Data Min Knowl Disc (2014) 28:1503–1529
DOI 10.1007/s10618-014-0368-8

A peek into the black box: exploring classifiers by randomization

Andreas Henelius · Kai Puolamäki · Henrik Boström · Lars Asker · Panagiotis Papapetrou

Received: 14 February 2014 / Accepted: 19 June 2014 / Published online: 23 July 2014

Abstract Classifiers are often opaque and cannot easily be inspected to gain understanding of which factors are of importance. We propose an efficient iterative algorithm to find the attributes and dependencies used by any classifier when making predictions. The performance and utility of the algorithm is demonstrated on two synthetic and 26 real-world datasets, using 15 commonly used learning algorithms to generate the classifiers. The empirical investigation shows that the novel algorithm is indeed able to find groupings of interacting attributes exploited by the different classifiers. These groupings allow for finding similarities among classifiers for a single dataset as well as for determining the extent to which different classifiers exploit such interactions in general.

Keywords Classifiers · Randomization

Responsible editor: Toon Calders, Floriana Esposito, Eyke Hüllermeier, Rosa Meo.

A. Henelius (✉) · K. Puolamäki
Finnish Institute of Occupational Health, Topeliuksenkatu 41 a A, 00250 Helsinki, Finland
e-mail: andreas.henelius@ttl.fi

K. Puolamäki
e-mail: kai.puolamaki@ttl.fi

H. Boström · L. Asker · P. Papapetrou
Department of Computer and Systems Sciences, Stockholm University,
Forum 100, 164 40 Kista, Sweden
e-mail: henrik.bostrom@dsv.su.se

L. Asker
e-mail: asker@dsv.su.se

P. Papapetrou
e-mail: panagiotis@dsv.su.se

1 Introduction

For many predictive data mining tasks, there is a desire to not only find models with high predictive performance, but also to allow for end users to understand what factors are of importance for the predictions. An example would be a situation where medical researchers need a model for screening for persons with high risk of a certain disease. Such a model is of interest both for accurately identifying persons with increased risk and for understanding and verifying the underlying causes of the disease. Hence, the model should be both reliable and easy to understand. Reliability of models has been studied rather extensively, e.g., Lijffijt et al. (2014); Misra et al. (2012), while interpretability has often been neglected in exchange for accuracy. Many frequently employed high-performing learning algorithms, such as support vector machines (SVMs) (Cortes and Vapnik 1995) and random forests (Breiman 2001), do not allow for direct interpretation of the models due to their complexity, i.e., they are normally considered to be black box, or opaque, models. Depending on the requirements of a particular data mining project, an option may be to sacrifice predictive performance by only considering interpretable models, such as decision trees (Breiman et al. 1984; Quinlan 1986) and decision lists (Clark and Niblett 1989). Another possible option is to employ methods that explicitly balance accuracy and interpretability (Plate 1999; Ishibuchi and Nojima 2007; Pulkkinen and Koivisto 2008), e.g., allowing the user to specify how losses in predictive performance should be weighed against model complexity. A third option is to translate the black box model into an approximate model that allows for interpretation, as described in the extensive literature on rule extraction, see, e.g., Andrews et al. (1995); Johansson et al. (2003). It should be noted that even models that do allow for direct inspection, may in practice be very hard to interpret, e.g., the number of nodes in a decision tree or the number of conditions in a rule may be overwhelming. Furthermore, underlying assumptions of the learning algorithm, e.g., the independence assumption in the naïve Bayes classifier, may be violated, leading to the classifiers effectively utilizing correlations that cannot be inferred by observing the model parameters alone (Domingos and Pazzani 1997).

In tasks where the best-performing model is indeed a black box and where one is not willing to trade predictive performance for interpretability, the above three options are not feasible. Instead one has to rely on techniques for analyzing the models, as a complement to, rather than a replacement for, the original black box model. One such technique, originally proposed for random forests, but applicable to any type of predictive model, is based on estimating the attribute importance by randomly permuting the values for each independent attribute in an example set and measuring the relative effect of each permutation on the predictive performance (Breiman 2001). The underlying assumption is that attributes that do not contribute to the predictions will not be affected by permuting their values, and hence will obtain low importance scores, while attributes that are crucial for the outcome will get high importance scores. This technique for determining attribute importance is clearly not dependent on the type of predictive model investigated, since only the input and output of the models are considered, and in addition to being used in conjunction with random forests, it has also been used with, e.g., SVMs (Zacarias and Boström 2013).

One obvious limitation of the above procedure is that the effect of permuting values is investigated only for one attribute at a time. This means that in cases where the model exploits a set of correlated attributes to determine the predicted value, permuting the values for just one of these attributes may have little or no impact on the estimated predictive performance. In effect, this means that the calculated importance scores for such attributes may be close or identical to the scores of non-important attributes, e.g., those that are not even used by the model. Strobl et al. (2007) observed that the scores are not reliable in situations where the attributes vary in their scale of measurement or their number of categories and a remedy, specific to random forests, was proposed. Strobl et al. (2008) further observed that the attribute importance method using random forests is biased in favor of correlated attributes, and another remedy, again specific to random forests, was proposed. Janitza et al. (2013) also observed that the importance score technique is highly susceptible to class imbalance, and an alternative measure, based on the area under the ROC curve instead of accuracy, was proposed.

However, it should be noted that the scores provided by the above techniques do not contain information on potential interactions between the attributes. The latter can be of importance, e.g., when identifying groups of risk factors. For example, studies in which amino acid sequences are analyzed frequently need to consider interactions between sequence positions, see e.g., Segal et al. (2001). Another example concerns predicting the success of hip arthroplasty (Jakulin et al. 2003), where it was found that, e.g., "the presence of a neurological disease is a high risk factor only in the presence of other complications during operation".

Randomization testing has been proposed for detecting if a classifier uses interactions between attributes (Ojala and Garriga 2010). This method follows the emerging approach of separating the learning algorithm and the discovered patterns, e.g., rules or clusters, from their statistical analysis, see, e.g., Hanhijärvi et al. (2009), De Bie (2011a), De Bie (2011b), Henelius et al. (2013), Lijffijt et al. (2014). However, the method proposed by Ojala and Garriga (2010) only detects the existence of attribute interaction, and does not provide information on which attributes interact.

There is hence a need for techniques that in addition to providing information on the contribution of individual attributes in a predictive model, or indicating that some interactions among the attributes exist, can also provide insights on these interactions. In this paper, we present one such method for analyzing what interactions are important for any generic (possibly black box) classifier. The method, which can be viewed as a generalization of the methods proposed by Breiman (2001), Ojala and Garriga (2010), and Lijffijt et al. (2014), employs an efficient randomization approach to find interactions without any assumptions on the classifier or the underlying distribution of the data.

It should be noted that the task bears a close resemblance to that of finding attribute interactions when analyzing datasets in general, see e.g., Freitas (2001), Jakulin et al. (2003), Zhao and Liu (2007), Zhao and Liu (2009), Chanda et al. (2009). An important difference is, however, that the previous approaches (implicitly or explicitly) assume a specific type of underlying model for the dataset to be analyzed, while the proposed method is a tool for analyzing interactions exploited by any type of classifier. Hence, the aim for the novel method is not to discover any underlying "true" interactions, but to find out what interactions are exploited by some specific classifier of interest on a

given dataset. The choice of classifier and dataset is hence crucial for what interactions will be discovered.

In summary, the main contributions of this paper are:

- a novel problem of finding *groups* of attributes whose interactions affect the predictive performance of a given classifier is studied. We formulate this as an optimization problem, using fidelity (the fraction of matching predictions between the original dataset and a randomized dataset) as a metric for how the performance is affected;
- an approximation algorithm, `GoldenEye`, is proposed for solving the optimization problem in an iterative manner. It is shown that under reasonable assumptions, the proposed algorithm can be used to find an optimal solution;
- an empirical investigation is presented in which the proposed algorithm is used to analyze 15 different classifiers generated from two synthetic datasets with known structure and 26 datasets from the UCI Machine Learning Repository (Bache and Lichman 2014). It is shown that the algorithm can be used to analyze how a classifier exploits attribute interactions in making predictions and for comparing such interactions across classifiers.

In the next section, we start out by providing a motivating example where attribute interactions are indeed important and use this example to illustrate how such interactions may be detected using the proposed randomization method. In Sect. 3, we formalize the task using some central definitions and present the algorithm for efficiently exploring attribute interactions. We also include some theoretical guarantees for the algorithm. In Sect. 4, we investigate the use of the novel algorithm for analyzing a large number of classifiers obtained by state-of-the-art algorithms on a large set of public datasets. The results provide some novel insights both regarding properties of the generated classifiers and regarding the datasets. The contribution of the work and its relation to previous approaches are further discussed in Sect. 5. Finally, we summarize the conclusions and outline directions for future research in Sect. 6.

2 Motivating example

In this section, we provide a motivating example for the problem at hand. We study how the predictive performance of a simple classifier behaves on a toy dataset when the attributes are randomized independently and in groups.

Toy dataset Consider the dataset shown in Table 1a. The dataset consists of 16 observations (rows) and 4 binary attributes (columns): $\mathcal{A}_1$–$\mathcal{A}_4$. In the toy dataset, the class attribute $c \in \{+, -\}$ can be described by the following Boolean expression, where $\oplus$ and $\vee$ denote the logical "exclusive or" and "or" operations, respectively,

$$c = (\mathcal{A}_1 \oplus \mathcal{A}_2) \vee \mathcal{A}_3. \tag{1}$$

Hence, the class attribute is determined using the above interaction between the first three attributes, while the fourth attribute is irrelevant for classification. Furthermore,

Table 1 A binary toy dataset. The class variable (+ or −) is given by column c. The prediction of a classifier, defined by the binary relation $f(\mathcal{A}) = (\mathcal{A}_1 \oplus \mathcal{A}_2) \vee \mathcal{A}_3$, on the original data is given in column y. The prediction of the classifier on the randomized data is given in the column $y^* = f(\mathcal{A})$; non-matching predictions that cause a drop in fidelity are shown encircled in red bold font

(a) Original data

c	y	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$
+	+	1	0	1	1
+	+	1	0	1	0
+	+	0	1	1	1
+	+	0	1	1	0
+	+	0	0	1	1
+	+	0	0	1	0
+	+	1	1	1	1
+	+	1	1	1	0
+	+	1	0	0	1
+	+	1	0	0	0
+	+	0	1	0	1
+	+	0	1	0	0
−	−	1	1	0	1
−	−	1	1	0	0
−	−	0	0	0	1
−	−	0	0	0	0

(b) Randomized

y	y^*	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$
+	+	0	1	1	1
+	+	0	1	1	0
+	+	0	1	1	1
+	+	1	0	1	0
+	+	0	0	1	1
+	+	1	0	1	0
+	+	1	1	1	1
+	+	1	1	1	0
+	+	1	0	0	1
+	+	1	0	0	0
+	+	0	1	0	1
+	⊖	0	0	0	0
−	−	0	0	0	1
−	−	1	1	0	0
−	−	0	0	0	1
−	−	1	1	0	0

(c) Randomized

y	y^*	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$
+	+	1	0	1	1
+	+	0	1	1	0
+	+	1	0	1	1
+	+	0	0	1	0
+	+	1	0	1	1
+	+	0	1	1	0
+	+	1	1	1	1
+	+	1	1	1	0
+	⊖	0	0	0	1
+	⊖	0	0	0	0
+	+	0	1	0	1
+	+	1	1	0	0
−	⊕	0	1	0	1
−	−	0	0	0	0
−	−	1	1	0	1
−	⊕	1	0	0	0

(d) Randomized

y	y^*	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$
+	+	1	0	1	1
+	+	1	0	1	0
+	+	0	1	0	1
+	+	0	1	0	0
+	+	0	0	1	1
+	+	0	0	1	0
+	⊖	1	1	0	1
+	+	1	1	1	0
+	+	1	0	0	1
+	+	1	0	0	0
+	+	0	1	1	1
+	+	0	1	0	0
−	−	1	1	0	1
−	−	1	1	0	0
−	−	0	0	0	1
−	−	0	0	0	0

attributes $\mathcal{A}_1$ and $\mathcal{A}_2$ contain information about the class variable together, but not if separated, i.e., there is an attribute interaction.

