Context-Sensitive Feature Selection for Lazy Learners

Pedro Domingos
Department of Information and Computer Science
University of California, Irvine
Irvine, California 92697, U.S.A.
pedrod@ics.uci.edu

Abstract

High sensitivity to irrelevant features is arguably the main shortcoming of simple lazy learners. In response to it, many feature selection methods have been proposed, including forward sequential selection (FSS) and backward sequential selection (BSS). Although they often produce substantial improvements in accuracy, these methods select the same set of relevant features everywhere in the instance space, and thus represent only a partial solution to the problem. In general, some features will be relevant only in some parts of the space; deleting them may hurt accuracy in those parts, but selecting them will have the same effect in parts where they are irrelevant. This article introduces RC, a new feature selection algorithm that uses a clustering-like approach to select sets of locally relevant features (i.e., the features it selects may vary from one instance to another). Experiments in a large number of domains from the UCI repository show that RC almost always improves accuracy with respect to FSS and BSS, often with high significance. A study using artificial domains confirms the hypothesis that this difference in performance is due to RC's context sensitivity, and also suggests conditions where this sensitivity will and will not be an advantage. Another feature of RC is that it is faster than FSS and BSS, often by an order of magnitude or more.

Keywords: Lazy learning, feature selection, nearest neighbor, induction, machine learning

1 Introduction and motivation

Induction is the art and science of generalizing from the known to the unknown, so that appropriate responses to the unknown can be formulated when it appears. Classification is an example of an induction task relevant in a wide variety of domains. Given a set of preclassified examples, each typically represented by a vector of features, inductive learning algorithms attempt to produce class descriptions that will be accurate for new examples. The class assigned to a new example can then be used to decide how to process it (e.g., if the classification task is to diagnose an illness, the diagnosis is used to decide on the treatment). *Eager* approaches to induction explicitly produce generalizations representing the classes under study, often in a language different from that used to represent the examples. *Lazy* approaches, in contrast, delay this generalization until classification time; it is performed implicitly when a new example is compared to the stored instances and the class of the nearest one(s) is assigned to it. Lazy learners, also known as instance-based (Aha, 1991), memory-based (Stanfill & Waltz, 1986), exemplar-based (Salzberg, 1991), case-based (Kolodner, 1993) and others, have several advantages when compared to eager methods like decision-tree (Quinlan, 1993) and rule induction (Clark & Niblett, 1989). They are conceptually simple, and yet able to form complex decision boundaries in the instance space even when relatively little information is available. They combine naturally with analogical reasoning, apply easily to numeric domains, and with appropriate distance measures can also outperform other approaches in symbolic ones (Cost & Salzberg, 1993). Special cases that may be missed by abstraction-forming approaches can be retained and recognized. Learning is often simple to perform, because it involves mainly storing the examples, possibly with some selection and indexing.

Lazy learners have some shortcomings, however. The memory cost of the class descriptions they produce is typically greater, and they can be harder for a human to understand. Classification can also take longer, even with suitable indexing schemes. However, the most significant problem for lazy learning is arguably that posed by irrelevant features. If many such features are present in the example descriptions, lazy learners will be confused by them when they compare examples, resulting in a possibly severe degradation of accuracy. A natural solution to this problem is identifying the irrelevant features, and discarding them before storing the examples for future use. Several algorithms have been proposed for this purpose (see (Kittler, 1986) for a survey), of which two of the most widely known are forward sequential search (FSS) and backward sequential search (BSS) (Devijver & Kittler, 1982). Many variations of these exist (e.g., (Aha & Bankert, 1994)). Their use can have a large positive impact on accuracy.

However, all of these algorithms have the common characteristic that they ignore the fact that some features may be relevant only in context (i.e., given the values of other features). They may discard features that are highly relevant in a restricted sector of the instance space because this relevance is swamped by their irrelevance everywhere else. They may retain features that are relevant in most of the space, but unnecessarily confuse the classifier in some regions.

Consider, for example, an instance space defined by a set of numeric features $\mathbf{F}$, and a class composed of two hyperrectangles, one of which is defined by intervals $f_i \in [a_i, b_i]$ in a subset $\mathbf{F_1}$ of the features, and the other by intervals in a subset $\mathbf{F_2}$ disjoint from the first.

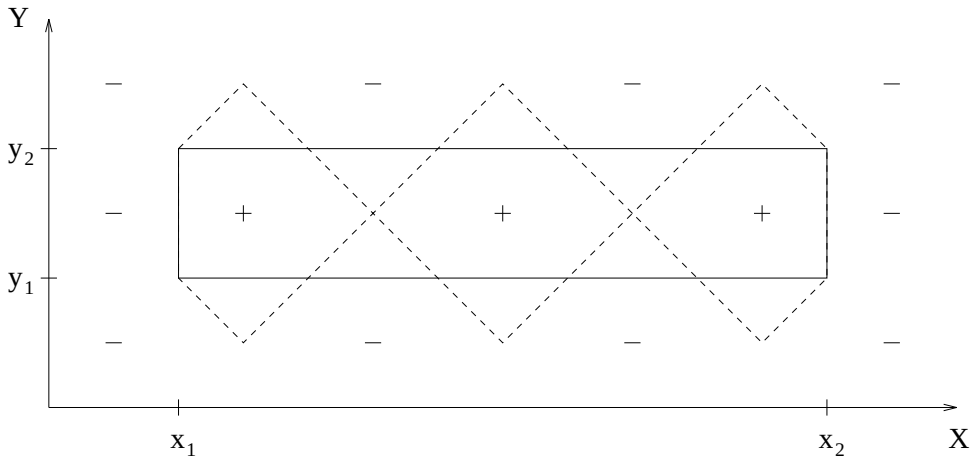

Figure 1: A concept for context-sensitive feature selection.

Current feature selection algorithms would retain all features in $\mathbf{F_1}$ and $\mathbf{F_2}$, because each of those features is relevant to identifying examples in one of the hyperrectangles. However, the features in $\mathbf{F_2}$ act as noise when identifying examples defined by $\mathbf{F_1}$, and vice-versa. Instead of storing the same set of features for all instances, a better algorithm would discard the features in $\mathbf{F_2}$ from the stored instances of the first hyperrectangle, and the features in $\mathbf{F_1}$ from those of the second one. This article describes an algorithm that does this.

As another example, consider Figure 1, where the concept to be learned is the rectangle delimited by the solid line, and + and – indicate the positive and negative examples in the training set, respectively. The basic one-nearest-neighbor algorithm would produce the boundary shown as a dashed line, resulting in a large error. A context-free feature selection algorithm would retain both features, producing the same boundary, or delete one of them, collapsing the plane to a line and resulting in an even greater error. A context-sensitive algorithm, on the other hand, would ignore feature X when $Y > y_2$ or $Y < y_1$, because in those areas all examples are negative irrespective of the X coordinate, and it would take X into consideration when $y_1 \leq Y \leq y_2$, because here examples are positive if $x_1 \leq X \leq x_2$, and negative otherwise. X is thus relevant or not depending on the context (i.e., on the value of Y), and recognizing this leads to the correct boundary being induced.

The next section describes the lazy learner used in the studies reported here. Next, the FSS and BSS algorithms are summarized. RC, a context-sensitive feature selection algorithm, is then introduced. The worst-case time complexities of FSS, BSS and RC are compared. An empirical study follows, using datasets from the UCI repository and purposely-constructed ones. Finally RC's relationship to other learning algorithms is discussed, directions for future research are suggested, and some conclusions are drawn.

2 A lazy learner

The study described in this article was carried out in the context of RISE, a multistrategy learning system with a lazy-learning component (Domingos, 1995; Domingos, 1996). In this section we describe the lazy component (LazyRISE), providing the groundwork for the

descriptions of the feature selection algorithms to follow.

LazyRISE inputs a *training set* of preclassified *examples*. An example is represented by a vector of *features* and a corresponding *class*. Features can be *symbolic* (nominal, categorical, non-ordinal discrete) or *numeric* (linear, real-valued). In LazyRISE the class is always symbolic. *Test examples* are examples not in the training set and submitted for classification. A distinction is made between examples and *instances*. The former are the input data, and the latter comprise the class description (i.e., instances are the stored examples used to classify new ones). The two may be syntactically identical, but differ semantically: an example represents one point in the instance space, whereas an instance represents the entire region of the space that is closer to it than to any other instance, and therefore inherits its class. Additionally, an instance will differ syntactically from the corresponding example if some features were deemed irrelevant and dropped from it. An instance is said to *win* an example when the example is closer to it than to any other stored instances, according to the distance measure used.

