Journal of Machine Learning Research 5 (2004) 1331–1362 Submitted 11/02; Revised 6/03; Published 10/04

Randomized Variable Elimination

David J. Stracuzzi STRACUDJ@CS.UMASS.EDU
Paul E. Utgoff UTGOFF@CS.UMASS.EDU
Department of Computer Science
University of Massachusetts at Amherst
140 Governors Drive
Amherst, MA 01003, USA

Editor: Haym Hirsh

Abstract

Variable selection, the process of identifying input variables that are relevant to a particular learning problem, has received much attention in the learning community. Methods that employ a learning algorithm as a part of the selection process (wrappers) have been shown to outperform methods that select variables independently from the learning algorithm (filters), but only at great computational expense. We present a randomized wrapper algorithm whose computational requirements are within a constant factor of simply learning in the presence of all input variables, provided that the number of relevant variables is small and known in advance. We then show how to remove the latter assumption, and demonstrate performance on several problems.

1. Introduction

When learning in a supervised environment, a learning algorithm is typically presented with a set of N-dimensional data points, each with its associated target output. The learning algorithm then outputs a hypothesis describing the function underlying the data. In practice, the set of N input variables is carefully selected by hand in order to improve the performance of the learning algorithm in terms of both learning speed and hypothesis accuracy.

In some cases there may be a large number of inputs available to the learning algorithm, few of which are relevant to the target function, with no opportunity for human intervention. For example, feature detectors may generate a large number of features in a pattern recognition task. A second possibility is that the learning algorithm itself may generate a large number of new concepts (or functions) in terms of existing concepts. Valiant (1984), Fahlman and Lebiere (1990), and Kivinen and Warmuth (1997) all discuss situations in which a potentially large number of features are created during the learning process. In these situations, an automatic approach to variable selection is required.

One approach to variable selection that has produced good results is the wrapper method (John et al., 1994). Here, a search is performed in the space of variable subsets, with the performance of a specific learning algorithm based on such a subset serving as an evaluation function. Using the actual generalization performance of the learning algorithm as an evaluation metric allows this approach to search for the most predictive set of input variables with respect to the learner. However, executing the learning algorithm for each selection of variables during the search ultimately renders the approach intractable in the presence of many irrelevant variables.

In spite of the cost, variable selection can play an important role in learning. Irrelevant variables can often degrade the performance of a learning algorithm, particularly when data are limited. The main computational cost associated with the wrapper method is usually that of executing the learning algorithm. The learner must produce a hypothesis for each subset of the input variables. Even greedy selection methods (Caruana and Freitag, 1994) that ignore large areas of the search space can produce a large number of candidate variable sets in the presence of many irrelevant variables.

Randomized variable elimination avoids the cost of evaluating many variable sets by taking large steps through the space of possible input sets. The number of variables eliminated in a single step depends on the number of currently selected variables. We present a cost function whose purpose is to strike a balance between the probability of failing to select successfully a set of irrelevant variables and the cost of running the learning algorithm many times. We use a form of backward elimination approach to simplify the detection of relevant variables. Removal of any relevant variable should immediately cause the learner's performance to degrade. Backward elimination also simplifies the selection process when irrelevant variables are much more common than relevant variables, as we assume here.

Analysis of our cost function shows that the cost of removing all irrelevant variables is dominated by the cost of simply learning with all N variables. The total cost is therefore within a constant factor of the cost of simply learning the target function based on all N input variables, provided that the cost of learning grows at least polynomially in N. The bound on the complexity of our algorithm is based on the complexity of the learning algorithm being used. If the given learning algorithm executes in time $O(N^2)$, then removing the $N-r$ irrelevant variables via randomized variable elimination also executes in time $O(N^2)$. This is a substantial improvement compared to the factor N or more increase experienced in removing inputs one at a time.

2. Variable Selection

The specific problem of variable selection is the following: Given a large set of input variables and a target concept or function, produce a subset of the original input variables that predict best the target concept or function when combined into a hypothesis by a learning algorithm. The term "predict best" may be defined in a variety of ways, depending on the specific application and selection algorithm. Ideally the produced subset should be as small as possible to reduce training costs and help prevent overfitting.

From a theoretical viewpoint, variable selection should not be necessary. For example, the predictive power of Bayes rule increases monotonically with the number of variables. More variables should always result in more discriminating power, and removing variables should only hurt. However, optimal applications of Bayes rule are intractable for all but the smallest problems. Many machine learning algorithms perform sub-optimal operations and do not conform to the strict conditions of Bayes rule, resulting in the potential for a performance decline in the face of unnecessary inputs. More importantly, learning algorithms usually have access to a limited number of examples. Unrelated inputs require additional capacity in the learner, but do not bring new information in exchange. Variable selection is thus a necessary aspect of inductive learning.

A variety of approaches to variable selection have been devised. Most methods can be placed into one of two categories: *filter* methods or *wrapper* methods. Filter approaches perform variable selection independently of the learning algorithm, while wrappers make learner-dependent selections. A third group of special purpose methods perform feature selection in the context of neural

networks, known as parameter pruning. These methods cannot directly perform variable selection for arbitrary learning algorithms; they are approaches to removing irrelevant inputs from learning elements.

Many variable selection algorithms (although not all) perform some form of search in the space of variable subsets as part of their operation. A forward selection algorithm begins with the empty set and searches for variables to add. A backward elimination algorithm begins with the set of all variables and searches for variables to remove. Optionally, forward algorithms may occasionally choose to remove variables, and backward algorithms may choose to add variables. This allows the search to recover from previous poor selections. The advantage of forward selection is that, in the presence of many irrelevant variables, the size of the subsets will remain relatively small, helping to speed evaluation. The advantage of backward elimination is that recognizing irrelevant variables is easier. Removing a relevant variable from an otherwise complete set should cause a decline in the evaluation, while adding a relevant variable to an incomplete set may have little immediate impact.

2.1 Filters

Filter methods use statistical measures to evaluate the quality of the variable subsets. The goal is to find a set of variables that is best with respect to the specific quality measure. Determining which variables to include may either be done via an explicit search in the space of variable subsets, or by numerically weighting the variables individually and then selecting those with the largest weight. Filter methods often have the advantage of speed. The statistical measures used to evaluate variables typically require very little computation compared to cost of running a learning algorithm many times. The disadvantage is that variables are evaluated independently, not in the context of the learning problem.

Early filtering algorithms include FOCUS (Almuallim and Dietterich, 1991) and Relief (Kira and Rendell, 1992). FOCUS searches for a smallest set of variables that can completely discriminate between target classes, while Relief ranks variables according to a distance metric. Relief selects training instances at random when computing distance values. Note that this is not related to our approach of selecting variables at random.

Decision trees have also been employed to select input variables by first inducing a tree, and then selecting only those variables tested by decision nodes (Cardie, 1993; Kubat et al., 1993). In another vein, Koller and Sahami (1996) discuss a variable selection algorithm based on cross entropy and information theory.

Methods from statistics also provide a basis for a variety of variable filtering algorithms. Correlation-based feature selection (CFS) (Hall, 1999) attempts to find a set of variables that are each highly correlated with the target function, but not with each other. The ChiMerge (Kerber, 1992) and Chi2 algorithms (Liu and Setiono, 1997) remove both irrelevant and redundant variables using a χ^2 test to merge adjacent intervals of ordinal variables.

Other methods from statistics solve problems closely related to variable selection. For example, principal component analysis (see Dunteman, 1989) is a method for transforming the observed variables into a smaller number of dimensions, as opposed to removing irrelevant or redundant variables. Projection pursuit (Friedman and Tukey, 1974) and factor analysis (Thurstone, 1931) (see Cooley and Lohnes, 1971, for a detailed presentation) are used both to reduce dimensionality and to detect structure in relationships among variables.

Discussion of filtering methods for variable selection also arises in the pattern recognition literature. For example, Devijver and Kittler (1982) discuss the use of a variety of linear and non-linear distance measures and separability measures such as entropy. They also discuss several search algorithms, such as branch and bound and plus *l*-take away *r*. Branch and bound is an optimal search technique that relies on a careful ordering of the search space to avoid an exhaustive search. Plus *l*-take away *r* is more akin to the standard forward and backward search. At each step, *l* new variables are selected for inclusion in the current set and *r* existing variables are removed.