Predictive performance task We would like to investigate how a given classifier behaves on this dataset. Specifically, we would like to identify which attributes or groups of attributes that are taken into account by the classifier in order to predict the class attribute. Intuitively, attributes that are independent of the class attribute should not affect the predictive performance if their values are randomly permuted. Moreover, attribute interactions such as the one shown in our example, may cause a drop in predictive performance if they are affected by the randomization.

Our classifier Assume that we have found a classifier that takes as an input the four attributes and outputs a prediction for the class. Further assume that in this simple example we have found a classifier function f that gives the perfect classification accuracy on the original data of Table 1a, defined as

$$f(\mathcal{A}) = (\mathcal{A}_1 \oplus \mathcal{A}_2) \vee \mathcal{A}_3. \tag{2}$$

Evaluation Now assume a scenario where we have no prior knowledge neither of the true class relation of Eq. (1) nor of the classifier function of Eq. (2). Further assume that we are unable to study the parameters of the classifier function f, but that we can observe the predictions given by the classifier functions on datasets of our choice.

We create datasets by randomization, and measure the performance of the classifier on the randomized datasets by *fidelity*. Fidelity is defined as the fraction of matching predictions between the original (unrandomized) dataset and the randomized dataset.

For example, the classifier does not use the attribute $\mathcal{A}_4$. Hence, if we create a randomized instance of the data by permuting the values in column $\mathcal{A}_4$ at random, the predictions given by the classifier will not change, i.e., fidelity remains at unity. We

can therefore conclude, by observing the fidelity under randomization, that attribute $\mathcal{A}_4$ is neither used nor needed by the classifier.

Instead of permuting the values in a column of the data matrix fully at random, we can permute the values in a column within the predicted class variable. Intuitively, if the attribute is independent of the other attributes given the predicted class, then such a permutation should not cause a large drop in the performance of the classifier (Ojala and Garriga 2010). For example, permuting attribute $\mathcal{A}_3$ within class will result only in a small drop in fidelity to $15/16 = 0.94$, as shown in Table 1d, while the expected drop in fidelity would be much larger if the values in the column of attribute $\mathcal{A}_3$ would be permuted fully at random. Hence, we conclude that attribute $\mathcal{A}_3$ is important and independent of the other attributes given a class.

If we permute attributes $\mathcal{A}_1$ and $\mathcal{A}_2$ independently within the predicted class, as in Table 1c, the fidelity drops to $12/16 = 0.75$. However, if we permute attributes $\mathcal{A}_1$ and $\mathcal{A}_2$ within the predicted class together, with the same random permutation—as shown in Table 1b—then the fidelity drops only a little to $15/16 = 0.94$. Hence, we can conclude that attributes $\mathcal{A}_1$ and $\mathcal{A}_2$ are both important and that they must occur together as a group.

Summarizing, using this simple toy example, we would ideally like to identify groups of attributes, such as $\mathcal{A}_1$ and $\mathcal{A}_2$ that should be grouped together, while $\mathcal{A}_3$ should be put in a different group and $\mathcal{A}_4$ should not appear in any group since it is uninformative. Next, we formalize this problem and propose an iterative algorithm to solve it.

3 Methods

3.1 Definitions

Let the dataset X be an $n \times m$ matrix of n observations and m attributes. The set of possible values for the jth attribute are denoted by $\mathcal{A}_j$ and the set of values for all attributes by $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_m$. Each row i, denoted as $X(i, \cdot)$, corresponds to an observation while each column j, denoted as $X(\cdot, j)$, corresponds to attribute $j \in [m]$, where we have used the short-hand notation $[m] = \{1, \ldots, m\}$. Finally, $X(i, j)$ denotes the value of attribute j for observation i.

Furthermore, each observation i is associated with a class label $c_i \in \mathcal{C}$, where $\mathcal{C}$ is a finite set of class labels. Hence, we define $c = (c_1, \ldots, c_n)$ to be an n-dimensional vector of class attributes, each corresponding to an observation (i.e., row) in X.

Definition 1 *Classifier and vector of predicted class attributes.* The classifier $f : \mathcal{A} \mapsto \mathcal{C}$ is a function that attempts to estimate the class label, given the attributes. Typically, the classifier function is found by a separate training process on the original (unrandomized) dataset. We define $y = (y_1, \ldots, y_n)$ to be an n-dimensional vector of predicted class labels, each obtained by applying the classifier to an observation $y_i = f(X(i, \cdot))$.

In the remainder of this paper, we always train the classifier once with unrandomized data to obtain the function $f()$. After training the classifier, the original class labels

are no longer used. Instead, we only use the predictions of the classifier function on unrandomized data and on data randomized under various constraints, as described later in this section.

Our randomization method is based on applying carefully selected permutations to the attributes (i.e., columns) of the data matrix. Let $\Pi = \{\pi : [n] \mapsto [n] \mid \pi \text{ is a bijection}\}$ be a set of all permutation functions of n elements. We use $x^* = x(\pi)$, where $\pi \in \Pi$, to denote the permutation of the n-dimensional vector x, with the elements given by $x^*_i = x_{\pi(i)}$.

Definition 2 *Within-class permutation.* Let y be a vector of predicted class labels, introduced in Definition 1. The set of within-class permutations Π_y is defined as a subset of permutations that leaves the class labels in y unchanged, i.e., $\Pi_y = \{\pi \in \Pi \mid \forall i \in [n] \colon y_i = y_{\pi(i)}\}$.

Moreover, we define a permutation of a dataset as follows.

Definition 3 *Permutation of a dataset.* Let X be a dataset and $\overline{\pi} = (\pi_1, \ldots, \pi_m)$, where $\pi_j \in \Pi$, be a vector of m permutation functions, one for each attribute. The permuted dataset $X^* = X(\overline{\pi})$ is obtained by applying a permutation in $\overline{\pi}$ for each attribute, i.e., $X^*(i, j) = X(\pi_j(i), j)$.

Finally, we can parametrize the permutations by defining groupings of attributes.

Definition 4 *Group of attributes and grouping of attributes.* Given a set of m attributes, a subset S of attributes $S \subseteq [m]$ is called a *group of attributes*. A set of non-overlapping groups of attributes $\mathcal{S}$ is then called a *grouping of attributes* and is denoted $\mathcal{S} = \{S_1, \ldots, S_k\}$, such that $S_i \subseteq [m]$ and $S_i \cap S_j = \emptyset$ for all $i \neq j$. Notice that there can be attributes $j \in [m]$ that are not in any group of attributes.

Using the above definitions, we can next define a random permutation of a dataset which can be parametrized by a grouping of attributes so that all attributes within the same group are permuted together using within-class permutation, whereas attributes not in any group are permuted using a random unconstrained permutation.

Definition 5 *Random permutation of a dataset.* Given a dataset X and a grouping of attributes $\mathcal{S}$, a random permutation of X is defined as $X^*_{\mathcal{S}} = X(\overline{\pi})$. The vector of m random permutation functions $\overline{\pi}$ is constructed so that all attributes occurring in a group are permuted together within-class and those not occurring in any group of attributes are permuted independently at random. This is described by Algorithm 1.

Next, it follows to construct a way to find the best possible grouping of attributes that best describes the classifier and the dataset. To this end, we employ *fidelity* as our evaluation metric, which measures how much the classification performance changes due to randomization. The fidelity score varies in the range $[0, 1]$, where a fidelity score of one means that the randomization does not change the behavior of the classifier at all, while a low fidelity implies that the randomization has substantially affected the performance of the classifier.

```
RandomPermutation(S)
input : grouping of attributes S
output: vector π̄ of m permutation functions
Let π̄ be an empty vector of length m
for S ∈ S do
    /* Permute a group together, within-class                    */
    Let π be a permutation drawn uniformly at random from Π_y
    for j ∈ S do
        π̄_j ← π
    end
end
for j ∈ [m] \ ∪_{S∈S} S do
    /* Attributes not in any group are permutated fully at random */
    Let π̄_j be a permutation drawn uniformly at random from Π
end
return π̄
```

Algorithm 1: Algorithm for finding random column permutations.

Definition 6 *Fidelity.* Given a dataset X, a classifier f, and a grouping of attributes $\mathcal{S}$, fidelity is defined to be the expected fraction of the predicted class labels that remain unchanged in the randomization:

$$\text{fid}(\mathcal{S}) = E\left[I\left(y_i = y_i^*\right)\right], \tag{3}$$

where $I()$ is the indicator function, $y_i = f(X(i,\cdot))$ are the predicted class labels on the unrandomized data, and $y_i^* = f(X_{\mathcal{S}}^*(i,\cdot))$ are the predicted class labels on the random permutation $X_{\mathcal{S}}^*$ of the dataset.

The fidelity can be computed to the desired accuracy by using a dataset of sufficient size. Such a dataset can be obtained as follows. Given an original dataset X of size n, a new dataset X' of size $n' > n$ is generated by sampling a sufficient number of rows from a random permutation of X. The fidelity is then estimated by computing the approximate fidelity $\widehat{\text{fid}}(\mathcal{S})$ from X'. The standard deviation of the fidelity approximation is given by the standard deviation of the binomial distribution:

$$\sigma(\widehat{\text{fid}}) = \sqrt{\frac{\text{fid}(\mathcal{S})\,(1 - \text{fid}(\mathcal{S}))}{n'}}, \tag{4}$$

where n' denotes the number of rows in the dataset. Now, $\sigma(\widehat{\text{fid}})$ is upper bounded by $\frac{1}{2\sqrt{n'}}$, i.e.,

$$\sigma(\widehat{\text{fid}}) \leq \frac{1}{2\sqrt{n'}}. \tag{5}$$

Hence, if we, for example, want to compute the fidelity to one standard deviation accuracy of 1%, it always suffices to use a sample of $n' = 2500$ rows. n' rows can be efficiently sampled, e.g., by first sampling n' rows, with replacement if $n' > n$, from the original data and then applying the random permutation given by Algorithm 1 to the resulting dataset of n' rows.