Choosing the distance measure is one of the major design decisions when building a lazy learner. LazyRISE employs normalized Euclidean distance for numeric features, and a simplified version of Stanfill and Waltz's (1986) value difference metric for symbolic ones. Let $E = (e_1, e_2, \ldots, e_F, c_E)$ be an example with value e_i for the ith feature and class c_E, and let $I = (i_1, i_2, \ldots, i_F, c_I)$ be an instance, with similar notation. In instances only, a feature can have the special value '$*$', which means that the feature was found to be irrelevant and discarded. The distance $\Delta(I, E)$ between I and E is then defined as:

$$\Delta(I, E) = \sum_{j=1}^{F} \delta^2(i_j, e_j) \tag{1}$$

where the component distance $\delta(i_j, e_j)$ for the jth feature is:

$$\delta(i_j, e_j) = \begin{cases} 0 & \text{if } i_j = * \\ \delta_{num}(i_j, e_j) & \text{if } i \text{ is numeric and } i_j \neq * \\ SVDM(i_j, e_j) & \text{if } i \text{ is symbolic and } i_j \neq * \end{cases} \tag{2}$$

The component distance for numeric features is defined as:

$$\delta_{num}(i_j, e_j) = \left| \frac{i_j - e_j}{max_j - min_j} \right| \tag{3}$$

max_j and min_j being respectively the maximum and minimum values of the feature found in the training set. $SVDM(i_j, e_j)$ is the simplified value difference metric, defined as:

$$SVDM(x_i, x_j) = \sum_{h=1}^{C} |P(c_h|x_i) - P(c_h|x_j)| \tag{4}$$

where x_i and x_j are any legal values of the feature, C is the number of classes, c_h is the hth class, and $P(c_h|x_i)$ denotes the probability of c_h conditioned on x_i. SVDM is a more sophisticated alternative to the commonly-used *overlap* metric (i.e., 0 if $i = j$, and 1 otherwise). $SVDM(x_i, x_j)$ is still always 0 if $i = j$, but it can be less than 1 for two different feature values if they correlate similarly with the class variable. Thus SVDM incorporates into lazy learning some of the information used by eager and Bayesian classifiers. Versions

of this measure have been found to produce large improvements in accuracy compared to overlap in some symbolic domains (Cost & Salzberg, 1993). SVDM differs from Cost and Salzberg's MVDM in that the latter also includes weights for the instances being compared. Of particular interest is the fact that VDM-type metrics have some ability to mute globally irrelevant features, because they will tend to always be close to zero for them. Thus SVDM eases the task of the feature selection algorithms described in the next two sections, when selecting symbolic features.

The distance from a missing numeric value to any other value is defined as 0. This is equivalent to ignoring a feature when its value is unknown. If a symbolic feature's value is missing, it is assigned the special value "?". This is treated as a legitimate symbolic value, and its SVDM to all other values of the feature is computed and used. In the context of VDM-style metrics, this is a sensible policy: a missing value is taken to be roughly equivalent to a given possible value if it behaves similarly to it, and inversely if it does not.

In its simplest form, LazyRISE simply stores all the training examples as instances, and estimates their accuracy as classifiers using a leave-one-out methodology: each example is removed in turn from the training set, and given to the other instances to classify; the nearest instance is found, and its class assigned to the example. The *accuracy* of an instance is the fraction of the examples it won that were indeed of its class. Because this tends to over-estimate the accuracy of instances that win very few examples, the *Laplace-corrected* accuracy (Niblett, 1987) is used instead:

$$LAcc(I) = \frac{N_{corr}(I) + 1}{N_{won}(I) + C} \tag{5}$$

where I is any instance, C is the number of classes, $N_{won}(I)$ is the total number or examples won by I, and $N_{corr}(I)$ is the number of examples that I wins and correctly classifies.

At classification time, LazyRISE compares the new example with each of the stored instances using the distance measure defined in Equation 1, and assigns the example to the nearest instance's class. The previously-computed accuracies are used to choose between instances that are equally close to the test example. The effect of the Laplace correction is to make the estimate of an instance's accuracy converge to the "random guess" value of $1/C$ as the number of examples won by the instance decreases. Thus instances with high apparent accuracy are favored only if they also have high statistical support (i.e., if that apparent accuracy is not simply the result of a small sample). When the closest instances to the test example have the same Laplace accuracy, the most frequent class is chosen, and if the classes are equally frequent, a winner is selected at random.

3 Context-free feature selection

The forward sequential selection algorithm (FSS) starts with an empty feature set and repeatedly adds the "best" feature to it until no further improvement is possible, or all features have been included. The backward sequential selection algorithm (BSS) operates similarly, but starts with the full feature set and repeatedly removes the "worst" feature from it. The implementation of the two algorithms used here is described in pseudo-code in Figures 2

Input: FS is the set of features used to describe examples.

Procedure FSS (FS)

Let $SS = \emptyset$.
Let $BestEval = 0$.
Repeat
  Let $BestF = None$.
  For each feature F in FS and not in SS
    Let $SS' = SS \cup \{F\}$.
    If $Eval(SS') > BestEval$
      Then Let $BestF = F$,
        Let $BestEval = Eval(SS')$.
  If $BestF \neq None$
    Then Let $SS = SS \cup \{BestF\}$.
Until $BestF = None$ or $SS = FS$.
Return SS.

Figure 2: The forward sequential selection (FSS) algorithm.

and 3. In both cases, the final feature set can be empty and all examples assigned to the default class, if this leads to the highest accuracy.

The evaluation function $Eval()$ can be a heuristic measure, typically of the quality of the class separation produced by the feature set, or it can be the actual accuracy obtained by applying the classifier using those features. If $Eval()$ is a heuristic measure, the feature selection algorithm acts as a filter, extracting features to be used later by the main algorithm; if it is the actual accuracy, it acts as a wrapper around that algorithm (John, Kohavi & Pfleger, 1994). The wrapper strategy has been found to often yield the best results (Aha & Bankert, 1994), and this is attributable to the fact that its learning bias is that of the classifier itself, avoiding a possible mismatch between the feature selection and classification biases. Therefore the evaluation function used here is the classifier's accuracy on the training set; it is measured using a leave-one-out methodology (i.e., by removing each example from the training set in turn and using the remaining instances to classify it).

4 Context-sensitive feature selection

This section describes RC (Relevance in Context), a context-sensitive feature selection algorithm. RC is in many ways similar to BSS, but with the crucial difference that it makes local, instance-specific decisions on feature relevance, as opposed to global ones. This is done in a clustering-like fashion: each instance looks for the nearest example of the same class, and hypothesizes that the features along which they differ are irrelevant. This is tested by checking whether deleting those features from the instance has a positive or negative effect

Input: FS is the set of features used to describe examples.

Procedure BSS (FS)

Let $SS = FS$.
Let $BestEval = Eval(SS)$.
Repeat
  Let $WorstF = None$.
  For each feature F in SS
    Let $SS' = SS - \{F\}$.
    If $Eval(SS') \geq BestEval$
      Then Let $WorstF = F$,
        Let $BestEval = Eval(SS')$.
  If $WorstF \neq None$
    Then Let $SS = SS - \{WorstF\}$.
Until $WorstF = None$ or $SS = \emptyset$.
Return SS.

Figure 3: The backward sequential selection (BSS) algorithm.

on the accuracy of the classifier (i.e., a wrapper strategy is used as before). If the effect is positive or null, the features are effectively deleted, and on the following cycle the newly-simplified instance will look for the nearest example of its class that it does not yet cover, and repeat the process. If the effect is negative, the features are retained, and no more feature selection is attempted for this instance. This process is carried out in parallel for all instances, and terminates when feature selection has been attempted with negative results for all instances. Notice that duplicate instances may be produced, but are not removed, in keeping with the idea that only feature selection is being performed. The algorithm is summarized in pseudo-code in Figure 4.