2.2 Wrappers

Wrapper methods attempt to tailor the selection of variables to the strengths and weaknesses of specific learning algorithms by using the performance of the learner to evaluate subset quality. Each candidate variable set is evaluated by executing the learning algorithm given the selected variables and then testing the accuracy of the resulting hypotheses. This approach has the advantage of using the actual hypothesis accuracy as a measure of subset quality. The problem is that the cost of repeatedly executing the learning algorithm can quickly become prohibitive. Nevertheless, wrapper methods do tend to outperform filter methods. This is not surprising given that wrappers evaluate variables in the context of the learning problem, rather than independently.

2.2.1 ALGORITHMS

John, Kohavi, and Pfleger (1994) appear to have coined the term "wrapper" while researching the method in conjunction with a greedy search algorithm, although the technique has a longer history (Devijver and Kittler, 1982). Caruana and Freitag (1994) also experimented with greedy search methods for variable selection. They found that allowing the search to either add variables or remove them at each step of the search improved over simple forward and backward searches. Aha and Bankert (1994) use a backward elimination beam search in conjunction with the IB1 learner, but found no evidence to prefer this approach to forward selection. OBLIVION (Langley and Sage, 1994) selects variables for the nearest neighbor learning algorithm. The algorithm uses a backward elimination approach with a greedy search, terminating when the nearest neighbor accuracy begins to decline.

Subsequent work by Kohavi and John (1997) used forward and backward best-first search in the space of variable subsets. Search operators generally include adding or removing a single variable from the current set. This approach is capable of producing a minimal set of input variables, but the cost grows exponentially in the face of many irrelevant variables. Compound operators generate nodes deep in the search tree early in the search by combining the best children of a given node. However, the cost of running the best-first search ultimately remains prohibitive in the presence of many irrelevant variables.

Hoeffding races (Maron and Moore, 1994) take a different approach. All possible models (selections) are evaluated via leave-one-out cross validation. For each of the *N* evaluations, an error confidence interval is established for each model. Models whose error lower bound falls below the upper bound of the best model are discarded. The result is a set of models whose error is insignificantly different.

Several algorithms for constructing regression models are also forms of wrapper methods. For example, Least angle regression (Efron et al., 2003), which generalizes and improves upon several forward selection regression algorithms, adds variables to the model incrementally.

Genetic algorithms have been also been applied as a search mechanism for variable selection. Vafaie and De Jong (1995) describe using a genetic algorithm to perform variable selection. They used a straightforward representation in which individual chromosomes were bit-strings with each bit marking the presence or absence of a specific variable. Individuals were evaluated by training and then testing the learning algorithm. In a similar vein, SET-Gen (Cherkauer and Shavlik, 1996) used a fitness (evaluation) function that included both the accuracy of the induced model and the comprehensibility of the model. The learning model used in their experiments was a decision tree and comprehensibility was defined as a combination of tree size and number of features used. The FSS-EBNA algorithm (Inza et al., 2000) used Bayesian Networks to mate individuals in a GA-based approach to variable selection.

The relevance-in-context (RC) algorithm (Domingos, 1997) is based on the idea that some features may only be relevant in particular areas of the instance space for instance based (lazy) learners. Clusters of training examples are formed by finding examples of the same class with nearly equivalent feature vectors. The features along which the examples differ are removed and the accuracy of the entire model is determined. If the accuracy declined, the features are restored and the failed examples are removed from consideration. The algorithm continues until there are no more examples to consider. Results showed that RC outperformed other wrapper methods with respect to a 1-NN learner.

2.2.2 LEARNER SELECTIONS

Many learning algorithms already contain some (possibly indirect) form of variable selection, such as pruning in decision trees. This raises the question of whether the variable selections made by the learner should be used by the wrapper. Such an approach would almost certainly run faster than methods that rely only on the wrapper to make variable selections. The wrapper selects variables for the learner, and then executes the learner. If the resulting hypothesis is an improvement, then the wrapper further removes all variables not used in the hypothesis before continuing on with the next round of selections.

This approach assumes the learner is capable of making beneficial variable selections. If this were true, then both filter and wrapper methods would be largely irrelevant. Even the most sophisticated learning algorithms may perform poorly in the presence of highly correlated, redundant or irrelevant variables. For example, John, Kohavi, and Pfleger (1994) and Kohavi (1995) both demonstrate how C4.5 (Quinlan, 1993) can be tricked into making bad decisions *at the root*. Variables highly correlated with the target value, yet ultimately useless in terms of making beneficial data partitions, are selected near the root, leading to unnecessarily large trees. Moreover, these bad decisions cannot be corrected by pruning. Only variable selection performed outside the context of the learning algorithm can recognize these types of correlated, irrelevant variables.

2.2.3 ESTIMATING PERFORMANCE

One question that any wrapper method must consider is how to obtain a good estimate of the accuracy of the learner's hypothesis. Both the amount and quality of data available to the learner affect the testing accuracy. Kohavi and John (1997) suggest using multiple runs of five-fold cross-validation to obtain an error estimate. They determine the number of cross-validation runs by continuing until the standard deviation of the accuracy estimate is less than 1%. This has the nice property of (usually) requiring fewer runs for large data sets. However, in general, cross-validation

is an expensive procedure, requiring the learner to produce several hypotheses for each selection of variables.

2.3 Model Specific Methods

Many learning algorithms have built-in variable (or parameter) selection algorithms which are used to improve generalization. As noted above, decision tree pruning is one example of built-in variable selection. Connectionist algorithms provide several other examples, known as parameter pruning. As in the more general variable selection problem, extra weights (parameters) in a network can degrade the performance of the network on unseen test instances, and increase the cost of evaluating the learned model. Parameter pruning algorithms often suffer the same disadvantages as tree pruning. Poor choices made early in the learning process can not usually be undone.

One method for dealing with unnecessary network parameters is weight decay (Werbos, 1988). Weights are constantly pushed toward zero by a small multiplicitive factor in the update rule. Only the parameters relevant to the problem receive sufficiently large weight updates to remain significant. Methods for parameter pruning include the optimal brain damage (OBD) (LeCun et al., 1990) and optimal brain surgeon (OBS) (Hassibi and Stork, 1993) algorithms. Both rely on the second derivative to determine the importance of connection weights. Sensitivity-based pruning (Moody and Utans, 1995) evaluates the effect of removing a *network* input by replacing the input by its mean over all training points. The autoprune algorithm (Finnoff et al., 1993) defines an importance metric for weights based on the assumption that irrelevant weights will become zero. Weights with a low metric value are considered unimportant and are removed from the network.

There are also connectionist approaches that specialize in learning quickly in the presence irrelevant inputs, without actually removing them. The WINNOW algorithm (Littlestone, 1988) for Boolean functions and the exponentiated gradient algorithm (Kivinen and Warmuth, 1997) for real-valued functions are capable of learning linearly separable functions efficiently in the presence of many irrelevant variables. Exponentiated gradient algorithms, of which WINNOW is a special case, are similar to gradient descent algorithms, except that the updates are multiplicative rather than additive.

The result is a mistake bound that is linear in the number of relevant inputs, but only logarithmic in the number of irrelevant inputs. Kivinen and Warmuth also observed that the number of examples required to learn an accurate hypothesis also appears to obey these bounds. In other words, the number of training examples required by exponentiated gradient algorithms grows only logarithmicly in the number of irrelevant inputs.

Exponentiated gradient algorithms may be applied to the problem of separating the set of relevant variables from irrelevant variables by running them on the available data and examining the resulting weights. Although exponentiated gradient algorithms produce a minimum error fit of the data in non-separable problems, there is no guarantee that such a fit will rely on the variables relevant to a non-linear fit.

Many algorithms that are directly applicable in non-linear situations experience a performance decline in the presence of irrelevant input variables. Even support vector machines, which are often touted as impervious to irrelevant variables, have been shown to improve performance with feature selection (Weston et al., 2000). A more general approach to recognizing relevant variables is needed.

3. Setting

Our algorithm for randomized variable elimination (RVE) requires a set (or sequence) of N-dimensional vectors x_i with labels y_i. The learning algorithm $\mathcal{L}$ is asked to produce a hypothesis h based only on the inputs x_{ij} that have not been marked as irrelevant (alternatively, a preprocessor could remove variables marked irrelevant). We assume that the hypotheses bear some relation to the data and input values. A degenerate learner (such as one that produces the same hypothesis regardless of data or input variables) will in practice cause the selection algorithm ultimately to select zero variables. This is true of most wrapper methods. For the purposes of this article, we use generalization accuracy as the performance criteria, but this is not a requirement of the algorithm.