In the simplest cases, such as in the toy example of Sect. 2, the fidelity can even be computed analytically.

We further define a partial order relation $\prec_{\mathcal{S}}$ as follows.

Definition 7 *Order of grouping.* Consider three groupings of attributes, $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{S}$. Let A_{S} and B_{S} be subsets of the grouping $\mathcal{S}$ containing all groups of attributes that are fully contained in $\mathcal{A}$ and $\mathcal{B}$, respectively, i.e., $A_{\mathrm{S}} = \{X \in \mathcal{S} \mid \exists Y \in \mathcal{A}\colon X \subseteq Y\}$ and $B_{\mathrm{S}} = \{X \in \mathcal{S} \mid \exists Y \in \mathcal{B}\colon X \subseteq Y\}$. The partial order relation $\mathcal{A} \prec_{\mathcal{S}} \mathcal{B}$ is defined to be true if and only if $A_{\mathrm{S}} \subset B_{\mathrm{S}}$.

We use the partial order to define the *optimal grouping of attributes* $\mathcal{S}_{\mathrm{opt}}$.

Definition 8 *Optimal grouping of attributes* $\mathcal{S}_{\mathrm{opt}}$. We define a grouping of attributes $\mathcal{S}_{\mathrm{opt}}$ to be optimal if the fidelity is monotone with respect to the order relation $\prec_{\mathcal{S}_{\mathrm{opt}}}$, i.e., all groupings of attributes $\mathcal{A}$ and $\mathcal{B}$ that satisfy the order relation $\mathcal{A} \prec_{\mathcal{S}_{\mathrm{opt}}} \mathcal{B}$, also satisfy $\mathrm{fid}(\mathcal{A}) \leq \mathrm{fid}(\mathcal{B})$.

The monotonicity of fidelity is a natural property of a grouping of attributes, essentially claiming that fidelity cannot increase if we break one of the groups of attributes in the optimal solution, and that the attributes not belonging to any group have little impact on the performance of the classifier. The order of groupings, the optimal grouping of attributes, and the monotonicity relation are illustrated in Fig. 1 by using the toy example of Sect. 2.

Finally we can formulate the main problem studied in this paper.

Problem 1 *Optimal k-grouping of attributes.* Given a dataset X with m attributes, a classifier f, and a constant $k \in [m]$, find a grouping of all attributes $\mathcal{S}$, i.e., $\cup_{S \in \mathcal{S}} S = [m]$, of size $|\mathcal{S}| = k$ such that the fidelity $\mathrm{fid}(\mathcal{S})$ is maximized.

The above problem can also be seen as a *clustering problem*, where the task is to identify clusters under an optimization function, which here is given by fidelity. The difference between our problem and common clustering problems is that in the latter we are typically given a distance or similarity measure, whereas here we have to find the solution just by observing the costs of different groupings of attributes.

In order to solve Problem 1, we could build an exhaustive algorithm that computes all possible groupings of attributes $\mathcal{S}$. In fact, in the worst case and not making any prior assumptions of the classifier, the exhaustive search may be the only solution. This, however, would be computationally prohibitive. Hence, we propose an algorithm that can find an approximate solution under some reasonable assumptions about the dataset and the classifier.

If there exists an optimal k-grouping of attributes $\mathcal{S}_{\mathrm{opt}}$ such that the monotonicity property of Definition 8 holds, then $\mathcal{S}_{\mathrm{opt}}$ is a solution to Problem 1 as well.

Theorem 1 *If there exists an $\mathcal{S}_{opt}$ of size k such that the monotonicity property of Definition 8 holds and if the groups in $\mathcal{S}_{opt}$ contain all attributes, then $\mathcal{S}_{opt}$ is a solution to Problem 1.*

Proof The proof follows directly from the fact that any other solution $\mathcal{S} \neq \mathcal{S}_{\mathrm{opt}}$ of size k, $|\mathcal{S}| = k$, satisfies $\mathcal{S} \prec_{\mathcal{S}_{\mathrm{opt}}} \mathcal{S}_{\mathrm{opt}}$ and therefore $\mathrm{fid}(\mathcal{S}) \leq \mathrm{fid}(\mathcal{S}_{\mathrm{opt}})$. □

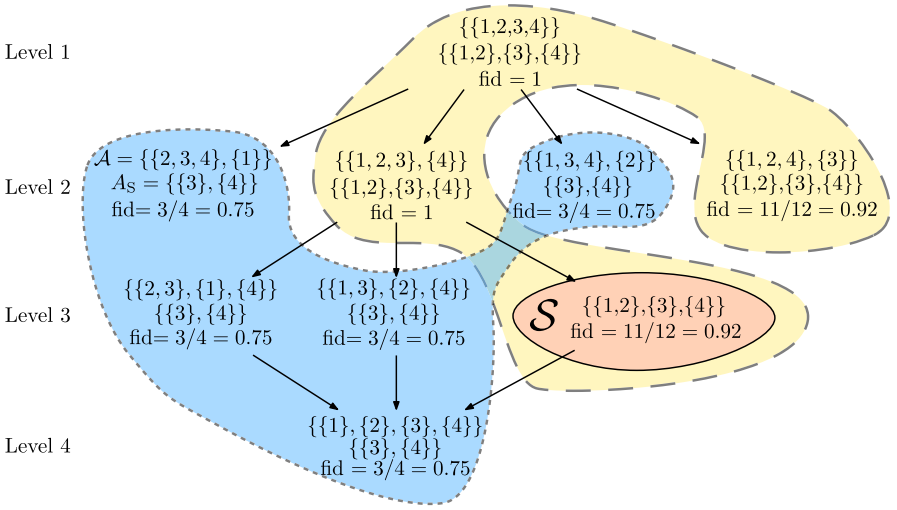

Fig. 1 A selection of groupings for the toy data presented in Sect. 2. For brevity we refer to the attributes by their indices, i.e., 1–4. The groupings are presented as a graph, where the nodes correspond to the groupings of attributes 1–4 and edges to the splits of the groupings. In this example, the grouping $\mathcal{S}$ of Definition 7 is given by $\mathcal{S} = \{\{1, 2\}, \{3\}, \{4\}\}$ (node in ellipse at level 3, red area with solid boundary). At the other nodes we show the grouping ($\mathcal{A}$, upper line at each node) and the subset of $\mathcal{S}$ that is fully contained in $\mathcal{A}$ (A_S, middle line). The fidelity, discussed in Sect. 2 and described in Definition 6, is shown in the bottom line of each node. The groupings satisfy $\mathcal{A} \prec_{\mathcal{S}} \mathcal{B}$ if and only if $\mathcal{A}$ is a grouping in the blue area (short-dashed boundary) and $\mathcal{B}$ is a grouping in the yellow area (long-dashed boundary). Notice that in this example $\mathcal{A} \prec_{\mathcal{S}} \mathcal{B}$ if and only if the fidelity of $\mathcal{A}$ is less than the fidelity of $\mathcal{B}$. Therefore, by Definition 8, $\mathcal{S}$ is an *optimal grouping of attributes* $\mathcal{S}_{\text{opt}}$

Theorem 1 links Problem 1 and the problem of finding an optimal solution under the monotonicity condition. Hence, any algorithm that is able to find the optimal solution $\mathcal{S}_{\text{opt}}$ of Definition 8 will find the solution to Problem 1 as well, if k is chosen suitably and if the monotonicity property holds.

In addition to Problem 1, we can also formulate a secondary problem.

Problem 2 *Optimal pruning of singleton attributes.* Given an optimal grouping of attributes $\mathcal{S}$ from Problem 1, remove the largest number of *singletons* $S \in \mathcal{S}$, $|S| = 1$, from $\mathcal{S}$ so that the fidelity $\text{fid}(\mathcal{S})$ is reduced by no more than δ after removing the singletons.

By solving Problem 2 after solving Problem 1, we are able to remove those singleton attributes from the optimal grouping $\mathcal{S}$ that are redundant and can be permuted using a random unconstrained permutation without significantly affecting classifier performance.

Theorem 1 still holds approximately if there are attributes that are not contained in any of the groups of the optimal solution $\mathcal{S}_{\text{opt}}$. The pruning of attributes may induce an error to the cost function which is, however, bounded by the cutoff parameter δ.

3.2 GoldenEye: an approximation algorithm

We propose a greedy algorithm, called `GoldenEye`, for solving Problems 1 and 2 in an iterative manner, where at each iteration we add the cluster that maximizes fidelity. Using this approach, we are able to solve the grouping problem in quadratic time with respect to the number of attributes in X.

We show later in Sect. 3.2.3 that if the monotonicity property of Definition 8 holds then it follows from Theorem 1 that $\mathcal{S}_{\text{opt}}$ is approximately the optimal solution of the algorithm, and that the greedy algorithm can find the solution.

3.2.1 An iterative approach

The monotonicity property implies a natural divisive approach to finding groupings. Hence, the proposed algorithm employs a top-down iterative approach for solving Problem 1. The algorithm identifies at each iteration one group that is contained in the optimal solution. Specifically, the algorithm first starts from a grouping containing all attributes. This grouping is gradually broken down, one attribute at a time, until we find a group, the breaking of which would drop the fidelity to the baseline level. We then iterate using the remaining attributes to detect the next group, until all groups have been found. This process is described in Algorithm 2.

Finally, we can proceed to also solve Problem 2, in which we test which of the remaining singleton groups can be dropped from the solution. The latter is performed as shown in Algorithm 3. The complete description of `GoldenEye` is given by Algorithm 4. The algorithm has been published as an R package.[1]

We should note that algorithms 2–4 contain only one free parameter, the sensitivity parameter δ, which is employed in the comparison of fidelity between different groupings, and is used in the calculation of the grouping threshold Δ. The value of δ regulates the granularity of the grouping of attributes and can be thought of as the *diameter* for the grouping. The effect of varying δ is shown in Table 8. The algorithm does not contain the complexity parameter k in Problem 1, instead, the sizes of the groupings can be modified by adjusting the parameter δ; see Sect. 4.2.4 for an example.