A question that arises in this framework is: when should two numeric values be considered different? If two real feature values are similar but not identical, the fact that they differ should obviously not be construed as evidence that the feature is irrelevant. Thus it is necessary to decide where the critical point should be. The policy adopted was to compute the mean and standard deviation of each numeric feature from the sample in the training set, and attempt dropping the feature only when the values for the instance and the example differ by more than one standard deviation.

To understand this choice, suppose that the observed values of a feature fall into two or three clusters. Given any two values, if they differ by less than one standard deviation they are most likely to be in the same cluster, and if they differ by more they are probably in different ones. Thus values in different clusters are judged to be significantly different, and only those. If there is a large number of clusters, the critical value should be less than one standard deviation. The choice made thus reflects a bias towards a low number of clusters;

Input: TS is the training set.

Procedure RC (TS)

Let IS be TS.
Compute $Acc(IS)$.
Activate all instances in IS.
Repeat
    For each active instance I in IS,
        Find the nearest example E to I at nonzero distance, and of I's class.
        Let $I' = I$ with all features that differ in I and E removed.
        Let $IS' = IS$ with I replaced by I'.
        If $Acc(IS') \geq Acc(IS)$
            Then replace IS by IS',
            Else deactivate I.
Until all instances are inactive.
Return IS.

Figure 4: The RC feature selection algorithm.

essentially, it assumes that the goal is either to distinguish between values above and below a certain threshold (the two-cluster case), or between a central range and values outside it (the three-cluster case). This is typically the case in many practical domains, like medical diagnosis (with variables like body temperature, blood pressure, levels of blood chemicals) and fault detection (voltage, stress, design dimensions with tolerances). In practice, an optimum value for this parameter can be determined by cross-validation, although this was not done in the studies described below.

The accuracy of an instance set $Acc(IS)$ is the fraction of the training examples that it correctly classifies using a leave-one-out methodology, as before. Since the accuracy is being measured on the whole training set, not just the examples won by an instance, the sample size is the same for all accuracies being compared, and no Laplace correction is necessary. If N is the training set size and F is the number of features, applying the leave-one-out methodology directly would result in a time cost of $O(N^2F)$ at each step of the algorithm. Fortunately, after the initial computation of $Acc(IS)$, only *differences* in accuracy due to dropping features from one instance need be computed. If each example's current closest instance and assigned class are stored with it, the change in accuracy can then be computed by matching the changed instance with all examples, and finding the ones it wins that it did not before. Previously misclassified examples that are now correctly classified add to the accuracy, and previously correctly classified examples that are now misclassified subtract from it. If the former are more numerous than the latter, the change in accuracy is positive, and the features are dropped. This reduces each step's cost to $O(NF)$.

5 Time complexity

In this section we show that the worst-case time complexity of RC is similar to that of an efficient implementation of FSS and BSS.

Let N be the training set size, as before. The basic step of FSS/BSS consists of adding/deleting a single feature and checking the results. Since this involves comparing all instances with all examples along $O(F)$ features, the cost of each such step is $O(N^2F)$. This step is repeated for all currently excluded/included features and the best one selected, which means that an $O(N^2F)$ step is repeated $O(F)$ times, resulting in a cost of $O(N^2F^2)$. Since in the worst case all features will be added/dropped, this cycle can be performed $O(F)$ times, resulting in a total cost of $O(N^2F^3)$.

However, this direct implementation of FSS and BSS is inefficient, because it unnecessarily repeats the computation of all distances along all features every time a feature is tentatively added or removed. A more efficient version will cache, for each example, the distances $\Delta(I, E)$ (Equation 1) of all instances to the example, and then, when considering adding or dropping a feature, add or subtract to each $\Delta(I, E)$ the distance component $\delta^2(i_j, e_j)$ along that feature. Once the example's predicted class is found and compared with the correct one, the original distance vector is reinstated, and the process repeats with the next feature. This implementation reduces the worst-case time complexity of FSS and BSS to $O(N^2F^2)$, and is the one used in the studies that follow. Note that it does not require $O(N^2)$ memory instead of $O(N)$, because only the distances for one example at a time are cached. This implies bringing the cycle that classifies each example outside the cycle that tries each feature (i.e., "for each example, add/delete each feature and classify the example," instead of "for each feature, add/delete the feature and classify each example"). Such a process, opening the "black box" of the classification algorithm and bringing the feature selection algorithm inside it, may not be possible for all lazy learning algorithms.

For RC, the basic step consists of finding an instance's nearest example, dropping the features in which they differ, and testing to see if this has a positive effect on global accuracy. Finding the nearest example to a given instance involves comparing the instance with all examples, and takes $O(NF)$ time. Finding and deleting the common features takes $O(F)$ time. Computing the resulting change in accuracy takes $O(NF)$ time, as seen in the previous section. The total cost of finding features to delete in one instance is therefore $O(NF) + O(F) + O(NF) = O(NF)$. In each "repeat" cycle (see Figure 4) this is performed for all instances, leading to a cost of $O(N^2F)$ per cycle. The "repeat" cycle is performed at worst $O(F)$ times, since for each instance in each cycle at least one feature is dropped, and there are at most $O(F)$ features to drop, or the instance is deactivated and the cycle stops early for that instance. Therefore RC's total time cost is $O(N^2F^2)$, similar to that of the efficient implementation FSS and BSS.

An optimization that is possible in RC, as well as in the $O(N^2F^3)$ implementations of FSS and BSS, is the following. When classifying each example using a leave-one-out methodology, the closest instance to it has to be found. This involves computing the distance of each instance to the example, but that computation needs to be carried out only up to the point where the instance's distance is found to be larger than the previous shortest distance found. Similarly, when RC drops features from an instance and searches for the examples it now wins, its distance to each example needs to be computed only until it becomes larger than

the current winning rule's one.

This optimization does not change the quadratic (or cubic) exponents in the time complexity, but can significantly reduce the average running time of the algorithm. It was therefore implemented in RC. Unfortunately, it is not possible in the $O(N^2F^2)$ versions of FSS and BSS, due to the inversion of the order of cycles previously mentioned.

6 Empirical study: UCI datasets

The central hypothesis of this article is that *RC will produce higher accuracies than FSS and BSS when feature relevance is significantly context-dependent, since it has the ability to select different features for different instances (i.e., to select different features given different values of other features).* On the other hand, when features are either globally relevant or globally irrelevant, RC should have no advantage. Furthermore, if few examples are available or the data is noisy, BSS and FSS should be able to detect the globally irrelevant features more easily than RC. This is due to the fact that they consider dropping a feature in all instances at once, instead of in one at a time, and so produce larger swings in accuracy, that can be detected over statistical fluctuations even when the examples are noisy and/or few.

To investigate empirically the hypothesis that RC's advantage increases with the context dependency of feature relevance, a measure of the latter is required. Unfortunately, in real-world domains the "true" degree of context dependency for a target concept is necessarily unknown. One way to circumvent this problem is to carry out studies in artificial domains, where the context dependency can be predetermined by the experimenter, and this is done in the next section. Another approach is to find an empirical measure that is thought to correlate positively with context dependency. One possibility is to find out how far RC strays from selecting the same features for all instances (i.e, from doing the same as FSS and BSS). More concretely, a possible measure is the average D for all pairs of instances of the number of features selected by RC for one but not the other:

$$D = \frac{2}{N(N-1)} \sum_{i=1}^{N} \sum_{j=1}^{i-1} \sum_{k=1}^{F} d_{ijk} \tag{6}$$

where N is the number of training examples, F is the number of features, and d_{ijk} is 1 if feature k was selected for instance i but not instance j or vice-versa, and 0 otherwise. This *feature difference* measure is necessarily imperfect, since the context dependency effects exhibited by RC may or may not be really present, but it is a legitimate one, in the sense that observing it can falsify the hypothesis that RC is more accurate relative to FSS and BSS when it detects greater context dependency. The core of the study that follows will thus be to correlate the feature difference D with the differential accuracy of RC and the context-free algorithms.

An empirical study was conducted using 24 datasets from the UCI repository (Murphy, 1995). These datasets were chosen so as to provide a wide variety of application areas, sizes, combinations of feature types, and difficulty as measured by the accuracy achieved on them by current algorithms. In this way, any conclusions that are reached can be regarded as having some degree of generality. More precisely, they can be expected to be valid for the population of domains of which the UCI repository is a sample. This population is

Table 1: Datasets used in the empirical study.