We make the assumption that the number r of relevant variables is at least two to avoid degenerate cases in our analysis. The number of relevant variables should be small compared to the total number of variables N. This condition is not critical to the functionality of the RVE algorithm; however the benefit of using RVE increases as the ratio of N to r increases. Importantly, we assume that the number of relevant variables is known in advance, although which variables are relevant remains hidden. Knowledge of r is a very strong assumption in practice, as such information is not typically available. We remove this assumption in Section 6, and present an algorithm for estimating r while removing variables.

4. The Cost Function

Randomized variable elimination is a wrapper method motivated by the idea that, in the presence of many irrelevant variables, the probability of successfully selecting several irrelevant variables simultaneously at random from the set of all variables is high. The algorithm computes the cost of attempting to remove k input variables out of n remaining variables given that r are relevant. A sequence of values for k is then found by minimizing the aggregate cost of removing all $N-r$ irrelevant variables. Note that n represents the number of remaining variables, while N denotes the total number of variables in the original problem.

The first step in applying the RVE algorithm is to define the cost metric for the given learning algorithm. The cost function can be based on a variety of metrics, depending on which learning algorithm is used and the constraints of the application. Ideally, a metric would indicate the amount of computational effort required for the learning algorithm to produce a hypothesis.

For example, an appropriate metric for the perceptron algorithm (Rosenblatt, 1958) might relate to the number of weight updates that must be performed, while the number of calls to the data purity criterion (e.g. information gain (Quinlan, 1986)) may be a good metric for decision tree induction algorithms. Sample complexity represents a metric that can be applied to almost any algorithm, allowing the cost function to compute the number of instances the learner must see in order to remove the irrelevant variables from the problem. We do not assume a specific metric for the definition and analysis of the cost function.

4.1 Definition

The first step of defining the cost function is to consider the probability

$$p^+(n,r,k) = \prod_{i=0}^{k-1} \left(\frac{n-r-i}{n-i} \right)$$

of successfully selecting k irrelevant variables at random and without replacement, given that there are n remaining and r relevant variables. Next we use this probability to compute the expected number of consecutive failures before a success at selecting k irrelevant variables from n remaining given that r are relevant. The expression

$$E^-(n,r,k) = \frac{1 - p^+(n,r,k)}{p^+(n,r,k)}$$

yields the expected number of consecutive trials in which at least one of the r relevant variables will be randomly selected along with irrelevant variables prior to success.

We now discuss the cost of selecting and removing k variables, given n and r. Let $M(\mathcal{L},n)$ represent an upper bound on the cost of running algorithm $\mathcal{L}$ based on n inputs. In the case of a perceptron, $M(\mathcal{L},n)$ could represent an estimated upper bound on the number of updates performed by an n-input perceptron. In some instances, such as a backpropagation neural network (Rumelhart and McClelland, 1986), providing such a bound may be troublesome. In general, the order of the worst case computational cost of the learner with respect to the number of inputs is all that is needed. The bounding function should account for any assumptions about the nature of the learning problem. For example, if learning Boolean functions requires less computational effort than learning real-valued functions, then $M(\mathcal{L},n)$ should include this difference. The general cost function described below therefore need not make any additional assumptions about the data.

In order to simplify the notation somewhat, the following discussion assumes a fixed algorithm for $\mathcal{L}$. The expected cost of successfully removing k variables from n remaining given that r are relevant is given by

$$\begin{aligned} I(n,r,k) &= E^-(n,r,k) \cdot M(\mathcal{L},n-k) + M(\mathcal{L},n-k) \\ &= M(\mathcal{L},n-k)\,(E^-(n,r,k)+1) \end{aligned}$$

for $1 \leq k \leq n-r$. The first term in the equation denotes the expected cost of failures (i.e. unsuccessful selections of k variables) while the second denotes the cost of the one success.

Given this expected cost of removing k variables, we can now define recursively the expected cost of removing all $n-r$ irrelevant variables. The goal is to minimize locally the expected cost of removing k inputs with respect to the expected remaining cost, resulting in a global minimum expected cost for removing all $n-r$ irrelevant variables. The use of a greedy minimization step relies upon the assumption that $M(\mathcal{L},n)$ is monotonic in n. This is reasonable in the context of metrics such as number of updates, number of data purity tests, and sample complexity. The cost (with respect to learning algorithm $\mathcal{L}$) of removing $n-r$ irrelevant variables is represented by

$$I_{sum}(n,r) = \min_k (I(n,r,k) + I_{sum}(n-k,r)).$$

The first part of the minimization term represents the cost of removing the first k variables while the second part represents the cost of removing the remaining $n-r-k$ irrelevant variables. Note that we define $I_{sum}(r,r) = 0$.

The optimal value $k_{opt}(n,r)$ for k given n and r can be determined in a manner similar to computing the cost of removing all $n-r$ irrelevant inputs. The value of k is computed as

$$k_{opt}(n,r) = \arg\min_k (I(n,r,k) + I_{sum}(n-k,r)).$$

4.2 Analysis

The primary benefit of this approach to variable elimination is that the combined cost (in terms of the metric $M(\mathcal{L},n)$) of learning the target function and removing the irrelevant input variables is within a constant factor of the cost of simply learning the target function based on all N inputs. This result assumes that the function $M(\mathcal{L},n)$ is at least a polynomial of degree $j > 0$. In cases where $M(\mathcal{L},n)$ is sub-polynomial, running the RVE algorithm increases the cost of removing the irrelevant inputs by a factor of $\log(n)$ over the cost of learning alone as shown below.

4.2.1 REMOVING MULTIPLE VARIABLES

We now show that the above average-case bounds on the performance of the RVE algorithm hold. The worst-case is the unlikely condition in which the algorithm always selects a relevant variable. We assume integer division here for simplicity. First let $k = \frac{n}{r}$, which allows us to remove the minimization term from the equation for $I_{sum}(n,r)$ and reduces the number of variables. This value of k is not necessarily the value selected by the above equations. However, the cost function is computed via dynamic programming, and the function $M(\mathcal{L},n)$ is assumed monotonic. Any differences between our chosen value of k and the actual value computed by the equations can only serve to decrease further the cost of the algorithm. Note also that, because k depends on the number of current variables n, k changes at each iteration of the algorithm.

The probability of success $p^+(n,r,\frac{n}{r})$ is minimized when $n = r+1$, since there is only one possible successful selection and r possible unsuccessful selections. This in turn maximizes the expected number of failures $E^-(n,r,\frac{n}{r}) = r$. The formula for $I(n,r,k)$ is now rewritten as

$$I(n,r,\frac{n}{r}) \leq (r+1)\cdot M(\mathcal{L},n-\frac{n}{r}),$$

where both $M(\mathcal{L},n-k)$ terms have been combined.

The expected cost of removing all $n-r$ irrelevant inputs may now be rewritten as a summation

$$I_{sum}(n,r) \leq \sum_{i=0}^{r\lg(n)} \left((r+1)M\left(\mathcal{L},n\left(\frac{r-1}{r}\right)^{i+1}\right)\right).$$

The second argument to the learning algorithm's cost metric M denotes the number of variables used at step i of the RVE algorithm. Notice that this number decreases geometrically toward r (recall that $n = r$ is the terminating condition for the algorithm). The logarithmic factor of the upper bound on the summation, $\frac{\lg(n)-\lg(r)}{\lg(1+1/(r-1))} \leq r\lg(n)$, follows directly from the geometric decrease in the number of variables used at each step of the algorithm. The linear factor r follows from the relationship between k and r. In general, as r increases, k decreases. Notice that as r approaches N, RVE and our cost function degrade into testing and removing variables individually.

Concluding the analysis, we observe that for functions $M(\mathcal{L},n)$ that are at least polynomial in n with degree $j > 0$, the cost incurred by the first pass of RVE ($i = 0$) will dominate the remainder of the terms. The average-case cost of running RVE in these cases is therefore bounded by $I_{sum}(N,r) \leq O(rM(\mathcal{L},N))$. An equivalent view is that the sum of a geometrically decreasing series converges to a constant. Thus, under the stated assumption that r is small compared to (and independent of) N, RVE requires only a constant factor more computation than the learner alone.