3.2.2 Example of grouping of attributes

Consider the dataset presented in Sect. 2, having four attributes $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$, and $\mathcal{A}_4$. For brevity we refer to these attributes by their indices, i.e., 1–4. The optimal grouping of attributes in this dataset can be found using `GoldenEye` as described below. Line numbers refer to Algorithm 2 and Level numbers to Fig. 1, which we use to visualize the iterative grouping process. Intuitively, we can consider a lattice (Fig. 1) spanning all possible groupings of a set of attributes. Each level k contains all groupings that contain k groups. For example, Level 1 contains only a single grouping, where all

[1] The algorithm can be downloaded from https://bitbucket.org/aheneliu/goldeneye/ (accessed 7 July 2014) or easily installed using the command `install_bitbucket` from the `devtools` package (Wickham and Chang 2014) as follows: `install_bitbucket(repo = "goldeneye", username = "aheneliu")`.

```
1  Grouping(δ, m, fid())
   input  : Sensitivity parameter δ > 0, the number of attributes m, and a fidelity function fid()
   output: Grouping of attributes S
2  S ← ∅                          /* S contains the accumulated attribute groups */
3  R ← [m]                        /* R contains a group to test for */
4  A ← ∅                          /* A contains the removed attributes */
5  Δ ← fid(L([m])) + δ            /* Declare a grouping if fidelity drops below Δ */
6  while R ≠ ∅ or A ≠ ∅ do
7      if A = ∅ and fid({R} ∪ F(S)) < Δ then
           /* If we are already below Δ before removing any attribute,
              assign the remaining attributes to singleton groups */
8          S ← S ∪ L(R)
9          R ← ∅
10         A ← ∅
11     else
           /* Find an attribute j whose removal from R decreases the
              fidelity least */
12         j ← argmax_{j∈R} fid({{R \ {j}} ∪ {{j}} ∪ {A} ∪ F(S))
13         if |R| = 1 or fid({{R \ {j}} ∪ {{j}} ∪ {A} ∪ F(S)) < Δ then
               /* If the fidelity drops below Δ add the group of attributes
                  to the results and look for the next group of attributes
                  */
14             S ← S ∪ {R}
15             R ← A
16             A ← ∅
17         else
               /* If the fidelity stays above Δ continue removing the
                  grouping R */
18             R ← R \ {j}
19             A ← A ∪ {j}
20         end
21     end
22 end
23 return S
```

Algorithm 2: The iterative algorithm to find the optimal grouping of attributes. We have used auxiliary functions $L(X) = \cup_{i \in X}\{\{i\}\}$ and $F(X) = L(\cup_{Y \in X} Y)$ which produce sets of singletons. E.g., $L(\{1, 2, 3\}) = F(\{\{1, 2\}, \{3\}\}) = \{\{1\}, \{2\}, \{3\}\}$.

attributes belong to the same group, while Level 4 contains a grouping where each attribute defines its own group.

The grouping process using Algorithm 2 starts with a single group containing the set of attributes $\{\{1, 2, 3, 4\}\}$ (Level 1 in Fig. 1).

In Line 5, the baseline fidelity is calculated using, e.g., the full partitioning $\text{fid}(L[m]) = 0.75$, where $L[m]$ corresponds to a grouping with only singletons: $\{\{1\}, \{2\}, \{3\}, \{4\}\}$. The parameter δ can in this toy example be set to, e.g., 0.05, corresponding to a 5% change in absolute fidelity and resulting in $\Delta = 0.8$. Line 7 verifies that the fidelity of the original set of attributes is above the baseline fidelity. If not, no iteration is needed and each attribute can be grouped into singletons (lines 8–10).

```
1  PruneSingletons(δ, S, fid())
   input  : Sensitivity parameter δ > 0, the solution to Problem 1 without singletons pruned, and a
            fidelity function fid()
   output: Grouping of attributes S with singletons pruned
2  R ← {X ∈ S | |X| = 1}                                          /* Singletons */
3  Δ ← fid(S) − δ                                  /* Allowed drop in fidelity */
4  while R ≠ ∅ do
       /* Find singleton X pruning of which reduces the fidelity least
          */
5      X ← argmax_{X∈R} fid(S \ {X})
6      if fid(S \ {X}) ≥ Δ then
           /* Prune the singleton and continue                          */
7          S ← S \ {X}
8          R ← R \ {X}
9      else
           /* Pruning singleton X would reduce the fidelity too much    */
10         R ← ∅
11     end
12 end
13 return S
```

Algorithm 3: Algorithm to prune singletons.

```
1  GoldenEye(δ, m, fid())
   input  : Sensitivity parameter δ > 0, the number of attributes m, and a fidelity function fid()
   output: Grouping of attributes S with singletons pruned
2  S ←Grouping(δ, m, fid())                          /* Problem 1: Algorithm 2 */
3  S ←PruneSingletons(δ, S, fid())                   /* Problem 2: Algorithm 3 */
4  return S
```

Algorithm 4: `GoldenEye`: an iterative algorithm for solving Problems 1 and 2.

Next, in line 12, one attribute at a time is detached into a singleton group, leading to the four groupings on Level 2 in Fig. 1. Out of these singletons, the algorithm will identify the attribute that affects fidelity the least. This attribute will be pruned (line 18) and added to the set of pruned attributes A. This process is repeated until all attributes have been pruned or the fidelity drops below the threshold Δ (line 13).

In this example, attribute 4 can be detached, and hence the optimal grouping at Level 2 is $\{\{1, 2, 3\}, \{4\}\}$ (depicted by the second node from the left in Level 2 in Fig. 1). In the next iteration, attribute 3 is also detached without the fidelity dropping below the threshold, and the optimal partitioning at Level 3 is given by $\{\{1, 2\}, \{3\}, \{4\}\}$ (rightmost node on Level 3). However, the third iteration (going from Level 3 to Level 4) breaks the only remaining non-singleton group $\{1, 2\}$, and hence the fidelity drops below Δ. Hence, we have identified the first group, which is now inserted into the solution, i.e., $\mathcal{S} = \{\{1, 2\}\}$.

The process continues to the next iteration where we search for the next group to be formed by the remaining set of items, $R = \{3, 4\}$. As before the algorithm identifies the singleton group $\{3\}$ (lines 14–15) after which the condition of line 7 is satisfied, and hence the algorithm terminates after assigning the remaining attributes

to singleton groups, i.e., {4} is inserted into the solution. Therefore, the final grouping is $\mathcal{S} = \{\{1, 2\}, \{3\}, \{4\}\}$.

Finally, Algorithm 3 is applied to the extracted grouping $\{\{1, 2\}, \{3\}, \{4\}\}$ to prune those singletons in the grouping that do not affect the fidelity. Specifically, the algorithm determines that dropping the singleton {4} does not affect the fidelity and hence it is pruned (lines 6–8). At the next iteration, we find that the pruning of {3} reduces fidelity and hence it is retained.

The final grouping determined by `GoldenEye` is $\{\{1, 2\}, \{3\}\}$.

3.2.3 Properties of the solution

In this section we review some properties of `GoldenEye`. The following theorem shows that the iterative algorithm can find an optimal solution, if the monotonicity property holds.

Theorem 2 *If there exists an $\mathcal{S}_{opt}$ such that the monotonicity property of Definition 8 holds, and if on previous iterations of Algorithm 2 the algorithm has found groups in $\mathcal{S}_{opt}$, then one of the groups the iteration considers is in $\mathcal{S}_{opt}$.*

Proof Assume that at line 6 of Algorithm 2 the set $\mathcal{S}$ contains only groups that are in $\mathcal{S}_{\text{opt}}$, i.e., $\mathcal{S} \subseteq \mathcal{S}_{\text{opt}}$, and that we are starting to look for a new grouping, i.e., $A = \emptyset$. The algorithm will then proceed to consider a chain of candidate groups in R by moving attributes from R to A one at a time, until the chain is terminated by the condition of line 13. If $\mathcal{S}$ already contains all non-singleton groups in $\mathcal{S}_{\text{opt}}$, then the theorem is trivially true, because the chain of candidate groups will always terminate at one of the singleton groups in $\mathcal{S}_{\text{opt}}$, unless the iteration is terminated earlier at line 13. It is therefore sufficient to consider the case where $\mathcal{S}_{\text{opt}}$ contains at least one non-singleton group that is not in $\mathcal{S}$. Consider a chain of candidate groups in R, described above, at an iteration where there is only one non-singleton group from the optimal grouping left in R, and also assume that R still contains at least one attribute not in this non-singleton group. Now the algorithm has two choices: either remove an attribute from R that is in the non-singleton group resulting in a value of R denoted here by R_1, or remove an attribute from R that is not in the non-singleton group resulting in a value of R denoted here by R_2. Because a solution resulting from the first choice $\{R_1\} \cup \ldots$ has fewer groups from $\mathcal{S}_{\text{opt}}$ than $\{R_2\} \cup \ldots$ (see line 12), the groupings satisfy the order relation $\{R_1\} \cup \ldots \prec_{\mathcal{S}_{\text{opt}}} \{R_2\} \cup \ldots$. It follows from the monotonicity assumption that $\text{fid}(\{R_1\} \cup \ldots) \leq \text{fid}(\{R_2\} \cup \ldots)$. At line 12, Algorithm 2 therefore always chooses to remove from R an attribute that does not break the only remaining non-singleton group in $\mathcal{S}_{\text{opt}}$. Therefore, the potential chain of candidate groups (values of variable R) will always include a group in $\mathcal{S}_{\text{opt}}$. □

The algorithm may find a non-optimal solution, if the iteration is cut off prematurely or too late; the cut-off heuristic is parametrized by the parameter δ. It follows from Theorem 1 that under the monotonicity property and with a suitably chosen k, Algorithm 2 can find a solution to Problem 1 as well.

Table 2 The classifiers used in the experiments and their default settings

Classifier	Description	Notes
AdaBoostM1	Adaptive boosting	10 iterations
Bagging	Bagging using REPTree	10 iterations
DecisionStump	One-level decision tree	
IBk	One-nearest neighbour	
J48	C4.5 decision tree	Minimum of 2 instances per leaf
JRip	Propositional rule learner	3 folds, 2 optimization runs
LMT	Logistic model tree	Minimum of 15 instances per leaf
Logistic	Logistic regression	Ridge estimator
LogitBoost	Boosting	No cross-validation, 10 iterations
OneR	One-rule learner	
PART	Partial decision tree	Minimum of 2 instances per leaf
SMO	Support vector machine	Polynomial kernel of degree 1
SMO radial	Support vector machine	Radial kernel of degree 3
NaiveBayes	Naïve Bayes	
RandomForest	Random Forest	500 trees

4 Experiments

4.1 Experimental setup

4.1.1 Classifiers

To obtain empirical evidence on the performance of the `GoldenEye` algorithm, a number of experiments were performed using 15 different classifiers in R (R Core Team 2014). Of the classifiers, 14 were from the Weka data mining software (Hall et al. 2009), used via RWeka (Hornik et al. 2009). The Random Forest classifier was from the randomForest package (Liaw and Wiener 2002). All classifiers, presented in Table 2, were trained using their default settings, as the goal was not to achieve the best classification accuracy, but to gain insights into how the classifiers are using the data.

4.1.2 Experimental procedure

Grouping of attributes In order to determine the grouping of attributes in the synthetic and real datasets, the following procedure was used. The datasets were randomly split into two equal parts for training and testing, stratified by class, so that half of the examples in each class were placed into the training and testing sets. If we resample the test set, the standard deviation of fidelity is given by Eq. 4, which is upper bounded by $1/(2\sqrt{N})$, where N is the size of the test set (Eq. 5). We chose the parameter δ of the `GoldenEye` algorithm to be $\delta = 2 \max \sigma_{\text{fid}} = 1/\sqrt{N}$ for the results to be robust with respect to the resampling of the dataset.