Domain[a b]	Code	Exs.	Feats.	Num.	Classes	Missing	Incons.
Audiology	AD	200	69	0	24	291	No
Breast cancer	BC	286	9	4	2	9	Yes
Credit screening	CE	690	15	6	2	67	No
Pima diabetes	DI	768	8	8	2	0	No
Echocardiogram	EC	131	7	6	2	40	Yes
Glass	GL	214	9	9	6	0	No
Heart disease	HD	303	13	6	2	7	No
Hepatitis	HE	155	19	6	2	167	No
Horse colic	HO	300	22	7	2	1605	Yes
Iris	IR	150	4	4	3	0	No
Labor negotiations	LA	57	16	8	2	326	No
Lung cancer	LC	32	56	0	3	5	No
Liver disease	LD	345	6	6	2	0	No
LED	LI	100	7	0	10	0	Yes
Lymphography	LY	148	18	3	4	0	No
Post-operative	PO	90	8	8	3	3	Yes
DNA promoters	PR	106	57	0	2	0	No
Primary tumor	PT	339	17	0	21	225	Yes
Solar flare	SF	323	12	3	6	0	Yes
Sonar	SN	208	60	60	2	0	No
Soybean	SO	47	35	0	4	0	No
Voting records	VO	435	16	0	2	392	No
Wine	WI	178	13	13	3	0	No
Zoology	ZO	101	16	1	7	0	No

[a]BC: Ljubljana dataset; EC: class is 2nd feature, features 1 and 10-13 deleted, example with unknown class deleted; HD: Cleveland dataset, last feature deleted to yield a two-class problem; HO: class is 24th feature, features 3 and 25-28 deleted; LI: 100 examples, seed = 1, 10% noise; PO: pseudo-discretized values converted to numeric; SF: 1st feature used as class; SO: small dataset.

[b]The columns are, in order: name of the domain; 2-letter code used to refer to it in subsequent tables; number of examples; number of features; number of numeric features; number of classes; number of missing values in the entire dataset; and whether or not the dataset includes inconsistent examples (i.e., identical examples with different classes).

certainly not the set of all possible induction problems, but it is certainly a set that includes many relevant real-world ones. The choice was also made to use a large number of domains, with the goal of having enough data points to allow statistically sound conclusions. Table 1 summarizes the characteristics of the datasets used.

Twenty runs were carried out for each domain. In each, the training set was composed of two-thirds of the examples, chosen at random, and the remainder were used as test examples. For each of the three algorithms (RC, FSS and BSS), the accuracy obtained, running time and average number of features selected were recorded, and their averages for the 20 runs computed. Table 2 shows, for each domain, the feature difference D (Equation 6), the average

Table 2: Percentage accuracies of RC, FSS and BSS, and significances of the difference between RC and FSS/BSS.

Domain	Feature diff.	RC	FSS	Signif.	BSS	Signif.
LC	14.2±11.1	47.7±11.0	42.3±10.3	5.0	44.5±17.2	10.0
PR	**8.6±14.3**	**89.1±6.0**	**84.4±8.9**	**5.0**	**84.9±5.6**	**1.0**
HO	8.1± 2.8	80.6±4.0	75.3±6.1	0.5	78.2±3.4	1.0
AD	6.8± 4.1	77.0±4.9	71.2±5.7	0.5	75.4±4.4	2.5
VO	5.6± 3.5	95.7±1.7	89.5±13.1	2.5	94.7±1.6	2.5
LA	5.0± 2.0	91.1±6.9	87.4±6.7	5.0	85.8±9.2	1.0
PT	**4.9± 2.5**	**40.2±5.9**	**30.0±6.0**	**0.5**	**33.3±5.0**	**0.5**
SO	4.4± 2.9	100.0±0.0	94.4±8.8	1.0	95.0±6.3	0.5
HE	4.1± 2.2	77.1±4.8	80.5±5.2	95.0	75.7±3.8	10.0
LY	3.8± 1.8	81.2±5.6	76.5±5.2	0.5	79.1±6.0	5.0
ZO	3.1± 1.9	93.2±3.8	91.0±5.1	5.0	90.3±5.6	0.5
SF	2.4± 1.8	70.6±3.6	68.2±3.0	1.0	68.9±3.6	5.0
LI	**2.1± 1.3**	**61.4±6.2**	**47.1±13.3**	**0.5**	**54.7±7.8**	**0.5**
CE	2.0± 1.2	83.7±1.9	80.9±2.3	0.5	81.2±2.5	0.5
BC	1.6± 1.1	66.2±5.2	66.7±6.7	60.0	66.9±6.1	65.0
HD	1.5± 1.1	76.8±3.5	74.8±5.0	10.0	76.2±2.8	30.0
EC	0.8± 1.0	60.2±6.1	59.4±5.2	40.0	60.3±5.6	50.0
PO	**0.5± 0.5**	**60.8±6.1**	**68.5±5.0**	**99.5**	**68.0±6.9**	**99.5**
SN	0.5± 0.3	81.4±9.1	73.5±11.2	0.5	80.5±8.7	15.0
LD	0.3± 0.6	60.2±3.9	58.4±5.1	15.0	60.0±4.8	45.0
DI	0.2± 0.5	70.5±2.5	69.6±2.9	20.0	69.2±3.3	2.5
GL	0.0± 0.2	69.2±5.0	70.8±8.1	75.0	71.3±7.4	85.0
IR	0.0± 0.0	94.4±2.4	92.6±2.3	0.5	92.9±2.9	2.5
WI	0.0± 0.1	95.1±2.6	94.1±2.8	10.0	94.5±2.1	25.0

accuracy and standard deviation for each algorithm, and the significance of the difference between RC and each of the context-free algorithms using a one-tailed paired t test. The t test is appropriate because the accuracies being compared, being means of random samples, are normally distributed by the central limit theorem (DeGroot, 1986), and the variances are unknown and also being estimated; the one-tailed test is preferred over the two-tailed one because the goal is to determine in each case whether RC is better than the context-free algorithm, not whether the two are simply different. The more sensitive paired test is made possible by, in each run, testing all the algorithms on the same sample. Since 48 individual significance tests are reported, it is possible that some of the differences reported as significant are in fact not so (for example, with 40 tests we can expect 2 non-significant differences to be reported as significant at the 5% level by chance). However, most of the significances are very low, making this effect unlikely. The domains are ordered by decreasing feature difference. Some of the more interesting results are highlighted in boldface.

These results are presented in a more easily comprehended form in Figure 5, which

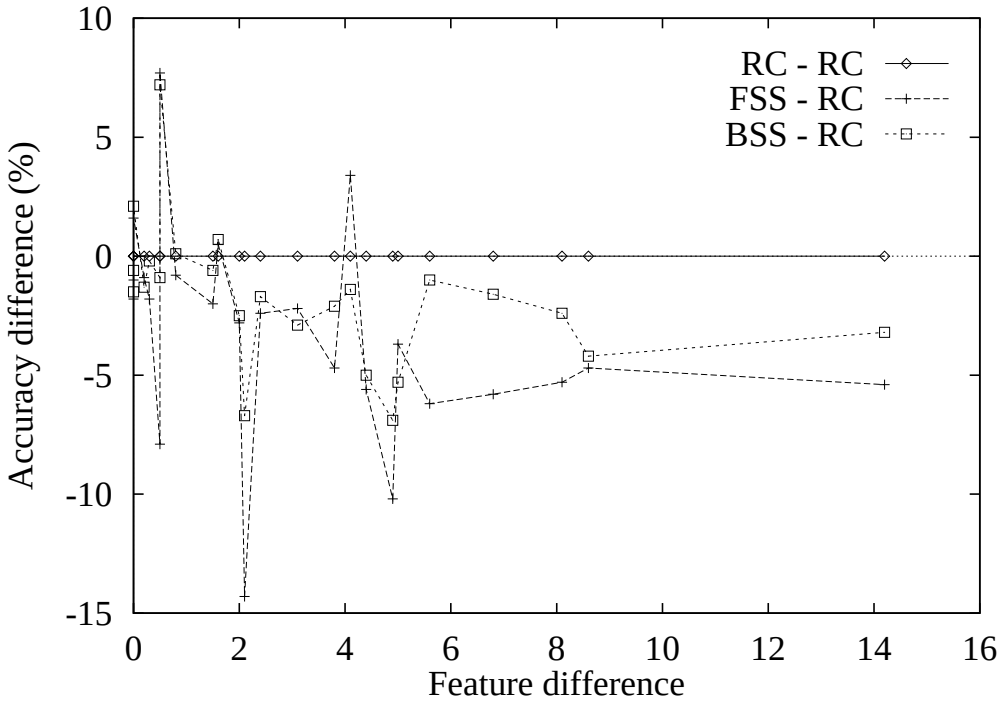

Figure 5: Empirical accuracy as a function of context dependency.