When $M(\mathcal{L},n)$ is sub-linear in n (e.g logarithmic), each pass of the algorithm contributes significantly to the total expected cost, resulting in an average-case bound of $O(r^2\log(N)M(\mathcal{L},N))$. Note

that we use average-case analysis here because in the worst case the algorithm can randomly select relevant variables indefinitely. In practice however, long streaks of bad selections are rare.

4.2.2 REMOVING VARIABLES INDIVIDUALLY

Consider now the cost of removing the $N-r$ irrelevant variables one at a time ($k=1$). Once again the probability of success is minimized and the expected number of failures is maximized at $n=r+1$. The total cost of such an approach is given by

$$I_{sum}(n,r) = \sum_{i=1}^{n-r} (r+1) \cdot M(\mathcal{L}, n-i).$$

Unlike the multiple variable removal case, the number of variables available to the learner at each step decreases only arithmetically, resulting in a linear number of steps in n. This is an important deviation from the multiple selection case, which requires only a logarithmic number of steps. The difference between the two methods becomes substantial when N is large. Concluding, the bound on the average-case cost of RVE is $I_{sum}(N,r) \leq O(NrM(\mathcal{L},N))$ when $k=1$. This is true regardless of whether the variables are selected randomly or deterministically at each step.

In principle, a comparison should be made between the upper bound of the algorithm that removes multiple variables per step and the lower bound of the algorithm that removes a single variable per step in order to show the differences clearly. However, generating a sufficiently tight lower bound requires making very strong assumptions on the form of $M(\mathcal{L},n)$. Instead, note that the two upper bounds are comparable with respect to $M(\mathcal{L},n)$ and differ only by the leading factor N.

4.3 Computing the Cost and k-Sequence

The equations for $I_{sum}(n,r)$ and $k_{opt}(n,r)$ suggest a simple $O(N^2)$ dynamic programming solution for computing both the cost and optimal k-sequence for a problem of N variables. Table 1 shows an algorithm for computing a table of cost and k values for each i with $r+1 \leq i \leq N$. The algorithm fills in the tables of values by starting with small n, and bootstrapping to find values for increasingly large n. The function $I(n,r,k)$ in Table 1 is computed as described above.

The $O(N^2)$ cost of computing the sequence of k values is of some concern. When N is large and the learning algorithm requires time only linear in N, the cost of computing the optimal k-sequence could exceed the cost of removing the irrelevant variables. In practice the cost of computing values for k is negligible for problems up to $N=1000$. For larger problems, one solution is simply to set $k=\frac{n}{r}$ as in Section 4.2.1. The analysis shows that this produces good performance and requires no computational overhead.

5. The Randomized Variable Elimination Algorithm

Randomized variable elimination conducts a backward search through the space of variable subsets, eliminating one or more variables per step. Randomization allows for selection of irrelevant variables with high probability, while selecting multiple variables allows the algorithm to move through the space without incurring the cost of evaluating the intervening points in the space. RVE conducts its search along a very narrow trajectory. The space of variable subsets is sampled sparsely, rather than broadly and uniformly. This structured yet random search allows RVE to reduce substantially the total cost of selecting relevant variables.

Given: $\mathcal{L}, N, r$

$I_{sum}[r+1..N] \leftarrow 0$
$k_{opt}[r+1..N] \leftarrow 0$

for $i \leftarrow r+1$ to N **do**
    $bestCost \leftarrow \infty$
    for $k \leftarrow 1$ to $i-r$ **do**
        $temp \leftarrow I(i,r,k) + I_{sum}[i-k]$
        if $temp < bestCost$ **then**
            $bestCost \leftarrow temp$
            $bestK \leftarrow k$
    $I_{sum}[i] \leftarrow bestCost$
    $k_{opt}[i] \leftarrow bestK$

Table 1: Algorithm for computing k and cost values.

A backward approach serves two purposes for this algorithm. First, backward elimination eases the problem of recognizing irrelevant or redundant variables. As long as a core set of relevant variables remains intact, removing other variables should not harm the performance of a learning algorithm. Indeed, the learner's performance may *increase* as irrelevant features are removed from consideration. In contrast, variables whose relevance depends on the presence of other variables may have no noticeable effect when selected in a forward manner. Thus, mistakes should be recognized immediately via backward elimination, while good selections may go unrecognized by a forward selection algorithm.

The second purpose of backward elimination is to ease the process of selecting variables. If most variables in a problem are irrelevant, then a random selection of variables is naturally likely to uncover them. Conversely, a random selection is unlikely to turn up relevant variables in a forward search. Thus, the forward search must work harder to find each relevant variable than backward search does for irrelevant variables.

5.1 Algorithm

The algorithm begins by computing the values of $k_{opt}(i,r)$ for all $r+1 \leq i \leq n$. Next it generates an initial hypothesis based on all n input variables. Then, at each step, the algorithm selects $k_{opt}(n,r)$ input variables at random for removal. The learning algorithm is trained on the remaining $n-k$ inputs, and a hypothesis h is produced. If the error $e(h')$ of hypothesis h' is less than the error $e(h)$ of the previous hypothesis h (possibly within a given tolerance), then the selected k inputs are marked as irrelevant and are all simultaneously removed from future consideration. Kohavi and John (1997) provide an in depth discussion on evaluating and comparing hypotheses based on limited data sets. If the learner was unsuccessful, meaning the new hypothesis had a larger error, then at least one of the selected variables was relevant. A new set of inputs is selected and the process repeats. The algorithm terminates when all $n-r$ irrelevant inputs have been removed. Table 2 shows the RVE algorithm.

Given: $\mathcal{L}$, n, r, *tolerance*

compute tables for $I_{sum}(i,r)$ and $k_{opt}(i,r)$
$h \leftarrow$ hypothesis produced by $\mathcal{L}$ on n inputs

while $n > r$ **do**
    $k \leftarrow k_{opt}(n,r)$
    select k variables at random and remove them
    $h' \leftarrow$ hypothesis produced by $\mathcal{L}$ on $n-k$ inputs
    if $e(h') - e(h) \leq$ *tolerance* **then**
        $n \leftarrow n-k$
        $h \leftarrow h'$
    else
        replace the selected k variables

Table 2: Randomized backward-elimination variable selection algorithm.

The structured search performed by RVE is easily distinguished from other randomized search methods. For example, genetic algorithms maintain a population of states in the search space and randomly mate the states to produce offspring with properties of both parents. The effect is an initially broad search that targets more specific areas as the search progresses. A wide variety of subsets are explored, but the cost of so much exploration can easily exceed the cost of a traditional greedy search. See Goldberg (1989) or Mitchell (1996) for detailed discussions on how genetic algorithms conduct search.

While GAs tend to drift through the search space based on the properties of individuals in the population, the LVF algorithm (Liu and Setino, 1996) samples the space of variable subsets uniformly. LVF selects both the size of each subset and the member variables at random. Although such an approach is not susceptible to "bad decisions" or local minima, the probability of finding a best or even good variable subset decreases exponentially as the number of irrelevant variables increases. Unlike RVE, LVF is a filtering method, which relies on the inconsistency rate (number of equivalent instances divided by number of total instances) in the data with respect to the selected variables.

5.2 A Simple Example

The preceding presentation of the RVE algorithm has remained strictly general, relying on no specific learning algorithm or cost metric. We consider now a specific example of how the randomized variable elimination algorithm may be applied to a linear threshold unit. The specific task examined here is to learn a Boolean function that is true when seven out of ten relevant variables are true, given a total of 100 input variables. In order to ensure that the hypotheses generated for each selection of variables has nearly minimal error, we use the thermal perceptron training algorithm (Frean, 1992). The thermal perceptron uses simulated annealing to settle weights regardless of data separability. The pocket algorithm (Gallant, 1990) is also applicable, but we found this to be slower and prone to more testing errors.