The experimental results shown in this study were calculated using the R package `goldeneye` described in Sect. 3.2.1, using a random seed of 42.

Cross-validation A cross-validation scheme was used to examine the stability of the groupings. The datasets were split into four folds, again stratified by class. The first fold was used to train the classifier and the following three folds were used as test sets to determine the grouping of the attributes.

Importance of Attributes After grouping, the relative importance of the final grouping was determined using the scheme proposed by Breiman (2001), but here using the drop in fidelity instead of misclassification rate as the measure of attribute importance. By this method, the importance of attribute j in a dataset with m attributes is determined by first calculating the fidelities $\mathrm{F}_j = \mathrm{fid}\,(\{[m] \setminus \{j\}\})$. The rank order of the fidelities F_j gives the attribute importance scores, and the attribute with the largest drop in fidelity is considered the most important.

4.1.3 Synthetic datasets

Two synthetic datasets with a known structure were created to investigate how different classifiers are able to utilize the structure of the data in the classification.

The `Single-XOR` dataset, discussed in Sect. 2 and shown in Table 1, consists of four binary attributes. The binary class c is given by $c = (\mathrm{a1} \oplus \mathrm{a2}) \vee \mathrm{a3}$. Attribute a4 is random. The correct grouping of the `Single-XOR` dataset is hence given by $\{\{1, 2\}, \{3\}\}$.

The `Double-XOR` dataset consists of six binary attributes. The class c is given by $c = (\mathrm{a1} \oplus \mathrm{a2}) \vee (\mathrm{a3} \oplus \mathrm{a4})$. Attributes a5 and a6 are random. The correct grouping of the `Double-XOR` dataset is hence $\{\{1, 2\}, \{3, 4\}\}$.

Each synthetic dataset contains 2000 instances. Noise was added to both artificial datasets by changing the class label for 10% of randomly chosen instances.

4.1.4 UCI datasets

The classifiers were trained on a set of 26 datasets chosen from the UCI Machine Learning Repository (Bache and Lichman 2014). The number of attributes in the datasets ranged from 4 to 60. Rows with missing values were omitted from the datasets. The number of instance, attributes, classes and the major class proportion for each dataset are given in Table 4.

4.2 Experimental results

4.2.1 Synthetic datasets

The results from the grouping of the synthetic datasets are shown in Table 3, together with the classification accuracy. The results show that several classifiers, such as, e.g., Bagging and Random Forest, utilize the correct underlying structure of the

Table 3 Groupings of the synthetic `Single-XOR` and `Double-XOR` datasets

	`Single-XOR`					`Double-XOR`						
Classifier	Acc.	a1	a2	a3	a4	Acc.	a1	a2	a3	a4	a5	a6
Correct partitioning		A	A	○	·		A	A	B	B	·	·
DecisionStump	0.74	·	·	·	·	0.68	·	·	·	·	·	·
OneR	0.74	·	·	·	·	0.68	·	·	·	·	·	·
SMO	0.74	·	·	·	·	0.68	·	·	·	·	·	·
naiveBayes	0.72	·	·	○	·	0.68	·	·	·	·	·	·
AdaBoostM1	0.69	A	A	A	A	0.68	·	·	·	·	·	·
Logistic	0.69	A	A	A	A	0.68	·	·	·	·	·	·
LogitBoost	0.69	A	A	A	A	0.68	·	·	·	·	·	·
Bagging	0.91	A	A	○	·	0.90	A	A	B	B	·	·
IBk	0.91	A	A	○	·	0.90	A	A	B	B	·	·
J48	0.91	A	A	○	·	0.90	A	A	B	B	·	·
JRip	0.91	A	A	○	·	0.90	A	A	B	B	·	·
LMT	0.91	A	A	○	·	0.90	A	A	B	B	·	·
PART	0.91	A	A	○	·	0.90	A	A	B	B	·	·
SMO radial	0.91	A	A	○	·	0.90	A	A	B	B	·	·
randomForest	0.90	A	A	○	·	0.88	B	B	A	A	·	·

The accuracies of the classifers are also provided
Major class proportion of the `Single-XOR` dataset: 0.72
Major class proportion of the `Double-XOR` dataset: 0.70

`Single-XOR` and `Double-XOR` datasets. However, some classifiers such as DecisionStump, OneR and naïve Bayes are incapable of discovering the complex relationships in the synthetic datasets due to the nature of these classifiers. The structure used by the classifiers is also reflected in the classification accuracy; those classifiers that exploit the correct structure in the datasets come close to the optimal classification accuracy, which for both synthetic datasets is 0.90, due to the addition of 10% noise. In the `Double-XOR` dataset, there is no importance relation between the groups formed by attributes $\{\{1, 2\}\}$ and the group formed by $\{\{3, 4\}\}$. Hence, the order in which the groups have been discovered varies between the classifiers, as seen in Table 3 for, e.g., J48 and randomForest.

In these datasets, the classifiers OneR and DecisionStump work as majority class predictors, which for these datasets leads to relatively high accuracy, although the structure of the dataset is not properly utilized. Also, the SVM with a linear kernel (SMO) does not utilize the structure in either dataset, whereas the SVM with a radial kernel (SMO radial) uses the correct structure in both datasets. These issues highlight the importance of not only considering classification accuracy, but also how the classifier is using the structure in the dataset. The same type of classifier can also perform differently depending on options such as, e.g., the type of kernel for SVMs.

In the `Single-XOR` dataset, the AdaBoostM1, Logistic and LogitBoost classifiers perceive the attributes a1–a4 as being one group. The low performance of these classifiers could be explained by the fact that the these classifiers exploit the structure of the attributes in a non-optimal manner.

4.2.2 UCI datasets

The results from the grouping of attributes in the UCI datasets, using all classifiers, are shown in Table 4. The table shows the number of groups and the number of attributes included in the groups for each classifier and for each dataset. The number of groups and attributes in groups are also averaged over both classifiers and datasets. These average numbers can be used to draw general conclusions regarding the datasets and the classifiers.

The nature of the relationships in a dataset can be determined from the average number of groups and attributes discovered by the classifiers. If the average number of groups is about the same as the number as attributes in the groups, it means that the groupings contain mostly singleton groups. Such datasets can typically be well modeled by a naïve Bayes classifier, since there are no associations between the attributes, i.e., they are independent. This holds for the datasets `iris`, `breast-w`, `vote` (with an exception for the Logistic classifier) and `zoo`.

For the `hepatitis` dataset, it appears that classifiers fail to effectively use the structure of the data. The radial SMO classifier uses no attribute, whereas J48, JRip and LMT use only one attribute. IBk uses one large cluster with 13 attributes, whereas all other classifiers use all-singleton structures with maximum 7 attributes. Even without using any of the structure in the data, the classifiers can still work as majority class predictors and reach an accuracy of at least 0.84.

The average number of groups and number of attributes for each classifier also reveals interesting information. As expected, DecisionStump and OneR use at most a singleton group, by design. Also, the naïve Bayes classifier has the highest average number of groups. This shows that this classifier indeed usually perceives attributes as independent, as expected. There are, however, exceptions, as for instance in the `vowel` dataset (Table 7) for which the classifier uses both a larger group and several singletons.

Some classifiers, e.g., as generated by IBk, LMT, Logistic and the SVMs, appear to often use groupings in which the groups contain several attributes. This implies that they are indeed utilizing associations between attributes.

Examples of groupings for the `diabetes` and `glass` datasets are shown in Tables 5 and 6. The tables show both the discovered groupings and the rank ordering of the individual attributes, using the method presented in Sect. 4.1.2. The groupings show, e.g., that the variables `plas` and `mass` are found to be associated by every classifier that can use more than one attribute, except by the radial SMO, IBk and PART classifiers. Also, the `skin` attribute is used only by one classifier. The ranking of the individual attributes shows that the most important variables are usually included in the groupings. However, there are differences between the rankings by the different classifiers, which shows that the same attribute can be exploited differently by different classifiers. Interestingly, the naïve Bayes classifier uses a large grouping of attributes, instead of only singletons. This indicates that the attributes in the dataset are not independent.

For the `glass` dataset, the individually most important attributes are also here included in the groupings, either in groups or as singletons. Here, the attributes `mg`, `al` and `ri` are often grouped together, but different classifiers use different attributes in