Table 3: Summary of accuracy results.

Measure	FSS	BSS
No. wins	20-4	20-4
No. signif. wins	15-2	14-1
Sign test	0.1	0.1
Wilcoxon test	0.1	0.1

shows the difference in accuracy between each of the algorithms and RC as a function of feature difference. The difference in accuracy between RC and BSS has a significant positive correlation with the feature difference (0.44), and similarly for FSS (0.36). We conclude that RC is indeed able to detect context dependency effects, and from the t test results in Table 2, that taking them into account in feature selection can produce significant improvements in accuracy.

Further analysis of the global results is shown in Table 3, and confirms the conclusion that RC is more accurate than FSS and BSS on this set of domains. The first line shows the number of domains in which RC achieved higher accuracy than the corresponding algorithm vs. the number in which the reverse occurred (e.g., RC was more accurate than BSS in 20 domains and less in 4). The second line shows the number of domains in which RC was more accurate than the other algorithm with a significance level of 5% or less, vs. the number in

which the opposite occurred (i.e., in which the significance is 95% or more). The results are very favorable to RC. The third line shows the results of applying a sign test to the values in line one (i.e., considering the number of times RC won as a binomial variable and asking how likely the results are under the null hypothesis that the two algorithms are equally accurate). This is expressed as a percentage in the table. For example, RC's 20 wins vs. BSS have only a probability of occurrence of 1/1000. This results in very high confidence that RC is a more accurate algorithm than FSS and BSS on the population of domains from which the 24 used are drawn. Line four shows the result of a Wilcoxon signed-ranks test (DeGroot, 1986), a more sensitive procedure that also takes into account the relative magnitudes of the differences observed, though not their absolute values; a large difference in accuracy is considered more significant than a small one. The very small values obtained lend further support to the conclusion that RC is the most accurate algorithm.

It should be remarked that, unlike what is common practice in the machine learning literature, these tests are being performed at the meta level: the question being asked is "Is RC better than the other algorithm on this ensemble of datasets?", as opposed to asking 24 times "Is RC better than the other algorithm on an ensemble of test sets from the same dataset?" Thus the use of a large number of datasets does not undermine the conclusions reached, but instead makes the very high confidences obtained possible.

The gains obtained by using the context-sensitive algorithm, although consistent, are typically moderate (around 2% on average vs. BSS, and 3% vs. FSS). This is consistent with Holte's observation that, for some datasets in the UCI repository, accuracies within a small range of the best recorded values can be obtained using only the single most relevant feature (Holte, 1993). If RC, BSS and FSS all incorporate the "best" features, then their accuracies should not be expected to differ by more than this amount.

The number of features that each algorithm selects on average is also an indication of how the algorithms' behavior differs. It is reported in Table 4. Since RC does not select the same set of features for all instances, its feature usage in each trial is defined as the average for all instances of the number of features used in each instance; for example, if a feature is used in only one of the N instances, it counts as only $1/N$ features. This is then averaged across all 20 trials. The average for all trials of the standard deviation *within each trial* of the number of features selected is also reported. As might be expected, it correlates positively with the feature difference, since a high value implies large variation in the features selected, even though the reverse is not true because two instances may have different features but the same number of features.

BSS always selects more features than FSS; this is not surprising, given their respective search strategies. RC almost always selects the most features, but this observation can be misleading: direct inspection of the simplified instances output by RC shows that it typically drops most features from just a few of the instances, and it is these highly simplified ones that win most of the test examples; the majority of the instances retain most of the features, but have little impact on classification. Inspection also reveals that the most highly simplified instances differ in the features they retain. This, together with RC's higher accuracies, is further evidence that it is indeed detecting context sensitivity effects. However, it is still true that if the goal is to reduce the feature set size as much as possible, even at some cost in accuracy, RC is clearly not the algorithm of choice: not only does it retain a higher number of features on average, but it only allows the removal of features that do not appear in any

Table 4: Average number of features selected by the algorithms, and average feature difference of RC's instances.

Domain	No. feats.	RC	FSS	BSS	Feature diff.
AD	69	64.1± 4.4	11.4	22.0	6.8± 4.1
BC	9	7.7± 1.1	2.3	4.8	1.6± 1.1
CE	15	13.5± 1.2	5.7	9.6	2.0± 1.2
DI	8	7.9± 0.3	1.9	6.6	0.2± 0.5
EC	7	6.6± 0.8	1.5	4.4	0.8± 1.0
GL	9	9.0± 0.1	4.7	5.6	0.0± 0.2
HD	13	12.1± 0.9	4.6	9.3	1.5± 1.1
HE	19	16.4± 2.0	3.5	11.0	4.1± 2.2
HO	22	14.8± 3.4	5.7	15.6	8.1± 2.8
IR	4	4.0± 0.0	2.2	2.6	0.0± 0.0
LA	16	11.3± 2.0	3.0	6.8	5.0± 2.0
LC	56	45.9±11.2	3.2	9.9	14.2±11.1
LD	6	5.9± 0.5	2.1	4.2	0.3± 0.6
LI	7	5.5± 1.2	5.5	6.3	2.1± 1.3
LY	18	15.2± 1.9	5.2	11.3	3.8± 1.8
PO	8	7.7± 0.5	2.0	3.5	0.5± 0.5
PR	57	52.1±11.7	5.4	16.1	8.6±14.3
PT	17	13.5± 2.7	7.5	12.0	4.9± 2.5
SF	12	10.1± 1.9	4.4	5.7	2.4± 1.8
SN	60	59.7± 0.3	5.8	37.2	0.5± 0.3
SO	35	25.2± 1.5	2.0	3.9	4.4± 2.9
VO	16	5.9± 3.5	7.1	9.4	5.6± 3.5
WI	13	13.0± 0.1	4.3	8.4	0.0± 0.1
ZO	16	13.2± 1.8	5.8	6.4	3.1± 1.9

final instance.

The post-operative patient data domain (PO) is an example of a situation where FSS's and BSS's bias is more appropriate than RC's. In this domain, most features appear to be globally irrelevant; simply assigning all test examples to the most frequent class, ignoring all feature information, produces higher accuracy than all three algorithms (and also than decision-tree and rule learners, as was found in a separate study (Domingos, 1995)). FSS and BSS correctly discard most of the features. However, because the dataset is small (90 examples) and noisy (as evinced by the fact that it contains inconsistent examples) RC has difficulty detecting the global irrelevance of features, and retains most of them for most instances.

Another variable of interest is the running time of the algorithms. These are shown in Table 5. A Sun 670 workstation was used for all runs. Figure 6 shows these values plotted on a log-log scale against N^2F^2, the worst-case asymptotic growth rate for all algorithms (derived in the previous section). RC is always faster than FSS and BSS, sometimes by

Table 5: Average running time of algorithms, in minutes, seconds, and hundredths of a second; and ratio of running times of FSS and BSS to running time of RC.

