Mach Learn (2010) 78: 421–428
DOI 10.1007/s10994-009-5164-0

TECHNICAL NOTE

Algorithms for subsetting attribute values with Relief

Janez Demšar

Received: 19 September 2008 / Revised: 13 November 2009 / Accepted: 16 November 2009 /
Published online: 22 December 2009

Abstract Relief is a measure of attribute quality which is often used for feature subset selection. Its use in induction of classification trees and rules, discretization, and other methods has however been hindered by its inability to suggest subsets of values of discrete attributes and thresholds for splitting continuous attributes into intervals. We present efficient algorithms for both tasks.

Keywords Machine learning · Attribute quality estimation · Relief

1 Introduction

Relief (Kira and Rendell 1992; Kononenko 1994) is a measure for assessing the quality of attributes. Its main advantage over the common impurity-based, strictly univariate measures is that it takes into account the effect of interacting attributes (Jakulin 2005). Kononenko and Šikonja (2007) list a number of potential uses of Relief, such as feature subset selection, choosing splitting criteria in tree induction, heuristics for induction of association and classification rules, discretization, feature ranking and weighing, and constructive induction. Its use for most of these purposes is however hindered by its inability to efficiently find thresholds for dividing values of continuous attributes in two intervals, or group values of discrete attribute into subsets. Even when Relief is used to identify the most relevant continuous attribute, deciding the splitting threshold (when required) is left to an impurity measure, which may be unable to find one. An extreme example is a domain with the class determined by the sign of the product of two continuous attributes with values from $[-1, 1]$. Relief would correctly identify the relevant attributes, yet the univariate measure used afterwards would fail to recognize the correct threshold, which would result in a random split.

Editor: Hendrik Blockeel.

J. Demšar (✉)
Faculty of Computer and Information Science, University of Ljubljana, Tržaška 25, Ljubljana, Slovenia
e-mail: janez.demsar@fri.uni-lj.si

Another shortcoming of Relief is that it assesses the relevance of a continuous attribute and not of the optimal split based on it, which is what we are actually interested in. A similar problem occurs with multi-valued attributes.

This paper presents efficient algorithms for computing the Relief score of all possible thresholds for continuous attributes and for grouping values of discrete attributes. Theoretical time complexity analysis and practical experience show that their execution times are comparable to that of the ordinary Relief.

2 Algorithms

The idea of Relief and its derivatives is to reward the attribute for having different values on a pair of similar examples from different classes, and punish it for having different values on examples from the same class. Our algorithms will be based on ReliefF (Kononenko 1994), but the formulation will be general enough for their use on all variations of Relief that we are aware of.

We will reformulate the definition of ReliefF to expose the role of pairs of examples. ReliefF randomly selects m reference examples E and for each finds its k closest neighbors from each of the c classes. Let F be one such neighbor. We will denote this set of mkc pairs (reference, neighbor) as $\mathcal{P}$. Each pair can *potentially contribute*

$$f(E,F)=\begin{cases}-\frac{1}{mk}; & C(E)=C(F)\\ \frac{1}{mk}\frac{p(C(F))}{1-p(C(E))}; & C(E)\neq C(F)\end{cases} \tag{1}$$

to ReliefF, where $C(E)$ and $C(F)$ are classes of E and F, and $p(C(E))$ and $p(C(F))$ are their prior probabilities. For an attribute A, the *actual contribution* of the pair depends on the attributes' values, E_A and F_A. In this paper we compute the ReliefF of discrete and binarized continuous attributes, which is defined as a sum of contributions of example pairs on which the attribute has different values,

$$R(A)=\sum_{(E,F)\in\mathcal{P}} f(E,F)\,[E_A\neq F_A]. \tag{2}$$

Pairs of examples with the same value of A do not contribute to ReliefF.

We will illustrate the algorithms on data from Table 1 with $m=3$ reference examples and $k=1$ neighbors from each class. Suppose that the algorithm randomly chose reference examples 1, 2, and 6, and that their closest neighbors from both classes are as given in the

Table 1 Data used in illustrations of algorithms. The right part shows the attributes derived by the algorithms presented in the paper

Id	A	B	…	y	Nearest neighbors	A_τ	$B_\mathcal{V}$
1	1	a	…	0	**4, 5**	0	V_1
2	2	b	…	1	**3, 5**	0	V_2
3	4	d	…	1		0	V_2
4	5	b	…	1		1	V_2
5	6	c	…	0		1	V_1
6	8	a	…	0	**3, 7**	1	V_1
7	9	c	…	0		1	V_1