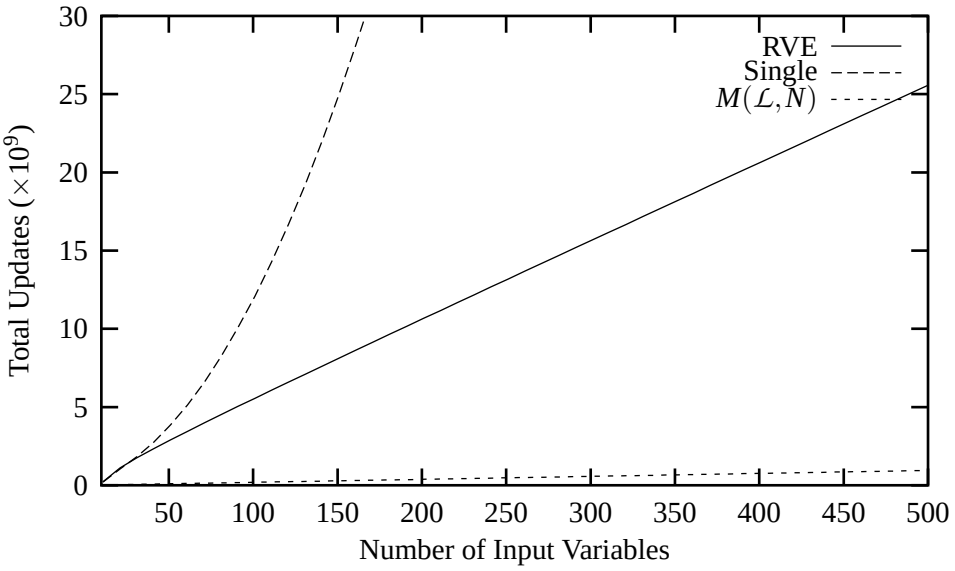

Figure 1: Plot of the expected cost of running RVE ($I_{sum}(N, r = 10)$) along with the cost of removing inputs individually, and the estimated number of updates $M(\mathcal{L},N)$.

Twenty problems were generated randomly with $N = 100$ input variables, of which 90 are irrelevant and $r = 10$ are relevant. Each of the twenty problems used a different set of ten relevant variables (selected at random) and different data sets. Two data sets, each with 1000 instances, were generated independently for each problem. One data set was used for training while the other was used to validate the error of the hypotheses generated during each round of selections. The values of the 100 input variables were all generated independently. The mean number of unique instances with respect to the ten relevant variables was 466 .

The first step in applying the RVE algorithm is to define the cost metric and the function $M(\mathcal{L},n)$ for learning on n inputs. For the perceptron, we choose the number of weight updates as the metric. The thermal perceptron anneals a temperature T that governs the magnitude of the weight updates. Here we used $T_0 = 2$ and decayed the temperature at a rate of 0.999 per training epoch until $T < 0.3$ (we observed no change in the hypotheses produced by the algorithm for $T < 0.3$). Given the temperature and decay rate, exactly 1897 training epochs are performed each time a thermal perceptron is trained. With 1000 instances in the training data, the cost of running the learning algorithm is fixed at $M(\mathcal{L},n) = 1897000(n+1)$. Given the above cost formula for an n-input perceptron, a table of values for $I_{sum}(n,r)$ and $k_{opt}(n,r)$ can be constructed.

Figure 1 plots a comparison of the computed cost of the RVE algorithm, the cost of removing variables individually, and the estimated number of updates $M(\mathcal{L},N)$ of an N-input perceptron. The calculated cost of the RVE algorithm maintains a linear growth rate with respect to N, while the cost of removing variables individually grows as N^2. This agrees with our analysis of the RVE and individual removal approaches. Relationships similar to that shown in Figure 1 arise for other values of r, although the constant factor that separates $I_{sum}(n,r)$ and $M(\mathcal{L},n)$ increases with r.

After creating the table $k_{opt}(n,r)$, the selection and removal process begins. Since the seven-of-ten learning problem is linearly separable, the tolerance for comparing the new and current hypotheses was set to near zero. A small tolerance of 0.06 (equivalent to about 15 misclassifications) is necessary since the thermal perceptron does not guarantee a minimum error hypothesis.

We also allow the current hypothesis to bias the next by not randomizing the weights (of remaining variables) after each pass of RVE. Small value weights, suggesting potential irrelevant variables, can easily transfer from one hypothesis to the next, although this is not guaranteed. Seeding the perceptron weights may increase the chance of finding a linear separator *if one exists*. If no separator exists, then seeding the weights should have minimal impact. In practice we found that the effect of seeding the weights was nullified by the pocket perceptron's use of annealing.

5.3 Example Results

The RVE algorithm was run using the twenty problems described above. Hypotheses based on ten variables were produced using an average of 5.45×10^9 weight updates, 81.1 calls to the learning algorithm, and 359.9 seconds on a 3.12 GHz Intel Xenon processor. A version of the RVE algorithm that removes variables individually (i.e. k was set permanently to 1) was also run, and produced hypotheses using 12.7×10^9 weight updates, 138.7 calls to the learner, and 644.7 seconds. These weight update values agree with the estimate produced by the cost function. Both versions of the algorithm generated hypotheses that included irrelevant and excluded relevant variables for three of the test problems. All cases in which the final selection of variables was incorrect were preceded by an initial hypothesis (based on all 100 variables) with unusually high error (error greater than 0.18 or approximately 45 misclassified instances). Thus, poor selections occured for runs in which the first hypothesis produced has high error due to annealing in the pocket perceptron.

Figure 2 plots the average number of inputs used for each variable set size (number of inputs) compared to the total number of weight updates. Each marked point on the plot denotes a size of the set of input variables given to the perceptron. The error bars indicate the standard deviation in number of updates required to reach that point. Every third point is plotted for the individual removal algorithm. Compare both the rate of drop in inputs and the number of hypotheses trained for the two RVE versions. This reflects the balance between the cost of training and unsuccessful variable selections. Removing variables individually in the presence of many irrelevant variables ignores the cost of training each hypothesis, resulting in a total cost that rises quickly early in the search process.

6. Choosing k When r Is Unknown

The assumption that the number of relevant variables r is known has played a critical role in the preceding discussion. In practice, this is a strong assumption that is not easily met. We would like an algorithm that removes irrelevant attributes efficiently without such knowledge. One approach would be simply to guess values for r and see how RVE fares. This is unsatisfying however, as a poor guess can destroy the efficiency of RVE. In general, guessing specific values for r is difficult, but placing a loose bound around r may be much easier. In some cases, the maximum value for r may be known to be much less than N, while in other cases, r can always be bounded by 1 and N.

Given some bound on the maximum r_{max} and minimum r_{min} values for r, a binary search for r can be conducted during RVE's search for relevant variables. This relies on the idea that RVE attempts to balance the cost of learning against the cost of selecting relevant variables for removal.

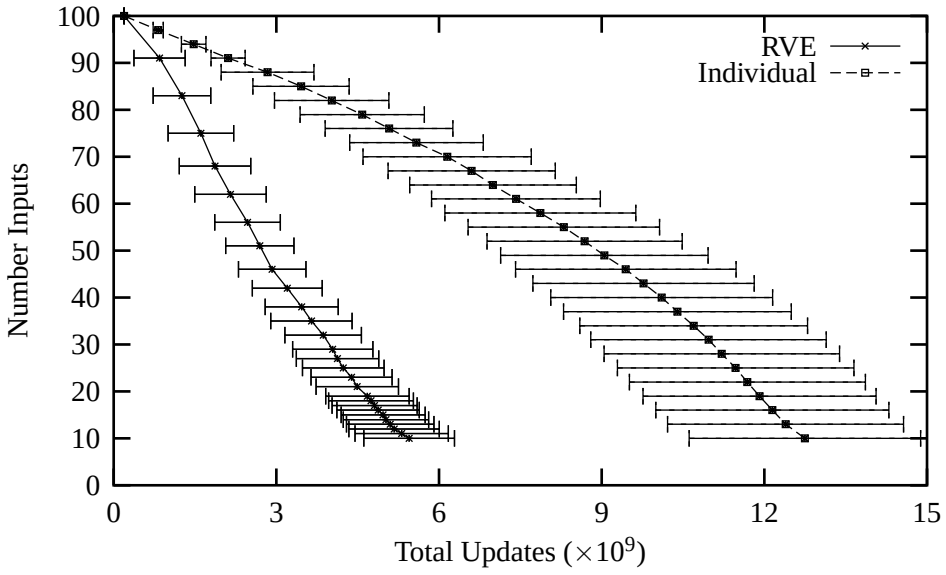

Figure 2: A comparison between the number of inputs on which the perceptrons are trained and the mean aggregate number of updates performed by the perceptrons.