Domain	RC	FSS	BSS	FSS/RC	BSS/RC
AD	0:10.68	4:14.60	11:15.61	23.8	63.3
BC	0:06.03	0:25.78	1:31.95	4.3	15.2
CE	1:42.25	7:58.87	7:44.39	4.7	4.5
DI	1:33.12	3:48.80	3:59.52	2.5	2.6
EC	0:01.21	0:03.40	0:05.53	2.8	4.6
GL	0:03.38	0:25.39	0:24.63	7.5	7.3
HD	0:08.10	1:02.13	1:06.16	7.7	8.2
HE	0:02.81	0:10.97	1:32.94	3.9	33.1
HO	0:10.06	2:54.30	2:26.87	17.3	14.6
IR	0:01.12	0:03.50	0:03.52	3.1	3.1
LA	0:00.56	0:01.76	0:04.26	3.1	7.6
LC	0:00.41	0:01.88	0:12.64	4.6	30.8
LD	0:06.47	0:25.89	1:32.14	4.0	14.2
LI	0:00.52	0:02.63	0:00.93	5.1	1.8
LY	0:02.29	0:18.56	0:21.17	8.1	9.2
PO	0:00.76	0:02.00	0:03.27	2.6	4.3
PR	0:03.40	1:34.18	2:29.97	27.7	44.1
PT	0:09.68	2:54.47	1:14.92	18.0	7.7
SF	0:06.81	1:51.47	1:45.35	16.4	15.5
SN	0:24.32	4:47.99	14:49.84	11.8	36.6
SO	0:00.74	0:01.85	0:11.16	2.5	15.1
VO	1:44.70	3:56.94	3:31.63	2.3	2.0
WI	0:03.86	0:26.00	1:33.19	6.7	24.1
ZO	0:01.10	0:07.77	0:10.17	7.1	9.2

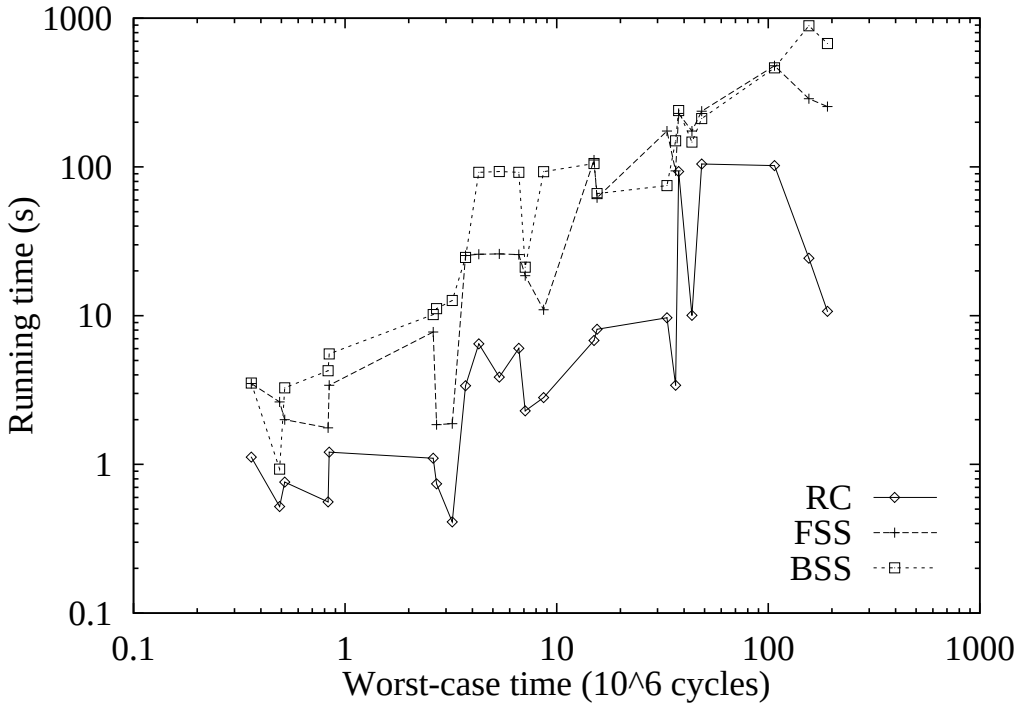

Figure 6: Running time in relation to N^2F^2.

large factors (see, for example, the audiology domain, AD). This can be partly attributed to variations in the number of features that each algorithm actually adds/drops: since RC typically drops fewer features than BSS, and fewer than FSS adds, it finishes in fewer cycles. Another reason for the observed difference in times is that the context-free algorithms have a higher multiplicative coefficient for N^2F^2, and the presence of additional lower-order terms in these. However, the most important factor is the optimization in the distance computation that was used in RC, but is not possible in the efficient versions of FSS and BSS (see previous section).

Considering these effects, and extrapolating from the average slopes of the log-log plots of the algorithms' running times, we are led to hypothesize that RC will still be a viable feature selection algorithm in domains where FSS's and BSS's time cost would exclude them from consideration. In the form used here, none of the algorithms are suitable for very large databases, since they are all necessarily quadratic in N, even in the average case; however, more efficient versions of FSS/BSS-style algorithms exist (Kittler, 1986; Aha & Bankert, 1994), and similar modifications of RC can be envisioned.

7 Empirical study: Artificial domains

The question arises of whether the conclusions formulated in the previous section are generally valid, or the favorable results obtained for RC are specific to the domains used in the study reported in the previous section. In other words, RC's observed benefits might apply

only when the biases represented in the UCI repository are verified, independently of the more general hypothesis that they are due to RC's context sensitivity. Another question is whether the feature difference estimate used effectively corresponds to the context dependency we seek to measure, and thus whether the results obtained are meaningful. These two problems were investigated by carrying out experiments in artificial domains. As argued in (Aha, 1992) and (Schaffer, 1989), more robust and general conclusions will be reached if whole classes of domains are considered, instead of the few individual cases typically used in the machine learning literature. Our hypotheses are that RC is more accurate than FSS and BSS over a broad range of domains ("broad" in the sense that they have no common bias save their context dependency), and that the difference in accuracy increases with the context dependency of feature relevance. In artificial domains, the target concept description is known *a priori*, and, if it is composed of a set of prototypes, the measure of feature difference defined in the previous section applied to that set of prototypes constitutes a suitable measure of context dependency. The empirical study thus proceeded by repeatedly selecting a value of D (Equation 6), generating a large number of domains at random characterized by that value, and observing the resulting accuracies of the three algorithms for that sample of domains.

Two-class problems were considered, with 100 examples in each dataset, described by 32 features. In each domain, each feature was chosen to be numeric or Boolean with equal probability (i.e., the number of numeric features is a binomial variable with expected value $F/2$ and variance $F/4$). Class 1 is defined by ten clusters, and class 0 is the complement of class 1. Each prototype or cluster is defined by a conjunction of conditions on the relevant features. The required value for a Boolean feature is chosen at random, with 0 and 1 being equally probable. Each numeric feature i must fall within a given range $[a_i, b_i]$, with a_i being the smaller of two values chosen from the interval [-1, 1] according to a uniform distribution, and b_i the larger one. A cluster is thus a hyperrectangle in the relevant numeric subspace, and a conjunction of literals in the Boolean one.

The choice of relevant features for each prototype is made at random, but in a way that guarantees that the desired value of D for the set of prototypes is maintained on average. The details of the procedure that does this are described in Appendix A. The feature difference D was varied from 0 to 8, the latter being the maximum value that can be produced given the number of features and prototypes used. Twenty domains were generated for each value of D, and two-thirds of the examples used as the training set. The average accuracy of RC, FSS and BSS on the remaining examples is shown graphically as a function of D in Figure 7.

All differences in accuracy between RC and FSS are significant at the 5% level, as are those between RC and BSS for $D = 1$, 2, 4, 5, and 8. This confirms our hypothesis that RC is more accurate than FSS and BSS over a broad range of domains. We also note that BSS's performance is sometimes quite close to RC's. The smallest difference occurs when $D = 0$, as might be expected, since this situation exactly fits BSS's bias. More generally, due to the small number of training examples used (100), BSS may benefit from its ability to produce larger, more easily detected swings in accuracy when attempting to delete features, as previously hypothesized. Increasing the training set size should increase the distance between RC and BSS, since RC will then have enough data to detect the finer local dependencies that BSS by definition cannot. FSS's and BSS's time limitations have precluded repeating the experiments with a significantly larger number of examples to

