. 617–621

[18] Domingos, Pedro ; Pazzani, Michael: Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier. In: *Machine Learning*, Morgan Kaufmann, 1996, S. 105–112

[19] Doquire, Gauthier ; Verleysen, Michel: An Hybrid Approach to Feature Selection for Mixed Categorical and Continuous Data. In: *KDIR*, 2011, S. 394–401

[20] Doquire, Gauthier ; Verleysen, Michel: Feature selection with missing data using mutual information estimators. In: *Neurocomputing* 90 (2012), S. 3–11

[21] Eberhart, Russell ; Kennedy, James: A new optimizer using particle swarm theory. In: *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on* IEEE, 1995, S. 39–43

[22] Farhangfar, Alireza ; Kurgan, Lukasz ; Dy, Jennifer: Impact of imputation of missing values on classification error for discrete data. In: *Pattern Recognition* 41 (2008), Nr. 12, S. 3692–3705

[23] García-Laencina, Pedro J. ; Sancho-Gómez, José-Luis ; Figueiras-Vidal, Aníbal R: Pattern classification with missing data: a review. In: *Neural Computing and Applications* 19 (2010), Nr. 2, S. 263–282

[24] Ghasemi, Asghar ; Zahediasl, Saleh: Normality tests for statistical analysis: a guide for non-statisticians. In: *International journal of endocrinology and metabolism* 10 (2012), Nr. 2, S. 486

[25] Gunnemann, Stephan ; Muller, Emmanuel ; Raubach, Sebastian ; Seidl, Thomas: Flexible fault tolerant subspace clustering for data with missing values. In: *Data Mining (ICDM), 2011 IEEE 11th International Conference on* IEEE, 2011, S. 231–240

[26] Gupta, Amit ; Lam, Monica S.: Estimating missing values using neural networks. In: *Journal of the Operational Research Society* 47 (1996), Nr. 2, S. 229–238

[27] Guyon, I: *NIPS 2001 Workshop on variable and feature Selection.* 2001

[28] Guyon, Isabelle ; Elisseeff, André: An introduction to variable and feature selection. In: *Journal of machine learning research* 3 (2003), Nr. Mar, S. 1157–1182

[29] Guyon, Isabelle ; Elisseeff, André: An introduction to feature extraction. In: *Feature extraction.* Springer, 2006, S. 1–25

[30] Guyon, Isabelle ; Gunn, Steve ; Ben-Hur, Asa ; Dror, Gideon: Result analysis of the NIPS 2003 feature selection challenge. In: *Advances in neural information processing systems*, 2005, S. 545–552

[31] Hall, Mark A. ; Smith, Lloyd A.: Practical feature subset selection for machine learning. In: *Computer scienceâĂŹ98 proceedings of the 21st Australasian computer science conference ACSC* Bd. 98, 1998, S. 181–191

[32] Hall, Mark A.: Correlation-based feature selection for machine learning. (1999)

[33] Ho, Tin K.: Nearest neighbors in random subspaces. In: *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)* Springer, 1998, S. 640–648

[34] Huang, Samuel H.: Supervised feature selection: A tutorial. In: *Artificial Intelligence Research* 4 (2015), Nr. 2, S. 22

[35] Keller, Fabian ; Muller, Emmanuel ; Bohm, Klemens: HiCS: high contrast subspaces for density-based outlier ranking. In: *Data Engineering (ICDE), 2012 IEEE 28th International Conference on* IEEE, 2012, S. 1037–1048

[36] Kohavi, Ron ; John, George H.: Wrappers for feature subset selection. In: *Artificial intelligence* 97 (1997), Nr. 1-2, S. 273–324

[37] Kononenko, Igor: Estimating attributes: analysis and extensions of RELIEF. In: *European conference on machine learning* Springer, 1994, S. 171–182

[38] Kozachenko, LF ; Leonenko, Nikolai N.: Sample estimate of the entropy of a random vector. In: *Problemy Peredachi Informatsii* 23 (1987), Nr. 2, S. 9–16

[39] Kraskov, Alexander ; Stögbauer, Harald ; Grassberger, Peter: Estimating mutual information. In: *Physical review E* 69 (2004), Nr. 6, S. 066138