Table 4 Summary of grouping of attributes for the UCI datasets

	Size	Classes	Attributes	Major class	Average grouping	AdaBoostM1	Bagging	DecisionStump	IBk	J48	JRip	LMT	Logistic	LogitBoost	OneR	PART	SMO	SMO radial	naiveBayes	randomForest
Average grouping						2 (2)	2 (5)	1 (1)	5 (12)	3 (6)	4 (6)	4 (8)	4 (12)	4 (7)	1 (1)	4 (7)	4 (12)	5 (11)	8 (11)	5 (9)
balance-scale	625	3	4	0.46	2 (3)	1 (3)	2 (4)	1 (1)	2 (4)	2 (4)	1 (3)	2 (4)	1 (4)	2 (4)	1 (1)	2 (4)	2 (4)	2 (4)	2 (4)	2 (4)
iris	150	3	4	0.33	2 (2)	2 (2)	1 (1)	1 (1)	2 (2)	2 (2)	1 (1)	2 (2)	2 (2)	2 (2)	1 (1)	2 (2)	2 (2)	2 (2)	2 (2)	2 (2)
diabetes	768	2	8	0.65	1 (4)	1 (3)	2 (4)	0 (0)	1 (4)	1 (3)	2 (6)	1 (5)	1 (4)	2 (5)	1 (1)	2 (3)	1 (5)	1 (4)	2 (6)	1 (5)
breast-w	683	2	9	0.65	3 (3)	3 (3)	3 (3)	1 (1)	4 (4)	3 (3)	3 (3)	3 (3)	4 (4)	3 (3)	1 (1)	2 (2)	3 (3)	7 (7)	8 (8)	4 (4)
glass	214	6	9	0.36	2 (6)	1 (1)	2 (7)	1 (1)	3 (7)	2 (6)	3 (3)	4 (8)	4 (7)	4 (8)	1 (1)	4 (8)	2 (6)	4 (8)	1 (9)	1 (8)
breast-cancer	277	2	9	0.71	1 (4)	2 (2)	0 (0)	1 (1)	2 (7)	2 (2)	2 (2)	1 (5)	1 (7)	1 (6)	1 (1)	2 (5)	1 (7)	0 (0)	1 (8)	1 (6)
heart-c	296	2	13	0.54	3 (6)	6 (6)	1 (7)	1 (1)	1 (7)	2 (5)	1 (4)	1 (10)	1 (9)	5 (5)	1 (1)	2 (7)	6 (6)	3 (7)	7 (7)	1 (12)
heart-statlog	270	2	13	0.56	3 (5)	1 (6)	1 (5)	1 (1)	1 (8)	1 (4)	3 (3)	5 (5)	6 (6)	4 (4)	1 (1)	2 (4)	5 (5)	2 (9)	6 (6)	5 (5)
vowel	990	11	13	0.09	4 (8)	1 (1)	2 (3)	1 (1)	6 (13)	4 (9)	2 (12)	4 (10)	5 (7)	6 (9)	1 (1)	5 (8)	7 (13)	5 (12)	6 (10)	2 (9)
credit-a	653	2	15	0.55	2 (4)	1 (1)	1 (6)	1 (1)	1 (6)	1 (4)	1 (4)	1 (9)	5 (6)	4 (4)	1 (1)	2 (5)	1 (1)	3 (8)	5 (5)	4 (4)
vote	232	2	16	0.53	3 (3)	2 (2)	1 (1)	1 (1)	5 (5)	1 (1)	1 (1)	1 (1)	1 (13)	2 (2)	1 (1)	1 (1)	1 (1)	13 (13)	6 (6)	1 (1)
primary-tumor	132	18	17	0.21	6 (10)	1 (1)	8 (8)	1 (1)	5 (12)	9 (9)	8 (8)	6 (14)	7 (15)	3 (15)	1 (1)	11 (11)	7 (14)	12 (12)	5 (15)	
zoo	101	7	17	0.41	5 (5)	1 (1)	0 (0)	1 (1)	10 (10)	5 (5)	6 (6)	5 (5)	9 (9)	5 (5)	0 (0)	5 (5)	10 (10)	0 (0)	10 (10)	
lymph	148	4	18	0.55	4 (6)	2 (2)	3 (3)	1 (1)	2 (13)	4 (4)	4 (4)	6 (6)	2 (12)	7 (7)	1 (1)	2 (8)	11 (11)	0 (0)	7 (7)	7 (7)
vehicle	846	4	18	0.26	5 (10)	1 (1)	4 (7)	1 (1)	7 (15)	5 (10)	4 (11)	5 (14)	5 (8)	3 (11)	1 (1)	8 (11)	7 (13)	8 (16)	13 (13)	2 (13)
hepatitis	80	2	19	0.84	3 (3)	3 (3)	3 (3)	0 (0)	1 (13)	1 (1)	1 (1)	1 (1)	6 (6)	4 (4)	1 (1)	3 (3)	4 (4)	0 (0)	7 (7)	4 (4)
segment	2310	7	19	0.14	5 (10)	1 (1)	3 (5)	1 (1)	11 (13)	7 (9)	7 (10)	6 (13)	11 (13)	6 (10)	1 (1)	7 (10)	9 (11)	9 (13)	1 (18)	1 (15)
credit-g	1000	2	20	0.70	2 (5)	2 (3)	1 (3)	0 (0)	1 (9)	2 (5)	1 (3)	2 (7)	2 (8)	2 (4)	1 (1)	3 (7)	3 (10)	0 (0)	1 (9)	2 (8)
mushroom	5644	2	22	0.62	2 (3)	1 (1)	1 (1)	1 (1)	3 (12)	1 (3)	1 (2)	2 (5)	5 (5)	2 (5)	1 (1)	1 (5)	3 (3)	0 (0)	5 (5)	2 (2)
autos	159	6	25	0.30	7 (9)	1 (1)	1 (3)	1 (1)	11 (15)	6 (6)	6 (6)	3 (7)	7 (14)	3 (9)	1 (1)	7 (7)	12 (18)	14 (17)	17 (17)	14 (14)
ionosphere	351	2	34	0.64	5 (11)	3 (3)	8 (8)	1 (1)	6 (26)	5 (6)	4 (4)	8 (12)	4 (14)	1 (9)	1 (1)	5 (7)	3 (17)	5 (30)	14 (14)	13 (13)
soybean	562	15	35	0.16	10 (13)	1 (1)	8 (10)	1 (1)	13 (24)	8 (11)	15 (17)	15 (15)	14 (27)	17 (17)	1 (1)	11 (16)	8 (24)	0 (0)	20 (20)	18 (18)
kr-vs-kp	3196	2	36	0.52	4 (12)	3 (4)	3 (9)	1 (1)	4 (13)	3 (15)	16 (20)	3 (12)	5 (16)	3 (4)	1 (1)	3 (12)	3 (25)	3 (9)	3 (18)	2 (18)
anneal	898	5	38	0.76	4 (9)	2 (2)	2 (3)	1 (1)	5 (8)	3 (4)	3 (4)	3 (6)	2 (37)	4 (10)	1 (1)	2 (10)	3 (8)	8 (17)	11 (17)	6 (6)
waveform-5000	5000	3	40	0.34	4 (19)	1 (1)	3 (18)	1 (1)	11 (39)	3 (25)	4 (12)	1 (15)	1 (32)	2 (13)	1 (1)	9 (13)	1 (36)	1 (40)	14 (14)	1 (32)
sonar	208	2	60	0.53	7 (17)	3 (9)	1 (13)	1 (1)	16 (31)	1 (5)	2 (5)	5 (14)	5 (26)	1 (10)	1 (1)	3 (7)	2 (44)	25 (51)	21 (21)	16 (16)

The first four columns gives the number of instances, number of classes and attributes and the proportion of the major class in the datasets. Each classifier is represented by two numbers. The first number shows the number of groups and the second number in parentheses shows the total number of attributes included in the grouping. The average groupings are calculated for the datasets and classifiers by aggregating over the number of groups and attributes

Table 5 Grouping of the `diabetes` dataset

`diabetes`		original accuracy	final accuracy	Bayes fidelity	final fidelity	plas	mass	preg	age	pedi	insu	pres	skin	plas	mass	preg	age	pedi	insu	pres	skin
size:	768																				
classes:	2																				
attributes:	8																				
major class:	0.65																				
Classifier																					
DecisionStump		0.66	0.66	1.00	1.00	·	·	·	·	·	·	·	·	2	6	1	8	7	5	3	4
SMO radial		0.76	0.70	0.88	0.88	A	·	A	·	A	A	·	·	1	4	2	6	3	5	7	8
IBk		0.69	0.59	0.67	0.69	A	·	A	A	A	·	·	·	2	7	1	3	4	8	5	6
AdaBoostM1		0.75	0.75	0.91	1.00	A	A	·	A	·	·	·	·	1	2	4	3	8	7	5	6
J48		0.75	0.72	0.86	0.95	A	A	·	A	·	·	·	·	1	3	8	2	6	7	4	5
Bagging		0.74	0.70	0.85	0.90	A	A	·	○	·	A	·	·	1	2	7	3	4	8	6	5
LogitBoost		0.76	0.74	0.89	0.95	A	A	·	○	·	A	A	·	1	2	7	3	6	5	4	8
Logistic		0.77	0.74	0.88	0.93	A	A	A	·	·	·	A	·	1	2	3	7	6	5	4	8
SMO		0.77	0.75	0.92	0.98	A	A	A	·	A	·	A	·	1	3	2	8	4	6	5	7
LMT		0.78	0.76	0.91	0.97	A	A	A	·	A	A	·	·	1	2	3	8	4	6	5	7
naiveBayes		0.78	0.75	0.90	0.93	A	A	A	A	·	○	A	·	1	3	2	4	8	5	6	7
randomForest		0.76	0.74	0.87	0.95	A	A	A	A	A	·	·	·	1	2	5	4	3	6	8	7
JRip		0.71	0.69	0.83	0.92	A	A	B	·	B	B	·	B	1	3	4	8	2	5	6	7
PART		0.78	0.75	0.88	0.95	A	○	·	A	·	·	·	·	1	3	5	2	8	7	4	6
OneR		0.73	0.73	1.00	1.00	○	·	·	·	·	·	·	·	1	6	2	8	7	5	3	4

The *original accuracy* shows the accuracy of the classifier without any randomization, *final accuracy* is the accuracy after randomizing using the grouping. *Bayes fidelity* is the fidelity when all attributes are in singleton groups. The final fidelity is calculated using the grouping of attributes discovered. Groups are denoted by roman letters. Singleton groups are marked with ○ and pruned-out singletons with ·. The rank order of importance of individual attributes is shown to the right of the groupings with numbers

the groupings. The accuracies of the classifiers are also generally low for this dataset, which in part could be explained by complex interactions between the attributes. The best accuracy is reached by random forest and Bagging. It is interesting to note, that although the SVM with a radial kernel uses a structure very similar to random forest and Bagging, the performance of the radial SMO classifier is lower. The performance of the radial SMO classifier could be explained by the known sensitivity of this classifier to parameter settings and the fact that default values were used in this experiment. These results show, that although different classifiers utilize similar combinations of attributes, no conclusions can directly be drawn regarding classification performance.

4.2.3 Stability of groupings

The stability of the groupings was investigated using the cross-validation scheme described above in Sect. 4.1.2. The results for the `vowel` dataset are shown in Table 7, for selected classifiers. The groupings vary between the cross-validation folds for a single classifier, due to the fact that the fold used to generate the classifier and the three folds used for analyzing the classifier are non-overlapping. However, the general structure is coherent within the classifiers, although the classifiers exploit the structure of the data differently. In general, several classifiers share a common structure for the grouping, e.g., randomForest, JRip and PART. Also, the results indicate that the

Table 6 Grouping of the `glass` dataset

glass		original accuracy	final accuracy	Bayes fidelity	final fidelity	mg	al	ri	si	na	fe	ca	k	ba	mg	al	ri	si	na	fe	ca	k	ba
size:	214																						
classes:	6																						
attributes:	9																						
major class:	0.36																						
Classifier																							
OneR		0.52	0.52	1.00	1.00	·	○	·	·	·	·	·	·	·	4	1	2	5	3	9	7	6	8
JRip		0.55	0.51	0.94	0.91	·	○	·	·	·	·	○	○	·	7	1	5	4	6	9	3	2	8
SMO		0.51	0.50	0.87	0.96	A	A	A	·	·	A	A	○	·	1	2	3	7	8	4	5	6	9
J48		0.58	0.57	0.87	0.97	A	A	A	·	A	A	·	·	○	2	1	5	7	4	6	9	8	3
randomForest		0.73	0.72	0.89	0.99	A	A	A	A	A	A	A	A	·	2	1	3	5	4	8	6	7	9
naiveBayes		0.52	0.51	0.90	0.96	A	A	A	A	A	A	A	A	A	1	4	8	9	7	5	6	3	2
Bagging		0.72	0.69	0.88	0.94	A	A	A	A	A	A	○	·	·	1	2	4	5	3	7	6	8	9
PART		0.63	0.57	0.78	0.90	A	A	A	A	A	○	·	○	○	4	1	7	5	2	8	9	6	3
IBk		0.69	0.55	0.64	0.74	A	A	A	A	○	A	○	·	·	1	2	4	3	5	6	7	8	9
SMO radial		0.66	0.60	0.78	0.89	A	A	A	A	○	A	○	○	·	1	2	4	7	5	3	6	8	9
LMT		0.55	0.52	0.77	0.88	A	A	A	A	○	○	·	A	○	2	1	7	5	4	8	9	3	6
Logistic		0.56	0.43	0.54	0.63	A	○	·	A	A	·	A	○	○	4	6	7	1	3	9	5	2	8
AdaBoostM1		0.47	0.47	1.00	1.00	○	·	·	·	·	·	·	·	·	1	4	2	5	3	9	7	6	8
DecisionStump		0.47	0.47	1.00	1.00	○	·	·	·	·	·	·	·	·	1	4	2	5	3	9	7	6	8
LogitBoost		0.65	0.61	0.78	0.91	○	A	A	A	A	·	A	○	○	4	5	3	6	2	9	1	8	7

The structure of the table is identical to Table 5

attributes `feature_0` and `feature_1` are associated, as they appear in the same group for several classifiers.