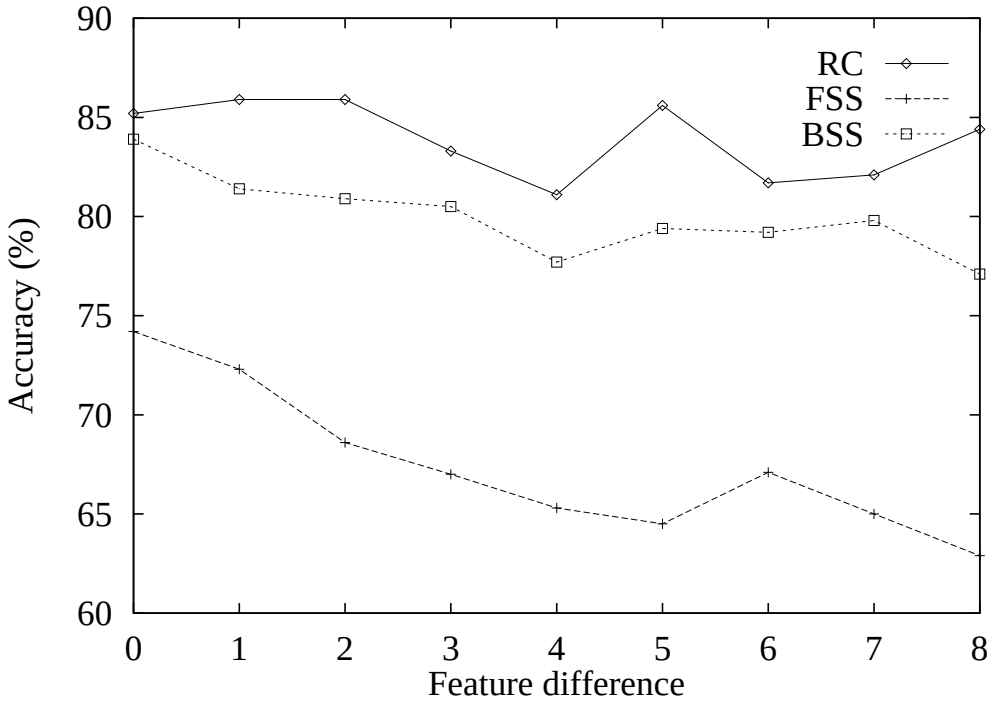

Figure 7: Accuracy as a function of context dependency.

investigate this point.[1]

The variation of the algorithms' accuracy with D is also of interest. All accuracies are negatively correlated with D, but the absolute value of the correlation is much smaller for RC (0.49) than for FSS and BSS (0.89 and 0.82, respectively). The downward slope of the regression line for RC's accuracy as a function of D (-0.35) is also much smaller than that for FSS (-1.21) and BSS (-0.61). We thus conclude that RC's higher performance is indeed at least partly due do its context sensitivity, and, pending further evidence, that RC is the feature selection algorithm of choice among the three studied, when feature relevance is significantly context-dependent.

8 Related work

Variations of FSS and BSS are described and evaluated in (Aha & Bankert, 1994). Beyond the pattern recognition approaches surveyed in (Devijver & Kittler, 1982) and (Kittler, 1986), many methods for feature selection have been proposed in the artificial intelligence literature in recent years (Almuallim & Dietterich, 1991; Kira & Rendell, 1992; Schlimmer, 1993; Vafaie & DeJong, 1993; Caruana & Freitag, 1994; John *et al.*, 1994; Skalak, 1994).

[1]With 100 examples, 20 runs with 9 values of D take approximately 10 hours of CPU time, of which less than 10 minutes is due to RC; 1000 examples would take on the order of $(1000/100)^2 \times 10$ hours, or 40 days. On the other hand, reducing F to allow more examples without increasing time would further reduce the observable range of D.

Cardie (1993) and Kibler and Aha (1987) use decision trees to select features for use in a lazy learner. Although each path through the tree represents a context-dependent set of relevant features, this information is discarded, and only the unstructured set of all the features used in the tree is passed to the lazy component. Another lazy-learner feature selection method, also based on decision trees, is described in (Langley & Sage, 1994). In this case all paths through the tree contain the same set of features, and so the level of context sensitivity is similar to that of BSS.

Decision-tree (Quinlan, 1993) and rule induction algorithms (Clark & Niblett, 1989) *per se* can be regarded as performing context-sensitive feature selection, and deriving most of their power from it. Because they search in a general-to-specific direction, adding one feature at a time, when seen as feature selectors they are most similar to FSS, and indeed have similar shortcomings with regard to detecting feature interactions (Pagallo & Haussler, 1990). It makes sense then to also view RC as a prototype specific-to-general rule induction algorithm, and expect it to have the same advantages relative to algorithms like ID3 and CN2 that BSS has with respect to FSS. This idea was taken further in the RISE system, described in (Domingos, 1995; Domingos, 1996). RISE is a full-fledged rule learner; starting with one rule per example, it searches for an optimal rule set, taking rule interactions during the induction process into account, allowing deletion of rules, and allowing generalization of numeric values to intervals. In (Domingos, 1996) it is compared with ID3 and CN2 on a large number of domains, and found to achieve significantly higher accuracy than either in most datasets, at the cost of increased memory usage, and with comparable running times. Lesion studies and monitoring of the algorithm's internals show that the lazy-learning component is essential to RISE's performance, as are the rule induction aspects and the conflict-resolution strategy used.

A related field is that of feature weighting (Aha, 1989; Kelly & Davis, 1991; Salzberg, 1991; Creecy, Masand, Smith & Waltz, 1992; Mohri & Tanaka, 1994). Feature selection can be seen as a special case of feature weighting where each weight is either 0 or 1, and thus weighting methods are potentially more powerful. However, because they have more degrees of freedom, they can also be harder to apply successfully, especially when there are few training examples.

Feature weights can be supplied by the designer, as in Skalak's (1992) Broadway system, or learned (see references above). Cain, Pazzani and Silverstein (1991) have an intermediate approach which combines lazy and explanation-based learning, assigning higher weights to features that appear in the derivation of the example's class using a pre-existing domain theory.

Feature weighting methods also vary in what the weights can depend on, and thus in their degree of context sensitivity. In the representationally simplest schemes, there is one weight per feature, and they are therefore completely context-free (Kelly & Davis, 1991; Salzberg, 1991; Lee, 1994; Mohri & Tanaka, 1994). More flexible approaches employ one weight per feature value (Nosofsky, Clark & Shin, 1989; Stanfill & Waltz, 1986), one weight per feature per class (Aha, 1989), or a combination of the two (Creecy *et al*, 1992), and thus exhibit a moderate degree of context sensitivity. In the case of continuous features, it is also possible to take into account the relative values of the feature in the instance and the example being classified, resulting in directional weights (Ricci & Avesani, 1995). The most elaborate algorithms have in effect one weight per feature per instance, and are consequently

fully context-sensitive; these weights can be assigned at classification time (Atkeson, Moore & Schaal, 1996) or at learning time (Aha & Goldstone, 1992). Seen as a 0-1 feature weighting algorithm, RC falls into this last category.

9 Concluding remarks

This article introduced a new feature selection algorithm for lazy learners. It differs from previous approaches in that it accounts for context dependency effects by selecting a possibly different set of relevant features for each instance. Empirical studies show that this often produces significant gains in accuracy, and that these gains increase with the the degree of context dependency of feature relevance.

Directions for future research include:

- Repeating the experiments described using other lazy learners, other context-free feature selection algorithms, and a wider variety of artificial domains, to check if the same qualitative results are obtained.
- Applying the approach described here to algorithms that use the k nearest instances to classify a test example, instead of only the nearest one.
- Combining RC with a context-free algorithm to optimize the selection of both context-free and context-sensitive relevant features, and to reduce the size of the feature set extracted.
- Developing versions of RC suitable for very large datasets and very large feature sets.

Acknowledgments

This work was partly supported by JNICT/Programa Ciência and Fulbright scholarships. The author is grateful to Dennis Kibler and Mike Pazzani for many helpful comments and suggestions, and to all the people who provided the datasets used in the empirical study, in particular M. Zwitter and M. Soklic of the University Medical Centre, Ljubljana, for supplying the lymphography, breast cancer and primary tumor datasets, and Robert Detrano, of the V.A. Medical Center, Long Beach and Cleveland Clinic Foundation, for supplying the heart disease dataset. Please see the documentation in the UCI Repository for detailed information on all datasets.

Appendix A

This appendix describes how, for each one of P prototypes, the relevant features are chosen at random in a way that guarantees that the feature difference between the prototypes (Equation 6) is on average a pre-specified value D.