[40] Kullback, Solomon ; Leibler, Richard A.: On information and sufficiency. In: *The annals of mathematical statistics* 22 (1951), Nr. 1, S. 79–86

[41] Li, Jundong ; Cheng, Kewei ; Wang, Suhang ; Morstatter, Fred ; Robert, Trevino ; Tang, Jiliang ; Liu, Huan: Feature Selection: A Data Perspective. In: *arXiv:1601.07996* (2016)

[42] Li, Ruey-Hsia ; Belford, Geneva G.: Instability of decision tree classification algorithms. In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* ACM, 2002, S. 570–575

[43] Li, Shengqiao ; Harner, E J. ; Adjeroh, Donald A.: Random KNN feature selection-a fast and stable alternative to Random Forests. In: *BMC bioinformatics* 12 (2011), Nr. 1, S. 450

[44] Little, Roderick J. ; Rubin, Donald B.: *Statistical analysis with missing data*. Bd. 333. John Wiley & Sons, 2014

[45] Massey Jr, Frank J.: The Kolmogorov-Smirnov test for goodness of fit. In: *Journal of the American statistical Association* 46 (1951), Nr. 253, S. 68–78

[46] Mazumder, Rahul ; Hastie, Trevor ; Tibshirani, Robert: Spectral regularization algorithms for learning large incomplete matrices. In: *Journal of machine learning research* 11 (2010), Nr. Aug, S. 2287–2322

[47] Nordbotten, Svein: Neural network imputation applied to the Norwegian 1990 population census data. (1996)

[48] Parzen, Emanuel: On estimation of a probability density function and mode. In: *The annals of mathematical statistics* 33 (1962), Nr. 3, S. 1065–1076

[49] Pedregosa, Fabian ; Varoquaux, Gaël ; Gramfort, Alexandre ; Michel, Vincent ; Thirion, Bertrand ; Grisel, Olivier ; Blondel, Mathieu ; Prettenhofer, Peter ; Weiss, Ron ; Dubourg, Vincent u. a.: Scikit-learn: Machine learning in Python. In: *Journal of machine learning research* 12 (2011), Nr. Oct, S. 2825–2830

[50] Peng, Hanchuan ; Long, Fuhui ; Ding, Chris: Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. In: *IEEE Transactions on pattern analysis and machine intelligence* 27 (2005), Nr. 8, S. 1226–1238

[51] Quinlan, J. R.: Induction of decision trees. In: *Machine learning* 1 (1986), Nr. 1, S. 81–106

[52] Quinlan, J R.: Unknown attribute values in induction. In: *Proceedings of the sixth international workshop on Machine learning* Elsevier, 1989, S. 164–168

[53] Quinlan, J R.: *C4. 5: programs for machine learning*. Elsevier, 2014

[54] Raghunathan, Trivellore E. ; Lepkowski, James M. ; Van Hoewyk, John ; Solenberger, Peter: A multivariate technique for multiply imputing missing values using a sequence of regression models. In: *Survey methodology* 27 (2001), Nr. 1, S. 85–96

[55] Robnik-Šikonja, Marko ; Kononenko, Igor: Theoretical and empirical analysis of ReliefF and RReliefF. In: *Machine learning* 53 (2003), Nr. 1-2, S. 23–69

[56] Ross, Brian C.: Mutual information between discrete and continuous data sets. In: *PloS one* 9 (2014), Nr. 2, S. e87357

[57] Rubin, Donald B.: *Multiple imputation for nonresponse in surveys*. Bd. 81. John Wiley & Sons, 2004

[58] Saeys, Yvan ; Inza, Inaki ; Larranaga, Pedro: A review of feature selection techniques in bioinformatics. In: *bioinformatics* 23 (2007), Nr. 19, S. 2507–2517

[59] Schafer, Joseph L.: *Analysis of incomplete multivariate data*. CRC press, 1997

[60] Shekar, Arvind K. ; Bocklisch, Tom ; Sánchez, Patricia I. ; Straehle, Christoph N. ; Müller, Emmanuel: Including Multi-feature Interactions and Redundancy for Feature Ranking in Mixed Datasets. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* Springer, 2017, S. 239–255