At each step of RVE, a certain number of failures, $E^-(n,r,k)$, are expected. Thus, if selecting variables for removal is too easy (i.e. we are selecting too few variables at each step), then the estimate for r is too high. Similarly, if selection fails an inordinate number of times, then the estimate for r is too low.

The choice of when to adjust r is important. The selection process must be allowed to fail a certain number of times for each success, but allowing too many failures will decrease the efficiency of the algorithm. We bound the number of failures by $c_1 E^-(n,r,k)$ where $c_1 > 1$ is a constant. This allows for the failures prescribed by the cost function along with some amount of "bad luck" in the random variable selections. The number of consecutive successes is bounded similarly by $c_2(r - E^-(n,r,k))$ where $c_2 > 0$ is a constant. Since $E^-(n,r,k))$ is at most r, the value of this expression decreases as the expected number of failures increases. In practice $c_1 = 3$ and $c_2 = 0.3$ appear to work well.

6.1 A General Purpose Algorithm

Randomized variable elimination including a binary search for r (RVErS — "reverse") begins by computing tables for $k_{opt}(n,r)$ for values of r between r_{min} and r_{max}. Next an initial hypothesis is generated and the variable selection loop begins. The algorithm chooses the number of variables to remove at each step based on the current value of r. Each time the bound on the maximum number of successful selections is exceeded, r_{max} reduces to r and a new value is calculated as $r = \frac{r_{max}+r_{min}}{2}$. Similarly, when the bound on consecutive failures is exceeded, r_{min} increases to r and r is recalculated. The algorithm also checks to ensure that the current number of variables never falls below r_{min}. If this occurs, r, r_{min} and r_{max} are all set to the current number of variables. RVErS

Given: $\mathcal{L}$, c_1, c_2, n, r_{max}, r_{min}, $tolerance$

compute tables $I_{sum}(i,r)$ and $k_{opt}(i,r)$ for $r_{min} \leq r \leq r_{max}$
$r \leftarrow \frac{r_{max}+r_{min}}{2}$
$success$, $fail \leftarrow 0$
$h \leftarrow$ hypothesis produced by $\mathcal{L}$ on n inputs

repeat
    $k \leftarrow k_{opt}(n,r)$
    select k variables at random and remove them
    $h' \leftarrow$ hypothesis produced by $\mathcal{L}$ on $n-k$ inputs
    if $e(h') - e(h) \leq tolerance$ **then**
        $n \leftarrow n-k$
        $h \leftarrow h'$
        $success \leftarrow success+1$
        $fail \leftarrow 0$
    else
        replace the selected k variables
        $fail \leftarrow fail+1$
        $success \leftarrow 0$

    if $r \leq r_{min}$ **then**
        r, r_{max}, $r_{min} \leftarrow n$
    else if $fail \geq c_1 E^-(n,r,k)$ **then**
        $r_{min} \leftarrow r$
        $r \leftarrow \frac{r_{max}+r_{min}}{2}$
        $success$, $fail \leftarrow 0$
    else if $success \geq c_2(r - E^-(n,r,k))$ **then**
        $r_{max} \leftarrow r$
        $r \leftarrow \frac{r_{max}+r_{min}}{2}$
        $success$, $fail \leftarrow 0$
until $r_{min} < r_{max}$ **and** $fail \leq c_1 E^-(n,r,k)$

Table 3: Randomized variable elimination algorithm including a search for r.

terminates when r_{min} and r_{max} converge and $c_1 E^-(n,r,k)$ consecutive variable selections fail. Table 3 shows the RVE*r*S algorithm.

While RVE*r*S can produce good performance without finding the exact value of r, how well the estimated value must approximate the actual value is unclear. An important factor in determining the complexity of RVE*r*S is how quickly the algorithm reaches a good estimate for r. In the best case, the search for r will settle on a good approximation of the actual number of relevant variables immediately, and the RVE complexity bound will apply. In the worst case, the search for r will proceed slowly over values of r that are too high, causing RVE*r*S to behave like the individual removal algorithm.

Algorithm	Mean Updates	Mean Time (s)	Mean Calls	Mean Inputs
RVE (k_{opt})	5.5×10^9	359.9	81.1	10.0
$r_{max} = 20$	6.5×10^9	500.7	123.8	10.8
$r_{max} = 40$	8.0×10^9	603.8	151.3	10.2
$r_{max} = 60$	9.3×10^9	678.8	169.0	10.0
$r_{max} = 80$	10.0×10^9	694.7	172.3	10.0
$r_{max} = 100$	11.7×10^9	740.7	184.1	9.9
RVE ($k = 1$)	12.7×10^9	644.7	138.7	10.0

Table 4: Results of RVE and RVE*r*S for several values of r_{max}. Mean calls refers to the number calls made to the learning algorithm. Mean inputs refers to the number of inputs used by the final hypothesis.

With respect to the analysis presented in Section 4.2.1, note that the constants c_1 and c_2 do not impact the total cost of performing variable selection. However, a large number of adjustments to r_{min} and r_{max} do impact the total cost negatively.

6.2 An Experimental Comparison of RVE and RVE*r*S

The RVE*r*S algorithm was applied to the seven-of-ten problems using the same conditions as the experiments with RVE. Table 4 shows the results of running RVE*r*S based on five values of r_{max} and $r_{min} = 2$. The results show that for increasing values of r_{max}, the performance of RVE*r*S degrades slowly with respect to cost. The difference between RVE*r*S with $r_{max} = 100$ and RVE with $k = 1$ is significant at the 95% confidence level ($p = 0.049$), as is the difference between RVE*r*S with $r_{max} = 20$ and RVE with $k = k_{opt}$ ($p = 0.0005$). However, this slow degradation does not hold in terms of run time or number of calls to the learning algorithm. Here, only versions of RVE*r*S with $r_{max} = 20$ or 40 show an improvement over RVE with $k = 1$.

The RVE*r*S algorithm termination criteria causes the sharp increase in the number of calls to the learning algorithm. Recall that as n approaches r the probability of a failed selection increases. This means that the number of allowable selection failures grows as the algorithm nears completion. Thus, the RVE*r*S algorithm makes many calls to the learner using a small number of inputs n in an attempt to determine whether the search should be terminated. The search for r compounds the effect. If, at the end of the search, the irrelevant variables have been removed but r_{min} and r_{max} have not converged, then the algorithm must work through several failed sequences in order to terminate.

Figure 3 plots of the number of variables selected compared to the average total number of weight updates for $r_{max} = 20$, 60 and 100. The error bars represent the standard deviation in the number of updates. Notice the jump in the number of updates required for the algorithm to reach completion (represented by number of inputs equals ten) compared to the number of updates required to reach twenty remaining inputs. This pattern does not appear in the results of either version of the RVE algorithm shown in Figure 2. Traces of the RVE*r*S algorithm support the conclusion that many calls to the learner are needed to reach termination even after the correct set of variables has been found.

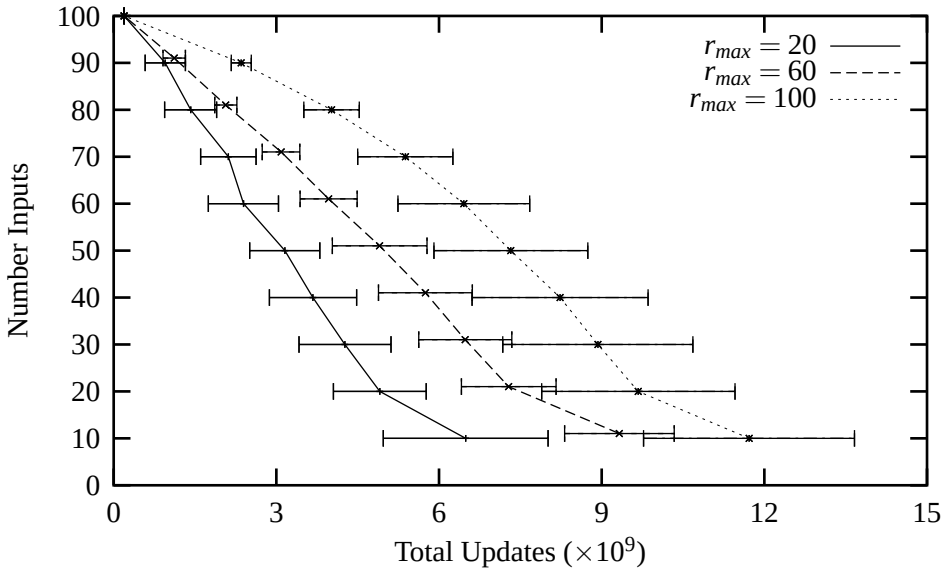

Figure 3: A comparison between the number of inputs on which the thermal perceptrons are trained and the aggregate number of updates performed using the RVE*r*S algorithm.