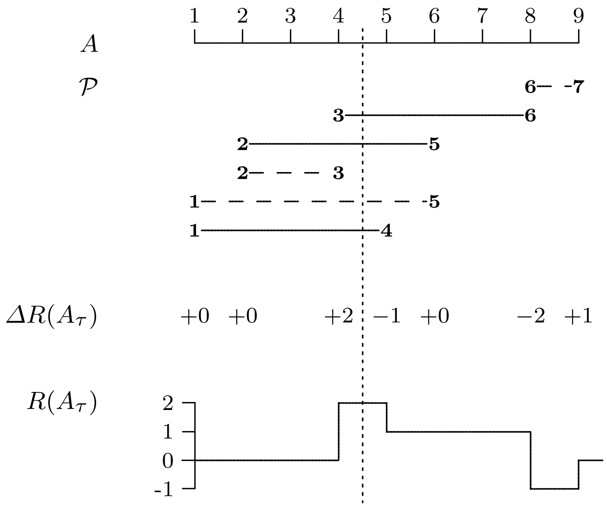

Fig. 1 Pairs from $\mathcal{P}$, changes of ReliefF, $\Delta R(A_\tau)$, and ReliefF, $R(A_\tau)$, both as functions of threshold τ. The vertical line represents a threshold at $\tau = 4.5$

table, so $\mathcal{P} = \{(1,4), (1,5), (2,3), (2,5), (6,3), (6,7)\}$. According to (1), all punishments and rewards are $-1/3$ and $1/3$, respectively; for sake of simplicity we will use -1 and 1 instead.

2.1 Threshold search

Let A be a continuous attribute, and let A_τ be a binary attribute derived from A so that $A_\tau = 0$ when $A \leq \tau$ and $A_\tau = 1$ when $A > \tau$. We need to find the value of τ which yields A_τ with the highest ReliefF.

The mental scaffolding behind the algorithm is illustrated in Fig. 1. The upper part shows example pairs from $\mathcal{P}$ represented by their indices (column *id* in Table 1) aligned with the corresponding values of A. Pairs from different classes (which contribute $f(E,F) = +1$ if $E_{A_\tau} \neq F_{A_\tau}$) are connected with solid lines and pairs from the same class ($f(E,F) = -1$) with dashed lines.[1] If we take, for instance, $\tau = 4.5$ (the dashed vertical line), the derived attribute A_τ has value 0 on examples 1 through 3, and value 1 on examples 4 through 7 (column A_τ in Table 1). Two pairs, (2, 3) and (6, 7), have the same value of A_τ and do not contribute to $R(A_\tau)$, according to (2). Pairs (3, 6), (2, 5) and (1, 4) give positive contributions since they are from different classes, and (1, 5) gives a negative contribution since examples 1 and 5 are from the same class, so $R(A_\tau) = 3 - 1 = 2$. The optimal threshold is the one which crosses as many solid lines and as few dashed lines as possible. In our case the optimum is at $4 \leq \tau < 5$.

$\Delta R(A_\tau)$ in Fig. 1 shows how $R(A_\tau)$ changes as τ goes from the minimal to the maximal value of A. Changes only occur where lines begin and end. Let $\mathcal{B}_\tau$ and $\mathcal{E}_\tau$ be sets of example pairs whose corresponding lines in Fig. 1 begin and end at τ, respectively:

$$\mathcal{B}_\tau = \{(E,F) \in \mathcal{P} : \min(E_A, F_A) = \tau\}, \tag{3}$$

$$\mathcal{E}_\tau = \{(E,F) \in \mathcal{P} : \max(E_A, F_A) = \tau\}. \tag{4}$$

[1] Note again that the actual contributions are $\frac{1}{3}$ and $-\frac{1}{3}$; in a non-binary classification problem lines would have different "weights" corresponding to $f(E,F)$.

Algorithm 1 Fast threshold search

```
*Input:* Attribute $A$, set of example pairs $\mathcal{P}$ and their potential contributions $f(E,F)$
*Output:* Stepwise ReliefF for $A_\tau$ tabulated at all jumps

$\Delta R \leftarrow$ empty ordered dictionary
**for** each $(E,F) \in \mathcal{P}$ **do**
    increase $\Delta R[\min(E_A, F_A)]$ by $f(E,F)$
    decrease $\Delta R[\max(E_A, F_A)]$ by $f(E,F)$
**end for**

$s \leftarrow 0$
$R \leftarrow$ empty dictionary
**for** each consecutive key $\tau$ in dictionary $\Delta R$ **do**
    $s \leftarrow s + \Delta R[\tau]$
    $R[\tau] \leftarrow s$
**end for**
**return** $R$
```