[61] Steuer, Ralf ; Kurths, Jürgen ; Daub, Carsten O. ; Weise, Janko ; Selbig, Joachim: The mutual information: detecting and evaluating dependencies between variables. In: *Bioinformatics* 18 (2002), Nr. suppl_2, S. S231–S240

[62] Strobl, Carolin ; Boulesteix, Anne-Laure ; Zeileis, Achim ; Hothorn, Torsten: Bias in random forest variable importance measures: Illustrations, sources and a solution. In: *BMC bioinformatics* 8 (2007), Nr. 1, S. 25

[63] Tang, Jiliang ; Alelyani, Salem ; Liu, Huan: Feature selection for classification: A review. In: *Data Classification: Algorithms and Applications* (2014), S. 37

[64] Tang, Wenyin ; Mao, KZ: Feature selection algorithm for mixed data with both nominal and continuous features. In: *Pattern Recognition Letters* 28 (2007), Nr. 5, S. 563–571

[65] Tran, Cao T. ; Zhang, Mengjie ; Andreae, Peter ; Xue, Bing: Improving performance for classification with incomplete data using wrapper-based feature selection. In: *Evolutionary Intelligence* 9 (2016), Nr. 3, S. 81–94

[66] Tran, Cao T. ; Zhang, Mengjie ; Andreae, Peter ; Xue, Bing: A wrapper feature selection approach to classification with missing data. In: *European Conference on the Applications of Evolutionary Computation* Springer, 2016, S. 685–700

[67] Tran, Cao T. ; Zhang, Mengjie ; Andreae, Peter ; Xue, Bing: Bagging and Feature Selection for Classification with Incomplete Data. In: *European Conference on the Applications of Evolutionary Computation* Springer, 2017, S. 471–486

[68] Troyanskaya, Olga ; Cantor, Michael ; Sherlock, Gavin ; Brown, Pat ; Hastie, Trevor ; Tibshirani, Robert ; Botstein, David ; Altman, Russ B.: Missing value estimation methods for DNA microarrays. In: *Bioinformatics* 17 (2001), Nr. 6, S. 520–525

[69] Vanschoren, Joaquin ; Van Rijn, Jan N. ; Bischl, Bernd ; Torgo, Luis: OpenML: networked science in machine learning. In: *ACM SIGKDD Explorations Newsletter* 15 (2014), Nr. 2, S. 49–60

[70] Wang, Guangtao ; Song, Qinbao ; Xu, Baowen ; Zhou, Yuming: Selecting feature subset for high dimensional data via the propositional FOIL rules. In: *Pattern Recognition* 46 (2013), Nr. 1, S. 199–214

[71] Wang, Yining ; Wang, Liwei ; Li, Yuanzhi ; He, Di ; Liu, Tie-Yan: A theoretical analysis of NDCG type ranking measures. In: *Conference on Learning Theory*, 2013, S. 25–54

[72] Wilson, D R. ; Martinez, Tony R.: Improved heterogeneous distance functions. In: *Journal of artificial intelligence research* 6 (1997), S. 1–34

[73] Wold, Svante ; Esbensen, Kim ; Geladi, Paul: Principal component analysis. In: *Chemometrics and intelligent laboratory systems* 2 (1987), Nr. 1-3, S. 37–52

[74] Xiong, Momiao ; Li, Wuju ; Zhao, Jinying ; Jin, Li ; Boerwinkle, Eric: Feature (gene) selection in gene expression-based tumor classification. In: *Molecular genetics and metabolism* 73 (2001), Nr. 3, S. 239–247

[75] Yu, Lei ; Liu, Huan: Feature selection for high-dimensional data: A fast correlation-based filter solution. In: *Proceedings of the 20th international conference on machine learning (ICML-03)*, 2003, S. 856–863

[76] Zhao, Zheng ; Morstatter, Fred ; Sharma, Shashvata ; Alelyani, Salem ; Anand, Aneeth ; Liu, Huan: Advancing feature selection research. In: *ASU feature selection repository* (2010), S. 1–28