The increase in run times follows directly from the increasing number of calls to the learner. The thermal perceptron algorithm carries a great deal of overhead not reflected by the number of updates. Since the algorithm executes for a fixed number of epochs, the run time of any call to the learner will contribute noticeably to the run time of RVE*r*S, regardless of the number of selected variables. Contrast this behavior to that of learner whose cost is based more firmly on the number of input variables, such as naive Bayes. Thus, even though RVE*r*S always requires fewer weight updates than RVE with $k = 1$, the latter may still run faster.

This result suggests that the termination criterion of the RVE*r*S algorithm is flawed. The large number of calls to the learner at the end of the variable elimination process wastes a portion of the advantage generated earlier in the search. More importantly, the excess number of calls to the learner does not respect the very careful search trajectory computed by the cost function. Although our cost function for the learner $M(\mathcal{L}, n)$ does take the overhead of the thermal perceptron algorithm into account, there is no allowance for unnecessary calls to the learner. Future research with randomized variable elimination should therefore include a better termination criterion.

7. Experiments with RVE*r*S

We now examine the general performance properties of randomized variable elimination via experiments with several data sets. The previous experiments with perceptrons on the seven-of-ten problem focused on performance with respect to the cost metric. The following experiments are concerned primarily with minimizing run time and the number of calls to the learner while maintaining or improving accuracy. All tests were run on a 3.12 GHz Intel Xenon processor.

Data Set	Variables	Classes	Train Size	Test Size	Values of r_{max}
internet-ad	1558	2	3279	CV	1558, 750, 100
mult. feature	240	10	2000	CV	240, 150, 50
DNA	180	3	2000	1186	180, 100, 50
LED	150	10	2000	CV	150, 75, 25
opt-digits	64	10	3823	1797	64, 40, 25
soybean	35	19	683	CV	35, 25, 15
sick-euthyroid	25	2	3164	CV	25, 18, 10
monks-2-local	17	2	169	432	17, 10, 5

Table 5: Summary of data sets.

Unlike the linearly-separable perceptron experiments, the problems used here do not necessarily have solutions with zero test error. The learning algorithms may produce hypotheses with more variance in accuracy, requiring a more sophisticated evaluation function. The utility of variable selection with respect to even the most sophisticated learning algorithms is well known, see for example Kohavi and John (1997) or Weston, Mukherjee, Chapelle, Pontil, Poggio, and Vapnik (2000). The goal here is to show that our comparatively liberal elimination method sacrifices little in terms of accuracy and gains much in terms of speed.

7.1 Learning Algorithms

The RVE*r*S algorithm was applied to two learning algorithms. The first is the C4.5 release 8 algorithm (Quinlan, 1993) for decision tree induction with options to avoid pruning and early stopping. We avoid pruning and early stopping because these are forms of variable selection, and may obscure the performance of RVE*r*S. The cost metric for C4.5 is based on the number of calls to the gain-ratio data purity criterion. The cost of inducing a tree is therefore roughly quadratic in the number of variables: one call per variable, per decision node, with at most a linear number of nodes in the tree. Recall that an exact metric is not needed, only the order with respect to the number of variables must be correct.

The second learning algorithm used is naive Bayes, implemented as described by Mitchell (1997). Here, the cost metric is based on the number of operations required to build the conditional probability table, and is therefore linear in the number of inputs. In practice, these tables need not be recomputed for each new selection of variables, as the irrelevant table entries can simply be ignored. However, we recompute the tables here to illustrate the general case in which the learning algorithm must start from scratch.

7.2 Data Sets

A total of eight data sets were selected. Table 5 summarizes the data sets, and documentation is generally available from the UCI repository (Blake and Merz, 1998), except for the DNA problem, which is from StatLog (King et al., 1995). The first five problems reflect a preference for data with an abundance of variables and a large number of instances in order to demonstrate the efficiency of RVE*r*S. The last three problems are included to show how RVE*r*S performs on smaller problems, and to allow comparison with other work in variable selection. Further tests on smaller data sets

are possible, but not instructive, as randomized elimination is not intended for data sets with few variables.

Three of the data sets (DNA, opt-digits, and monks) include predetermined training and test sets. The remaining problems used ten-fold cross validation. The version of the LED problem used here was generated using code available at the repository, and includes a corruption of 10% of the class labels. Following Kohavi and John, the monks-2 data used here includes a local (one of n) encoding for each of the original six variables for a total of 17 Boolean variables. The original monks-2 problem contains no irrelevant variables, while the encoded version contains six irrelevant variables.

7.3 Methodology

For each data set and each of the two learning algorithms (C4.5 and naive Bayes), we ran four versions of the RVE*r*S algorithm. Three versions of RVE*r*S use different values of r_{max} in order to show how the choice of r_{max} affects performance. The fourth version is equivalent to RVE with $k = 1$ using a stopping criterion based on the number of consecutive failures (as in RVE*r*S). This measures the performance of removing variables individually given that the number of relevant variables is completely unknown. For comparison, we also ran forward step-wise selection, backward step-wise elimination and a hybrid filtering algorithm. The filtering algorithm simply ranked the variables by gain-ratio, executed the learner using the first 1, 2, 3, ..., N variables, and selected the best.

The learning algorithms used here provide no performance guarantees, and may produce highly variable results depending on variable selections and available data. All seven selection algorithms therefore perform five-fold cross-validation using the training data to obtain an average hypothesis accuracy generated by the learner for *each selection of variables*. The methods proposed by Kohavi and John (1997) could be used to improve error estimates for cases in which the variance in hypothesis error rates is high. Their method should provide reliable estimates for adjusting the values of r_{min} and r_{max} regardless of learning algorithm.

Preliminary experiments indicated that the RVE*r*S algorithm is more prone to becoming bogged down during the selection process than deterministic algorithms. We therefore set a small tolerance (0.002) as shown in Table 3, which allows the algorithm to keep only very good selections of variables while still preventing the selection process from stalling unnecessarily. We have not performed extensive tests to determine ideal tolerance values.

The final selections produced by the algorithms were evaluated in one of two ways. Domains for which there no specific test set is provided were evaluated via ten-fold cross-validation. The remaining domains used the provided training and test sets. In the second case, we ran each of the four RVE*r*S versions five times in order to smooth out any fluctuations due to the random nature of the algorithm.

7.4 Results

Tables 6–9 summarize the results of running the RVE*r*S algorithm on the given data sets using naive Bayes and C4.5 for learning algorithms. In the tables, *iters* denotes the number of search iterations, *evals* denotes the number of subset evaluations performed, *inputs* denotes the size of the final set of selected variables, *error* rates include the standard deviation where applicable, and *cost* represents the total cost of the search with respect to the learner's cost metric. The first row in each block shows the performance of the learner prior to variable selection, while the remaining rows show