We can compute $\Delta R(A_\tau)$ and $R(A_\tau)$ as

$$\Delta R(A_\tau) = \sum_{(E,F)\in\mathcal{B}_\tau} f(E,F) - \sum_{(E,F)\in\mathcal{E}_\tau} f(E,F) \tag{5}$$

and

$$R(A_\tau) = \sum_{t\le\tau} \Delta R(A_t). \tag{6}$$

The first sum in (5) corresponds to contributions of pairs which are split (*i.e.* have different values of A_τ) at this threshold but not at a smaller one. Similarly, the second sum corresponds to contributions of pairs which would be split by a smaller threshold but not by this one. In a special case where $E_A = F_A$, the pair occurs in both sums at the same τ and its contribution cancels out.

Algorithm 1 first computes the value of $\Delta R(A_\tau)$ at all values of A appearing in the chosen example pairs $\mathcal{P}$, according to (5). A suitable structure for storing this data is an ordered dictionary implemented as a tree. Values of $R(A_\tau)$ are then computed as partial sums of $\Delta R(A_\tau)$. If we are interested only in the optimal split, we do not need to store the entire $R(A_\tau)$ but only record its maximum and the corresponding τ.

2.2 Searching for subsets of discrete values

Let B be a discrete attribute with domain $V = \{v_1, v_2, \ldots, v_n\}$. Let $\mathcal{V} = \{V_1, V_2, \ldots, V_l\}$ be a partitioning of V, and let $B_\mathcal{V}$ be a derived attribute with domain $\mathcal{V}$, so that $E_{B_\mathcal{V}} = V_j$ when $E_B \in V_j$. We need to find the partitioning $\mathcal{V}$ which yields $B_\mathcal{V}$ with the highest ReliefF.

Figure 2(a) shows a graph whose vertices correspond to values of attribute B from our running example. Pairs from $\mathcal{P}$ are represented by edges between the corresponding values of B; *e.g.* the values of B for examples 1 and 4 are a and b, so pair (1, 4) is represented by an edge between a and b. Solid lines represent example pairs with positive contributions and dashed lines represent those with negative contributions.[2] In the graph on Fig. 2(b)

[2]We again simplify the example by assuming $f(E,F) = \pm 1$.

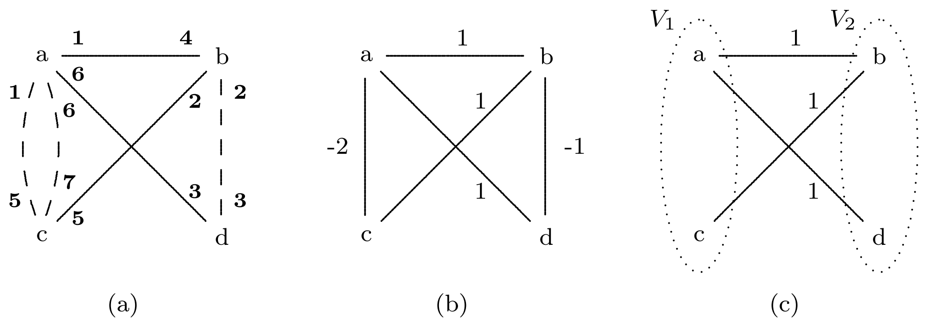

Fig. 2 Contributions to RelieF in terms of example pairs (**a**) as in (2), pairs of values (**b**) as in (8), pairs of values of $B_{\mathcal{V}}$ (**c**) as in (10)

we merged multiple edges between the same vertices and added their contributions. B is irrelevant since $R(B) = -2 + 1 + 1 + 1 - 1 = 0$.

Now consider an attribute $B_{\mathcal{V}}$ defined by $V_1 = \{a, c\}$, $V_2 = \{b, d\}$ (Fig. 2(c) and the right column of Table 1). The difference between $R(B)$ and $R(B_{\mathcal{V}})$ is that the latter does not include the contributions of pairs $a - c$ and $b - d$ since the corresponding pairs of examples—(1, 5) and (6, 7) for $a - c$, and (2, 3) for $b - d$—have the same values of $B_{\mathcal{V}}$. Therefore, $R(B_{\mathcal{V}}) = 1 + 1 + 1 = 3$.