4.2.4 Effect of varying δ

The effect of varying the sensitivity parameter δ is shown in Table 8 for the JRip, LMT, and randomForest classifiers. These results show, that the number and size of the groups vary depending on the value of δ, since it directly affects the grouping threshold Δ in Algorithm 2. In general, increasing the size of δ leads to larger group sizes, and can be seen as having the effect of merging groups, leading to a smaller number of groups, as parametrized by k in Problem 1. However, if δ is increased over a critical threshold, this results in an all-singleton grouping, as a result of the criterion on line 7 in Algorithm 2.

4.2.5 Scalability

The proposed method is computationally reasonably efficient, because the classifier needs to be trained only once for the unrandomized data and the permutation randomizations can be efficiently produced. Furthermore, the randomization is easily parallelizable, e.g., the step on line 12 in Algorithm 2 is embarrassingly parallel. The number of rows in the dataset does not affect the running time asymptotically, because a large dataset can be downsampled to a size which gives a sufficient accuracy to the fidelity estimate, as discussed in Sect. 3.1. The largest factor in the running time is that

Table 7 Groupings of cross-validation folds of the `Vowel` dataset for selected classifiers

vowel size: 990 classes: 11 attributes: 13 major class: 0.09 **Classifier**	original accuracy	final accuracy	Bayes fidelity	final fidelity	feature_1	feature_0	feature_4	feature_5	feature_3	feature_2	feature_7	spekr_nmbr	feature_8	feature_6	feature_9	sex	tran_r_tst
IBk (fold 1)	0.71	0.37	0.32	0.43	o	o	o	A	o	A	o	A	B	B	B	B	·
IBk (fold 2)	0.70	0.30	0.31	0.42	o	o	C	A	C	C	B	A	B	·	C	B	A
IBk (fold 3)	0.71	0.34	0.33	0.48	o	C	o	C	C	C	C	A	A	·	B	B	B
J48 (fold 1)	0.63	0.45	0.56	0.67	A	A	B	B	o	B	B	·	o	B	B	B	B
J48 (fold 2)	0.54	0.40	0.66	0.70	A	A	o	o	o	·	o	·	A	·	o	·	·
J48 (fold 3)	0.56	0.43	0.62	0.68	A	A	·	A	o	o	o	·	o	·	o	·	o
JRip (fold 1)	0.49	0.36	0.67	0.64	A	A	B	B	A	B	B	B	B	B	B	B	B
JRip (fold 2)	0.50	0.38	0.68	0.73	A	A	o	o	A	·	·	·	·	·	o	·	·
JRip (fold 3)	0.48	0.40	0.67	0.80	A	B	A	B	A	B	·	·	·	B	B	·	B
LMT (fold 1)	0.73	0.46	0.54	0.61	A	A	o	o	o	o	o	A	o	·	·	·	·
LMT (fold 2)	0.75	0.48	0.55	0.60	A	A	o	o	o	o	o	A	·	o	·	·	·
LMT (fold 3)	0.72	0.46	0.58	0.58	A	A	o	·	o	o	o	A	·	·	o	·	·
Logistic (fold 1)	0.62	0.34	0.41	0.45	o	A	A	A	o	A	o	A	·	o	·	·	·
Logistic (fold 2)	0.65	0.37	0.39	0.52	A	B	A	B	A	B	B	A	B	B	B	·	·
Logistic (fold 3)	0.69	0.37	0.41	0.46	A	o	o	A	o	A	o	A	·	·	A	·	·
LogitBoost (fold 1)	0.67	0.50	0.68	0.70	A	A	o	o	·	o	A	·	o	A	·	·	·
LogitBoost (fold 2)	0.65	0.50	0.70	0.70	A	A	o	o	o	A	A	·	·	A	·	·	·
LogitBoost (fold 3)	0.68	0.55	0.71	0.76	A	A	A	o	o	o	A	·	o	·	·	·	·
PART (fold 1)	0.61	0.45	0.57	0.69	A	A	B	·	B	B	B	·	B	·	·	B	·
PART (fold 2)	0.61	0.49	0.60	0.74	A	A	B	o	B	B	o	B	o	·	·	B	·
PART (fold 3)	0.54	0.42	0.64	0.69	A	A	o	o	o	A	o	·	o	·	·	·	·
SMO radial (fold 1)	0.54	0.12	0.17	0.17	o	o	·	·	·	·	o	A	·	·	·	A	o
SMO radial (fold 2)	0.56	0.19	0.21	0.24	o	o	o	o	·	·	·	A	·	·	o	o	A
SMO radial (fold 3)	0.56	0.15	0.20	0.24	o	o	·	o	o	o	o	A	·	·	·	A	o
naiveBayes (fold 1)	0.48	0.43	0.79	0.84	A	A	o	o	o	A	o	A	o	A	o	·	·
naiveBayes (fold 2)	0.44	0.39	0.78	0.86	A	A	A	o	o	o	A	A	A	A	o	·	·
naiveBayes (fold 3)	0.54	0.48	0.79	0.85	A	A	A	A	o	o	o	A	o	A	·	·	·
randomForest (fold 1)	0.80	0.62	0.66	0.78	A	A	A	B	B	o	B	B	B	B	·	·	·
randomForest (fold 2)	0.79	0.60	0.69	0.74	A	A	A	o	A	o	·	·	o	o	·	·	·
randomForest (fold 3)	0.81	0.61	0.67	0.74	A	A	A	o	A	o	o	·	o	·	o	·	·

The columns are as in Table 5

the iterative algorithm requires a quadratic number of steps with respect to the number of attributes. If the number of attributes becomes too large, it may therefore be necessary to reduce the number of attributes, e.g., by pruning out unnecessary attributes, e.g., by the Breiman method (Breiman 2001) before applying the `GoldenEye` algorithm. In the experiments discussed in this paper, the time per classifier to produce groupings of attributes ranged from a few seconds for the smallest dataset with 4 attributes to around ten minutes for the largest dataset with 60 attributes, using an unoptimized and

Table 8 Groupings using different values of δ for the `vowel` dataset for selected classifiers

`vowel` size: 990 classes: 11 attributes: 13 major class: 0.09																	
Classifier	δ	original accuracy	final accuracy	Bayes fidelity	final fidelity	`feature_0`	`feature_1`	`feature_3`	`feature_4`	`feature_5`	`feature_2`	`feature_8`	`feature_7`	`spekr_nmbr`	`feature_9`	`feature_6`	`tran_r_tst`	`sex`
JRip	0.05	0.56	0.45	0.63	0.75	A	A	B	○	A	B	B	B	B	B	B	·	·
JRip	0.1	0.56	0.43	0.64	0.72	A	A	○	○	A	○	A	·	·	·	○	·	·
JRip	0.15	0.57	0.50	0.68	0.83	A	A	A	○	A	○	A	·	·	A	·	·	·
JRip	0.2	0.58	0.49	0.62	0.82	A	A	A	○	A	·	A	·	·	A	○	·	·
JRip	0.25	0.58	0.55	0.65	0.91	A	A	A	○	A	A	A	·	·	A	○	·	·
JRip	0.3	0.57	0.57	0.67	1.00	A	A	A	A	A	A	A	A	·	A	A	·	·
JRip	1.0	0.56	0.37	0.71	0.58	○	○	○	○	○	·	○	·	·	·	·	·	·
LMT	0.05	0.87	0.51	0.47	0.56	A	A	○	○	○	○	○	○	A	·	·	○	·
LMT	0.1	0.85	0.51	0.45	0.56	A	A	○	○	○	○	○	○	A	·	·	·	·
LMT	0.15	0.87	0.52	0.47	0.57	A	A	○	○	·	A	·	○	A	·	·	·	·
LMT	0.2	0.88	0.65	0.52	0.69	A	A	○	A	A	A	·	A	A	○	·	·	·
LMT	0.25	0.89	0.72	0.55	0.79	A	A	A	A	A	A	·	A	A	·	○	·	·
LMT	0.3	0.86	0.75	0.54	0.84	A	A	A	A	A	A	·	A	A	·	A	·	·
LMT	1.0	0.89	0.39	1.00	0.42	○	○	○	○	○	·	·	○	·	·	·	·	·
randomForest	0.05	0.91	0.67	0.64	0.73	A	A	A	B	B	B	·	B	B	B	·	·	·
randomForest	0.1	0.92	0.67	0.63	0.69	A	A	A	○	A	○	·	○	·	·	·	·	·
randomForest	0.15	0.92	0.70	0.64	0.76	A	A	A	A	A	·	○	○	·	·	·	·	·
randomForest	0.2	0.92	0.71	0.63	0.78	A	A	A	A	A	·	○	·	·	·	·	·	·
randomForest	0.25	0.92	0.85	0.69	0.93	A	A	A	A	A	·	A	A	A	·	A	·	·
randomForest	0.3	0.91	0.55	0.74	0.60	○	○	○	○	·	·	○	○	·	·	·	·	·
randomForest	1.0	0.92	0.54	1.00	0.60	○	○	○	○	○	○	·	○	·	·	·	·	·

The columns are as in Table 5

sequential R implementation running on a computer with the GNU/Linux operating system with an eight-core 3.07GHz Intel i7 CPU and 12GB of RAM (using only one core).

5 Discussion

The results from the synthetic datasets highlight the importance of considering how classifiers use the structure of a dataset. If classification accuracy is used as a sole measure of goodness, wrong conclusions can easily be drawn. When comparing classifiers, it is of great value to be able to discern how the classifier is using the structure of the data. This is especially helpful when some idea of the true association between attributes exist; it can then be verified if the classifier correctly exploits the structure. However, two classifiers using a similar structure in the data do not necessarily perform similarly, as seen in the results, e.g., for the `glass` dataset.