Zusammenfassung in deutscher Sprache

Data-Mining bezeichnet die Anwendung statistischer Methoden auf große Datensätze mit dem Ziel versteckte Information zu identifizieren und Modelle zu erstellen, die Werte bzw. Klassen von Datenpunkten vorhersagen können. Aufgrund gesunkener Kosten werden immer mehr Sensor-Daten gespeichert. Hierbei sind nicht immer alle Attribute, genannt Features, relevant für das finale Modell und können dessen Qualität sogar verschlechtern. Des Weiteren enthalten viele Industrie- und Forschungsdatensätze fehlende Werte, welche die Anwendung von Standard-Algorithmen verhindern. Gründe für fehlende Werte sind vielseitig, z.B. fehlerhafte Sensoren oder Daten-Transformationen. Fehlende Werte werden in den meisten Fällen in einem Vorverarbeitungs-Schritt behandelt. So werden unvollständige Datenpunkte entweder gelöscht oder durch Schätzungen komplettiert. Das Löschen von Daten führt zu großem Informationsverlust und sollte daher vermieden werden. Im Gegensatz dazu ersetzt Imputation fehlende Werte durch Schätzungen. Diese können jedoch stark von den echten Werten abweichen und so Ergebnisse und Statistiken verfälschen. Multiple Imputation modelliert Unsicherheiten bei den Schätzungen. Diese gehen jedoch meist in den folgenden Schritten wieder verloren, indem fehlende Werte durch ihre durchschnittlichen Schätzungen ersetzt werden.

Mit dieser Arbeit stellen wir eine Alternative vor, in welcher nur Wahrscheinlichkeiten abgeschätzt werden müssen, ob fehlende Werte in einen bestimmten Bereich fallen. Wir haben diese Idee in einen neuartigen Feature Ranking Algorithmus (RaR) integriert, welcher Attribute nach ihrer Relevanz und Redundanz ordnet. Diese beiden Metriken entsprechen dem Kontrast zwischen marginalen und bedingten Verteilungen in zufällig gezogenen Teilräumen. Die bedingten Verteilungen werden hierbei durch Konditionierung aller Features in einem Teilraum empirisch abgeschätzt und mit der Marginal-Verteilung abgeglichen. Wir ermöglichen die Anwendung von RaR auf unvollständigen Datensätzen, indem wir eine Gewichtungsfunktion entwickeln. Diese ist essentiell, um abzuschätzen ob fehlende Werte Bedingungen an Teilräume erfüllen. Des Weiteren haben wir RaR durch gezieltere Evaluierung von Teilräumen verbessert und präsentieren Parameter, die auf verschiedensten Datensätzen gute Ergebnisse liefern.

Um unsere Erweiterungen zu evaluieren und um RaR mit verwandten Algorithmen und Imputationstechniken zu vergleichen, haben wir umfangreiche Experimente durchgeführt. Diese umfassen mehrere reale und künstliche Datensätze sowie verschiedene Prozentsätze und Mechanismen von fehlenden Werten. Die Experimente zeigen, dass die Vorteile von RaR auch auf unvollständigen Daten sichtbar sind und dass RaR im Vergleich zu etablierten Feature Selection und Imputationstechniken relevantere Attribute identifiziert. Zudem ist RaR die einzige Methode in unseren Experimenten, die alle folgenden Bereiche abdeckt: Berücksichtigung von Redundanz und Relevanz inklusive Interaktionen zwischen Attributen, robuste Behandlung von fehlenden Werten und unterschiedlichen Attributs-Typen sowie Skalierbarkeit und Effizienz. Die Experimente haben des Weiteren gezeigt, dass die Eliminierung von irrelevanten Attributen die Qualität von Klassifizierungsmodellen und Imputationstechniken verbessert.

Disclaimer

I certify that the material contained in this dissertation is my own work and does not contain significant portions of unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation.

Hiermit versichere ich, dass diese Arbeit selbständig verfasst wurde und dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt wurden. Diese Aussage trifft auch für alle Implementierungen und Dokumentationen im Rahmen dieses Projektes zu.

Potsdam, October 22, 2018

(Sebastian Rehfeldt)