Formally, let $\mathcal{C}_{v_i,v_j}$ be a subset of example pairs from $\mathcal{P}$ such that B equals v_i on one example and v_j on the other,

$$\mathcal{C}_{v_i,v_j} = \{(E, F) \in \mathcal{P} : (E_B = v_i \wedge F_B = v_j) \vee (E_B = v_j \wedge F_B = v_i)\}. \tag{7}$$

For example, $\mathcal{C}_{a,c} = \{(1, 5), (6, 7)\}$. Contributions of pairs of values (weights of edges in Fig. 2(b)) to ReliefF are

$$C(v_i, v_j) = \sum_{(E,F)\in\mathcal{C}_{v_i,v_j}} f(E, F). \tag{8}$$

Since $\mathcal{P}$ is the union of $\mathcal{C}_{v_i,v_j}$ over all v_i and v_j, we have, from (2) and (8)

$$R(B) = \sum_{v_i,v_j\in V} C(v_i, v_j). \tag{9}$$

We have expressed ReliefF as the sum of contributions of value pairs. Their computation is straightforward (Algorithm 2). We shall continue by expressing $R(B_{\mathcal{V}})$ with these same contributions. We have, by (8),

$$C(V_i, V_j) = \sum_{(E,F)\in\mathcal{C}_{V_i,V_j}} f(E, F) = \sum_{v_k\in V_i, v_l\in V_j} C(v_k, v_l). \tag{10}$$

By comparing (9) and (10) we see that the ReliefF of $B_{\mathcal{V}}$ equals that of B without the contributions of pairs of B's values which are merged into the same value of $B_{\mathcal{V}}$,

$$R(B_{\mathcal{V}}) = R(B) - \sum_{V_i\in\mathcal{V}} \sum_{v_k,v_l\in V_i} C(v_k, v_l). \tag{11}$$

Algorithm 2 Computation of value-pairs contributions to ReliefF

```
Input: Attribute $B$, set of example pairs $\mathcal{P}$ and their potential contributions $f(E, F)$
Output: Contributions of all value pairs to ReliefF

$C \leftarrow |V| \times |V|$ matrix initialized to 0
for each $(E, F) \in \mathcal{P}$ do
  increase $C[E_B, F_B]$ by $f(E, F)$
end for

for $i = 1$ to $|V|$ do
  for $j = i + 1$ to $|V|$ do
    $C[i, j] \leftarrow C[i, j] + C[j, i]$
    $C[j, i] \leftarrow 0$
  end for
end for
return $C$
```

	{a}	{b}	{c}	{d}
{a}	0			
{b}	1	0		
{c}	−2	1	0	
{d}	1	−1	0	0

(a) Initial contributions; $R(B_{\mathcal{V}}) = 0$

	{a, c}	{b}	{d}
{a, c}	0		
{b}	2	0	
{d}	1	−1	0

(b) After merging *a* and *c*; $R(B_{\mathcal{V}}) = 2$

	{a, c}	{b, d}
{a, c}	0	
{b, d}	3	0

(c) After merging *b* and *d*; $R(B_{\mathcal{V}}) = 3$

Fig. 3 Contributions $C(V_i, V_j)$ at each step of heuristic optimization

A possible heuristic approach to finding the partition which gives the maximal $R(B_{\mathcal{V}})$ is similar to Kramer's (1994) algorithm for binarization of attributes based on impurity measures. We start with $V_i = \{v_i\}$, so $B_{\mathcal{V}}$ is equivalent to B and $R(B_{\mathcal{V}}) = R(B)$. Then we iteratively merge the pair with the most negative contribution $C(V_i, V_j)$, increase the ReliefF of $B_{\mathcal{V}}$ by $C(V_i, V_j)$ and update the contributions for other pairs,

$$C(V_i \cup V_j, V_k) = C(V_i, V_k) + C(V_j, V_k), \tag{12}$$

as follows from (10). We stop when all the remaining contributions are positive or when there is only a single value left, which effectively removes the attribute. In our case, the algorithm would first merge a and c, and then b and d, arriving at the optimal partitioning $\mathcal{V} = \{\{a, c\}, \{b, d\}\}$ (Fig. 3).

When the number of values of B is reasonably small (modern computers can easily handle 15–20 values), we can perform an exhaustive search across all possible partitions. The algorithm starts with one subset containing all values and the other subset empty. The ReliefF of the single-valued $B_{\mathcal{V}}$ is 0. Then it moves one value at a time from one subset to the other and recomputes the ReliefF. As obvious from (11), moving v_k from V_i to V_j changes ReliefF by

$$\Delta R(B_{\mathcal{V}}) = \sum_{u \in V_i} C(v_k, u) - \sum_{u \in V_j} C(v_k, u). \tag{13}$$