The groupings can reveal interesting patterns. The naïve Bayes classifier sometimes takes associations between attributes into account, as seen in the `diabetes` dataset. This is in line with the observations of Domingos and Pazzani (1997), who demon-

Table 9 Two examples of binary datasets with $n = 6$ rows and $m = 2$ attributes

(a) Correlated

c	y	$\mathcal{A}_1$	$\mathcal{A}_2$
+	+	1	0
+	+	1	0
+	+	0	1
+	+	0	1
−	−	0	0
−	−	0	0

(b) Uncorrelated

c	y	$\mathcal{A}_1$	$\mathcal{A}_2$
+	+	1	1
+	+	1	0
+	+	0	1
+	−	0	0
−	−	0	0
−	−	0	0

For both datasets $\text{fid}(\{\{1, 2\}\}) = 1$, and a simple naïve Bayes classifier can be defined by $f(\mathcal{A}) = +$ if and only if $\mathcal{A}_1 + \mathcal{A}_2 \geq 1/2$. (a) The attributes are correlated within class +, hence, the naïve Bayes assumption does not hold. The classifier achieves perfect accuracy on the original data, the fidelity being $\text{fid}(\{\{1\}, \{2\}\}) = 5/6 = 0.83$. (b) The attributes are independent within classes. The accuracy of the classifier is $5/6 = 0.83$ and the fidelity of the singleton grouping $\text{fid}(\{\{1\}, \{2\}\}) = 17/18 = 0.94$. Due to the correlation, the `GoldenEye` algorithm finds the grouping $\{1, 2\}$ in case (a) but only singletons in case (b), with a suitable choice of the parameter δ (e.g., $\delta = 0.1$ in this example). The columns are as in Table 1

strated that the naïve Bayes classifier works well where it intuitively should not due to the independence assumption. A simple example where the naïve Bayes classifier utilizes correlations between attributes is also shown in Table 9.

The groupings of attributes are not always consistent between different classifiers for the same dataset as observed for the UCI datasets. The discovered groupings hence reflect the assumptions of the classifier and represent the interaction between the latent associations in the data and the classifier.

Attributes that on an individual level are important are most often included in the groupings. However, the groupings presented here provide additional information over the individual attribute importance scores, by allowing associations between the attributes to be found. The nature of the association between the attributes cannot, however, be directly determined based on the groupings. The reason why certain attributes must be kept in the same group could be due to interactive effects, i.e., the individual attributes provide no information of the class label on their own, or due to additive effects, i.e., the individual attributes each provide some, but not perfect, information of the class label. How to further characterize the relations between attributes in a group is an interesting issue that could be further pursued in future work.

The main utility of the groupings is in explorative data analysis. The method for finding the groupings is based on randomizing the data. No assumptions are made regarding either the nature of the data or the classifier being investigated. In contrast to the method proposed in Ojala and Garriga (2010), where randomization was employed for investigating whether or not a classifier is utilizing the structure in the data, our method goes beyond that by showing also which attributes actually are exploited by the classifier. This information also extends the comparison of classifiers beyond a simple comparison of classification accuracy. The groupings permit similarities between classifiers to be found within a single dataset, thus revealing some of the associations between the attributes.

In this study, we have chosen to use fidelity as the goodness measure. It would, however, be possible to use other measures as well. For example, in its current form, fidelity emphasizes larger classes, similarly to many other performance metrics. In some applications it might thus be useful to replace fidelity by a goodness measure taking class balance into account. Another possibility to account for class imbalance would be to sample the dataset so that the classes would be of equal size.

The problem addressed in this paper can be viewed as a constrained randomization problem, with the groups acting as constraints (Lijffijt et al. 2014). In the absence of any groups, i.e., $\mathcal{S} = \emptyset$, the columns of the data matrix are permuted uniformly at random. As groups are added, the randomizations are restricted to a subset of uniform randomizations, i.e., the groups of attributes in $\mathcal{S}$ correspond to constraints on the randomization. Even though the groupings presented in this paper are produced by an algorithm (`GoldenEye`), the constrained randomization methods are well suited for interactive explorative data analysis, as proposed in Hanhijärvi et al. (2009). At each iteration, the user can add constraints, e.g., adding an attribute to an existing group of attributes, or remove them, e.g., removing an existing group of attributes. The change in fidelity gives information about how important the added or removed constraint is, given the prior constraints.

The randomization approach has a major advantage in that it cleanly separates the algorithmic data analysis method, here the classifier function, and the statistical treatment of patterns, here represented by random permutations of attributes where the groupings of attributes are used as constraints. As a result, it is not necessary to tweak or see inside a classifier to understand how it works on a particular dataset.

6 Concluding remarks

In this paper we have presented an efficient randomization-based algorithm for discovering how classifiers exploit associations between attributes in a dataset. The method makes no assumptions regarding the distributions in the data or regarding the classifier. The structure of the exploited attributes is presented in the form of groupings, which may provide insights into the structure of the dataset. This allows observations to be made, e.g., that the naïve Bayes classifier actually exploits dependencies between attributes in some cases. The groupings extend the notion of attribute importance to sets, and in addition to only providing importance scores, also reveals which attributes are associated.

The method of detecting groupings of attributes is a usable tool in explorative data mining tasks, as it provides insight into otherwise opaque classifiers and thus aids in the interpretation of the results. The method is also closely related to permutation testing, as the groupings identified are in effect used as constraints in the randomization process. In this way, the method could be extended to hypothesis testing of structures in datasets, which is an interesting area of future research. Also, the method presented here opens up other new avenues for research.

One direction for future work is to investigate the use of the proposed method for feature selection, e.g., for dimensionality reduction or minimizing feature acquisition cost. The found groupings could be an effective alternative to evaluating and selecting each feature individually, which typically leads to sub-optimal feature sets.

Another, related, direction for future work is to exploit the found groupings when constructing ensemble classifiers. For example, methods that rely on sampling from the feature set may benefit from taking the discovered associations into account. Another direction for future work is to use characterizations of found groupings in a meta-learning framework, i.e., to choose a suitable (computationally costly) learning algorithm based on an analysis of the found groupings when applying learning algorithms of low computational cost. For example, the extent to which groupings are non-singletons when applying a naïve Bayes classifier may provide some information on what type of kernel to use for an SVM classifier.

Acknowledgments AH and KP were partly supported by the Revolution of Knowledge Work project, funded by Tekes. HB and LA were partly supported by the project High-Performance Data Mining for Drug Effect Detection at Stockholm University, funded by Swedish Foundation for Strategic Research under grant IIS11-0053.

References

Andrews R, Diederich J, Tickle AB (1995) Survey and critique of techniques for extracting rules from trained artificial neural networks. Knowl Based Syst 8(6):373–389

Bache K, Lichman M (2014) UCI machine learning repository. http://archive.ics.uci.edu/ml

Breiman L (2001) Random forests. Mach Learn 45(1):5–32

Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and regression trees. Wadsworth, Belmont

Chanda P, Cho YR, Zhang A, Ramanathan M (2009) Mining of attribute interactions using information theoretic metrics. In: IEEE International Conference on Data Mining Workshops, pp 350–355

Clark P, Niblett T (1989) The CN2 induction algorithm. Mach Learn 3(4):261–283

Cortes C, Vapnik V (1995) Support-vector networks. Mach Learn 20:273–293

De Bie T (2011a) An information theoretic framework for data mining. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, KDD '11, pp 564–572

De Bie T (2011b) Maximum entropy models and subjective interestingness: an application to tiles in binary databases. Data Min Knowl Discov 23(3):407–446

Domingos P, Pazzani MJ (1997) On the optimality of the simple bayesian classifier under zero-one loss. Machine Learning 29(2–3):103–130

Freitas AA (2001) Understanding the crucial role of attribute interaction in data mining. Artif Intell Rev 16(3):177–199

Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The weka data mining software: an update. ACM SIGKDD Explor Newsl 11(1):10–18

Hanhijärvi S, Ojala M, Vuokko N, Puolamäki K, Tatti N, Mannila H (2009) Tell me something i don't know: Randomization strategies for iterative data mining. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, KDD '09, pp 379–388

Henelius A, Korpela J, Puolamäki K (2013) Explaining interval sequences by randomization. In: Blockeel H, Kersting K, Nijssen S, Železný Filip (eds) Machine Learning and Knowledge Discovery in Databases, Lecture Notes in Computer Science, vol 8188, pp 337–352

Hornik K, Buchta C, Zeileis A (2009) Open-source machine learning: R meets Weka. Comput Stat 24(2):225–232. doi:10.1007/s00180-008-0119-7

Ishibuchi H, Nojima Y (2007) Analysis of interpretability-accuracy tradeoff of fuzzy systems by multiobjective fuzzy genetics-based machine learning. Int J Approx Reason 44(1):4–31

Jakulin A, Bratko I, Smrke D, Demsar J, Zupan B (2003) Attribute interactions in medical data analysis. In: 9th Conference on Artificial Intelligence in Medicine in Europe, pp 229–238

Janitza S, Strobl C, Boulesteix AL (2013) An auc-based permutation variable importance measure for random forests. BMC Bioinform 14:119

Johansson U, König R, Niklasson L (2003) Rule extraction from trained neural networks using genetic programming. In: 13th International Conference on Artificial Neural Networks, pp 13–16

Liaw A, Wiener M (2002) Classification and regression by randomforest. R News 2(3):18–22, http://CRAN.R-project.org/doc/Rnews/

Lijffijt J, Papapetrou P, Puolamäki K (2014) A statistical significance testing approach to mining the most informative set of patterns. Data Min Knowl Discov 28:238–263. doi:10.1007/s10618-012-0298-2

Misra G, Golshan B, Terzi E (2012) A Framework for Evaluating the Smoothness of Data-Mining Results. In: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases, vol II, pp 660–675

Ojala M, Garriga GC (2010) Permutation tests for studying classier performance. J Mach Learn Res 11:1833–1863

Plate T (1999) Accuracy versus interpretability in flexible modeling: implementing a tradeoff using gaussian process models. Behaviormetrika 26:29–50

Pulkkinen P, Koivisto H (2008) Fuzzy classifier identification using decision tree and multiobjective evolutionary algorithms. Int J Approx Reason 48(2):526–543

Quinlan JR (1986) Induction of decision trees. Mach Learn 1:81–106

R Core Team (2014) R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, http://www.R-project.org/

Segal MR, Cummings MP, Hubbard AE (2001) Relating amino acid sequence to phenotype: analysis of peptide-binding data. Biometrics 57(2):632–643

Strobl C, Boulesteix AL, Zeileis A, Hothorn T (2007) Bias in random forest variable importance measures: Illustrations, sources and a solution. BMC Bioinformatics 8(25):

Strobl C, Boulesteix AL, Kneib T, Augustin T, Zeileis A (2008) Conditional variable importance for random forests. BMC Bioinform 9:307

Wickham H, Chang W (2014) devtools: Tools to make developing R code easier. http://CRAN.R-project.org/package=devtools, r package version 1.5

Zacarias OP, Boström H (2013) Comparing support vector regression and random forests for predicting malaria incidence in Mozambique. In: International conference on advances in ICT for Emerging regions, IEEE, pp 217–221

Zhao Z, Liu H (2007) Searching for interacting features. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence, pp 1156–1161

Zhao Z, Liu H (2009) Searching for interacting features in subset selection. Intell Data Anal 13(2):207–228