If P_k is the number of prototypes in which feature k is relevant, Equation 6 can also be written as:

$$D = \frac{2}{P(P-1)} \sum_{k=1}^{F} P_k(P - P_k) \tag{7}$$

where P corresponds to N in (6), and F is the number of features. Let:

$$\rho_k = P_k(P - P_k) \tag{8}$$

and let $\overline{\rho}$ be the average value of ρ for the F features. $\overline{\rho}$ is determined by the desired value of D:

$$\overline{\rho} = \frac{P(P-1)}{2F} D \tag{9}$$

Next, k values of ρ_k such that their average is the value $\overline{\rho}$ above can be obtained from a uniform distribution in the interval $[0, 2\overline{\rho}]$. The corresponding P_ks are found by solving (8) for P_k, which is possible in general iff:

$$D \leq \frac{F}{4(1 - \frac{1}{P})} \tag{10}$$

This constrains the observable range of D given F and P. Finally, feature k is included in each prototype with probability P_k/P.

References

Aha, D. W. (1989). Incremental, Instance-Based Learning of Independent and Graded Concept Descriptions. In Proceedings of *The Sixth International Workshop on Machine Learning*, 387–391. Ithaca, NY: Morgan Kaufmann.

Aha, D. W. (1992). Generalizing from Case Studies: A Case Study. In Proceedings of *The Ninth International Workshop on Machine Learning*, 1–10. Aberdeen, Scotland: Morgan Kaufmann.

Aha, D. W. & Bankert, R. L. (1994). Feature Selection for Case-Based Classification of Cloud Types: An Empirical Comparison. In Proceedings of *The 1994 AAAI Workshop on Case-Based Reasoning*, 106–112. Seattle, WA: AAAI Press.

Aha, D. W. & Goldstone, R. L. (1992). Concept Learning and Flexible Weighting. In Proceedings of *The Fourteenth Annual Conference of the Cognitive Science Society*, 534–539. Evanston, IL: Lawrence Erlbaum.

Aha, D. W., Kibler, D. & Albert, M. K. (1991). Instance-Based Learning Algorithms. *Machine Learning*, 6:37–66.

Almuallim, H. & Dietterich, T. G. (1991). Learning with Many Irrelevant Features. In Proceedings of *The Ninth National Conference on Artificial Intelligence*, 547–552. Menlo Park, CA: AAAI Press.

Atkeson, C. G., Moore, A. W. & Schaal, S. (1996). Locally Weighted Learning. *Artificial Intelligence Review*. This issue.

Cain, T., Pazzani, M. J. & Silverstein, G. (1991). Using Domain Knowledge to Influence Similarity Judgments. In Proceedings of *The Case-Based Reasoning Workshop*, 191–199. Washington, DC: Morgan Kaufmann.

Cardie, C. (1993). Using Decision Trees to Improve Case-Based Learning. In Proceedings of *The Tenth International Conference on Machine Learning*, 25–32. Amherst, MA: Morgan Kaufmann.

Caruana, R. & Freitag, D. (1994). Greedy Attribute Selection. In Proceedings of *The Eleventh International Conference on Machine Learning*, 28–36. New Brunswick, NJ: Morgan Kaufmann.

Clark, P. & Niblett, T. (1989). The CN2 Induction Algorithm. *Machine Learning*, 3:261–283.

Cost, S. & Salzberg, S. (1993). A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features. *Machine Learning*, 10:57–78.

Creecy, R. H., Masand, B. M., Smith, S. J. & Waltz, D. L. (1992). Trading MIPS and Memory for Knowledge Engineering. *Communications of the ACM*, 35(8):48–63.

DeGroot, M. H. (1986). *Probability and Statistics*, Second Edition. Reading, MA: Addison-Wesley.

Devijver, P. A. & Kittler, J. (1982). *Pattern Recognition: A Statistical Approach*. Englewood Cliffs, NJ: Prentice/Hall.

Domingos, P. (1995). The RISE 2.0 System: A Case Study in Multistrategy Learning. TR-95-2, Department of Information and Computer Science, University of California at Irvine, Irvine, CA.

Domingos, P. (1996). Unifying Instance-Based and Rule-Based Induction. *Machine Learning*, 24:141–168.

Holte, R. C. (1993). Very Simple Classification Rules Perform Well on Most Commonly Used Datasets. *Machine Learning*, 11:63–91.

John, G. H., Kohavi, R. & Pfleger, K. (1994). Irrelevant Features and the Subset Selection Problem. In Proceedings of *The Eleventh International Conference on Machine Learning*, 121–129. New Brunswick, NJ: Morgan Kaufmann.

Kelly, J. D. & Davis, L. (1991). A Hybrid Genetic Algorithm for Classification. In Proceedings of *The Twelfth International Joint Conference on Artificial Intelligence*, 645–650. Sydney: Morgan Kaufmann.

Kibler, D. & Aha, D. W. (1987). Learning Representative Exemplars of Concepts: An Initial Case Study. In Proceedings of *The Fourth International Workshop on Machine Learning*, 24–30, Irvine, CA: Morgan Kaufmann.

Kira, A. & Rendell, L. A. (1992). A Practical Approach to Feature Selection. In Proceedings of *The Ninth International Workshop on Machine Learning*, 249–256. Aberdeen, Scotland: Morgan Kaufmann.

Kittler, J. (1986). Feature Selection and Extraction. In Young, T. Y. & Fu, K. S. (eds.) *Handbook of Pattern Recognition and Image Processing*. New York: Academic Press.

Kolodner, J. (1993). *Case-Based Reasoning*. San Mateo, CA: Morgan Kaufmann.

Langley, P. & Sage, S. (1994). Oblivious Decision Trees and Abstract Cases. In Proceedings of *The 1994 AAAI Workshop on Case-Based Reasoning*, 113–117. Seattle, CA: AAAI Press.

Lee, C. (1994). An Instance-Based Learning Method for Databases: An Information Theoretic Approach. In Proceedings of *The Ninth European Conference on Machine Learning*, 387–390. Catania, Italy: Springer-Verlag.

Mohri, T. & Tanaka, H. (1994). An Optimal Weighting Criterion of Case Indexing for Both Numeric and Symbolic Attributes. In Proceedings of *The 1994 AAAI Workshop on Case-Based Reasoning*, 123–127. Seattle, WA: AAAI Press.

Murphy, P. M. (1995). UCI Repository of Machine Learning Databases. Machine-Readable Data Repository, Department of Information and Computer Science, University of California at Irvine, Irvine, CA.

Niblett, T. (1987). Constructing Decision Trees in Noisy Domains. In Proceedings of *The Second European Working Session on Learning*, 67–78. Bled, Yugoslavia: Sigma.

Nosofsky, R. M., Clark, S. E. & Shin, H. J. (1989). Rules and Exemplars in Categorization, Identification, and Recognition. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 15:282–304.

Pagallo, G. & Haussler, D. (1990). Boolean Feature Discovery in Empirical Learning. *Machine Learning*, 3:71–99.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.

Ricci, F. & Avesani, P. (1995). Learning a Local Similarity Metric for Case-Based Reasoning. In Proceedings of *The First International Conference on Case-Based Reasoning*, 301–312. Sesimbra, Portugal: Springer-Verlag.

Salzberg, S. (1991). A Nearest Hyperrectangle Learning Method. *Machine Learning*, 6:251–276.

Schaffer, C. (1989). Analysis of Artificial Data Sets. In Proceedings of *The Second International Symposium on Artificial Intelligence*, 607–617. Monterrey, Mexico: McGraw-Hill.

Schlimmer, J. C. (1993). Efficiently Inducing Determinations: A Complete and Systematic Search Algorithm that Uses Optimal Pruning. In Proceedings of *The Tenth International Conference on Machine Learning*, 284–290. Amherst, MA: Morgan Kaufmann.

Skalak, D. B. (1992). Representing Cases as Knowledge Sources that Apply Local Similarity Metrics. In Proceedings of *The Fourteenth Annual Conference of the Cognitive Science Society*, 325–330. Evanston, IL: Lawrence Erlbaum.

Skalak, D. B. (1994). Prototype and Feature Selection by Sampling and Random Mutation Hill Climbing Algorithms. In Proceedings of *The Eleventh International Conference on Machine Learning*, 293–301. New Brunswick, NJ: Morgan Kaufmann.

Stanfill, C. & Waltz, D. (1986). Toward Memory-Based Reasoning. *Communications of the ACM*, 29:1213–1228.

Vafaie, H. & De Jong, K. (1993). Robust Feature Selection Algorithms. In Proceedings of *The Fifth IEEE International Conference on Tools for Artificial Intelligence*, 356–363. Boston, MA: IEEE Computer Society Press.