V_1	V_2	$R(B_{\mathcal{V}})$	Change	$\Delta R(B_{\mathcal{V}})$
{a, b, c, d}	{ }	0		
			d: $V_1 \to V_2$	$(1 + -1 + 0) - () = 0$
{a, b, c}	{d}	0		
			c: $V_1 \to V_2$	$(-2 + 1) - (0) = -1$
{a, b}	{c, d}	-1		
			d: $V_2 \to V_1$	$(0) - (1 + -1) = 0$
{a, b, d}	{c}	-1		
			b: $V_1 \to V_2$	$(1 + -1) - (1) = -1$
{a, d}	{b, c}	-2		
			d: $V_1 \to V_2$	$(1) - (-1 + 0) = +2$
{a}	{b, c, d}	0		
			c: $V_2 \to V_1$	$(1 + 0) - (-2) = +3$
{a, c}	{b, d}	3		
			d: $V_2 \to V_1$	$(-1) - (1 + 0) = -2$
{a, c, d}	{b}	1		

Fig. 4 Computation of ReliefF for all binarizations of B with exhaustive search

Moving only a single value at a time and still making only $2^{|V|-1}$ steps to explore all binary splits is possible by ordering the subsets by the Gray code. The computation for our example is shown in Fig. 4.

2.3 Time complexity

Let there be N examples from c classes described by A attributes. Let m and k be the number of reference examples and neighbors from each class, respectively. Šikonja and Kononenko (2003) showed that constructing $\mathcal{P}$ can be done in $O(mNA)$ time, assuming that $O(mkc \ln N)$ is less than $O(mNA)$.

In addition to that, the threshold search (Algorithm 1) stores mkc values into a dictionary whose size is at most the number examples, N, which takes $O(mkc \ln N)$ time. Construction of value-pairs contributions (Algorithm 2) requires adding the contributions of mkc pairs of examples to the matrix, hence the complexity is $O(mkc)$. Under the above assumption, the asymptotic time complexity of both algorithms is dominated by that of the basic ReliefF.

The time complexity of merging the attribute's values after constructing the matrix depends upon the chosen algorithm. The heuristic algorithm is essentially an agglomerative clustering with a specific linkage and stopping condition, so its time complexity is $O(|V|^2 \ln |V|)$ (Day and Edelsbrunner 1984). Exhaustive search takes $2^{|V|-1}$ steps with $|V|$ additions (13), hence its complexity is $O(|V|e^{|V|})$. Discrete attributes rarely have more than 10 values and hence at most 512 binary partitions, which can be scored instantly by modern computers.

Practical implementations of ReliefF evaluate all attributes at once to avoid the costly construction of $\mathcal{P}$ for each attribute separately. The same technique can be applied here. In practice, there is no noticeable difference between the running time of pure ReliefF and additionally executing the algorithms described in this paper.

3 Conclusion

The described algorithms solve the basic problem prohibiting a wider use of Relief for tasks such as induction of classification trees and rules, discretization of continuous attributes and feature construction. Although we derived them for ReliefF, they are easily adaptable to other variations of Relief, for instance RRelief (Šikonja and Kononenko 2003), for as long as the basic premises about pair contributions hold, such as that the example pair with the same value of the attribute does not contribute to Relief.

Acknowledgement This work was supported by grants from the Slovenian Research Agency (J2-2194, P2-0209).

References

Day, W. H., & Edelsbrunner, H. (1984). Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of Classification*, *1*(1), 7–24.

Jakulin, A. (2005) Machine learning based on attribute interactions. PhD thesis, University of Ljubljana. http://eprints.fri.uni-lj.si/205/

Kira, K., & Rendell, L. A. (1992). The feature selection problem: Traditional methods and a new algorithm. In *AAAI* (pp. 129–134). Cambridge: AAAI Press/MIT Press.

Kononenko, I. (1994). Estimating attributes: Analysis and extensions of Relief. In Bergadano, F., & Raedt, L. D. (Eds.) *Lecture notes in computer science: Vol. 784. ECML* (pp. 171–182). Berlin: Springer.

Kononenko, I., & Šikonja, M. R. (2007). Non-myopic feature quality evaluation with (R)ReliefF. In Liu, H., & Motoda, H. (Eds.) *Computational methods of feature selection*. Boca Raton: Chapman & Hall/CRC.

Kramer, S. (1994) CN2-MCI: A two-step method for constructive induction. In: *Proc. ML-COLT '94 workshop on constructive induction and change of representation*, New Brunswick, New Jersey.

Šikonja, M. R., & Kononenko, I. (2003). Theoretical and empirical analysis of ReliefF and RReliefF. *Machine Learning*, *53*(1–2), 23–69.