Data Set	Learning Algorithm	Selection Algorithm	Iters	Subset Evals	Inputs	Percent Error	Time (sec)	Search Cost
internet	Bayes				1558	3.0±0.9	0.5	4.61×10^{6}
		$r_{max}=100$	137	137	37.8	3.0±1.2	165	6.13×10^{8}
		$r_{max}=750$	536	536	9.2	3.2±1.2	790	3.99×10^{9}
		$r_{max}=1558$	845	845	17.5	2.9±0.8	1406	8.26×10^{9}
		$k=1$	1658	1658	8.8	3.0±1.2	2685	1.46×10^{10}
		forward	20	30810	18.9	2.5±0.8	22417	5.32×10^{9}
		backward				NA		
		filter	1558	1558	837	3.1±0.9	2614	1.44×10^{10}
internet	C4.5				1558	3.0±0.8	48	2.04×10^{5}
		$r_{max}=100$	340	340	33.9	3.3±1.0	5386	3.30×10^{7}
		$r_{max}=750$	1233	1233	25.7	3.7±1.2	40656	2.61×10^{8}
		$r_{max}=1558$	1489	1489	20.0	3.2±1.3	78508	5.02×10^{8}
		$k=1$	1761	1761	20.6	3.3±1.0	91204	6.02×10^{8}
		forward	19	28647	17.5	3.2±1.2	18388	1.95×10^{7}
		backward				NA		
		filter	1558	1558	640	3.1±1.0	77608	3.98×10^{8}
mult-ftr	Bayes				240	34.1±4.5	0.1	4.51×10^{5}
		$r_{max}=50$	53	53	18.8	18.3±2.0	13	3.09×10^{7}
		$r_{max}=150$	84	84	19.4	17.5±4.7	27	7.28×10^{7}
		$r_{max}=240$	112	112	19.9	17.5±2.2	41	1.13×10^{8}
		$k=1$	341	341	17.2	15.7±3.0	99	2.57×10^{8}
		forward	20	4539	18.7	12.3±1.6	527	7.55×10^{8}
		backward	186	27323	55.6	13.9±1.7	12097	3.52×10^{10}
		filter	240	240	53.6	22.5±2.7	83	2.30×10^{8}
mult-ftr	C4.5				240	22.0±4.0	0.6	3.74×10^{4}
		$r_{max}=50$	306	306	22.3	22.1±2.0	241	1.13×10^{7}
		$r_{max}=150$	459	459	21.3	20.2±2.7	427	2.12×10^{7}
		$r_{max}=240$	474	474	22.0	22.1±3.5	519	2.66×10^{7}
		$k=1$	460	460	22.9	20.5±2.5	523	2.71×10^{7}
		forward	26	5960	25.3	20.4±3.5	2004	5.06×10^{7}
		backward	151	24722	90.8	20.4±3.1	51018	2.90×10^{9}
		filter	240	240	140	21.2±2.7	354	1.93×10^{7}

Table 6: Variable selection results using the naive Bayes and C4.5 learning algorithms.

the performance of the seven selection algorithms. Finally, "NA" indicates that the experiment was terminated due to excessive computational cost.

The performance of RVE*r*S on the five largest data sets is encouraging. In most cases RVE*r*S was comparable to the performance of step-wise selection with respect to generalization, while requiring substantially less computation. This effect is most clear in the mult-ftr data set, where forward selection with the C4.5 learner required nearly six CPU days to run (for ten-fold cross-validation) while the slowest RVE*r*S version required just six hours. An exception to this trend occurs with the internet-ad data using C4.5. Here, the huge cost of running C4.5 with most of the variables included overwhelms RVE*r*S's ability to eliminate variables quickly. Only the most aggressive run of the algorithm, with $r_{max}=100$, manages to bypass the problem.

Data Set	Learning Algorithm	Selection Algorithm	Iters	Subset Evals	Inputs	Percent Error	Time (sec)	Search Cost
DNA	Bayes				180	6.7	0.08	3.63×10^5
		$r_{max}=50$	359	359	24.2	4.7±0.7	52	1.52×10^8
		$r_{max}=100$	495	495	30.0	4.9±0.8	75	2.39×10^8
		$r_{max}=180$	519	519	25.6	5.0±0.5	84	2.89×10^8
		$k=1$	469	469	23.6	4.7±0.3	76	2.56×10^8
		forward	19	3249	18.0	5.8	269	2.99×10^8
		backward	34	5413	148	6.5	1399	7.16×10^9
		filter	180	180	101	5.7	32	1.33×10^8
DNA	C4.5				180	9.7	0.5	1.95×10^4
		$r_{max}=50$	356	356	17.0	8.1±1.7	198	8.42×10^6
		$r_{max}=100$	384	384	16.2	7.1±1.5	222	9.07×10^6
		$r_{max}=180$	432	432	13.8	6.5±1.2	282	1.21×10^7
		$k=1$	374	374	14.4	6.5±1.1	274	1.18×10^7
		forward	13	2262	12.0	5.9	418	2.33×10^6
		backward	110	13735	72.0	8.7	18186	8.23×10^8
		filter	180	180	17.0	7.6	163	7.10×10^6
LED	Bayes				150	30.3±3.0	0.09	2.75×10^5
		$r_{max}=25$	127	127	22.7	26.9±3.9	19	3.97×10^7
		$r_{max}=75$	293	293	17.4	26.0±3.3	50	1.09×10^8
		$r_{max}=150$	434	434	25.6	25.9±2.6	86	2.02×10^8
		$k=1$	423	423	23.7	27.0±2.1	85	2.04×10^8
		forward	14	2006	13.0	26.6±2.9	141	1.54×10^8
		backward	14	1870	138.0	30.1±2.6	667	1.95×10^9
		filter	150	150	23.7	27.1±2.1	34	8.49×10^7
LED	C4.5				150	43.9±4.5	0.5	5.48×10^4
		$r_{max}=25$	85	85	51.1	42.0±3.0	89	1.01×10^7
		$r_{max}=75$	468	468	25.8	42.5±4.5	363	3.70×10^7
		$r_{max}=150$	541	541	25.2	40.8±5.7	440	4.48×10^7
		$k=1$	510	510	32.4	42.5±2.7	439	4.63×10^7
		forward	9	1286	7.8	27.0±3.2	196	9.52×10^5
		backward	61	7218	90.9	43.5±3.5	11481	1.33×10^9
		filter	150	150	7.1	27.3±3.5	156	1.69×10^7

Table 7: Variable selection results using the naive Bayes and C4.5 learning algorithms.

The internet-ad via C4.5 experiment highlights a second point. Notice how the forward selection algorithm runs faster than all but one version of RVE*r*S. In this case, the cost and time of running C4.5 many times on a small number of variables is less than that of running C4.5 few times on many variables. However, note that a slight change in the number of iterations needed by the forward algorithm would change the time and cost of the search dramatically. This is not the case for RVE*r*S, since each iteration involves only a single evaluation instead of $O(N)$ evaluations.

The number of subset evaluations made by RVE*r*S is also important. Notice the growth in number of evaluations with respect to the total (initial) number of inputs. For aggressive versions of RVE*r*S, growth is very slow, while more conservative versions, such as $k=1$ grow approximately linearly. This suggests that the theoretical results discussed for RVE remain valid for RVE*r*S. Addi-

Data Set	Learning Algorithm	Selection Algorithm	Iters	Subset Evals	Inputs	Percent Error	Time (sec)	Search Cost
opt-digits	Bayes				64	17.4	0.08	2.59×10^5
		$r_{max}=25$	111	111	14.2	15.7±1.5	15.9	4.05×10^7
		$r_{max}=40$	157	157	13.2	14.9±0.7	22.9	5.92×10^7
		$r_{max}=64$	162	162	14.4	14.7±1.0	24.9	6.66×10^7
		$k=1$	150	150	14.0	14.2±1.1	24.7	6.76×10^7
		forward	17	952	16.0	14.1	93.8	1.91×10^8
		backward	41	1781	25.0	13.5	423.0	1.39×10^9
		filter	64	64	37.0	16.1	11.9	3.63×10^7
opt-digits	C4.5				64	43.2	0.7	2.15×10^4
		$r_{max}=25$	130	130	12.0	42.4±2.0	148	3.64×10^6
		$r_{max}=40$	158	158	10.8	42.2±0.3	181	4.42×10^6
		$r_{max}=64$	216	216	10.4	42.5±1.2	253	6.36×10^6
		$k=1$	140	140	11.4	42.1±1.1	189	5.33×10^6
		forward	16	904	15.0	41.6	645	9.64×10^6
		backward	28	1378	38.0	44.0	2842	9.55×10^7
		filter	64	64	50.0	43.6	87	2.12×10^6
soybean	Bayes				35	7.8±2.4	0.02	2.40×10^4
		$r_{max}=15$	142	142	12.6	8.9±4.2	5.9	6.47×10^6
		$r_{max}=25$	135	135	11.9	10.5±5.8	5.8	6.26×10^6
		$r_{max}=35$	132	132	11.2	9.8±5.1	5.8	6.17×10^6
		$k=1$	88	88	12.3	9.6±5.0	4.6	4.67×10^6
		forward	13	382	12.3	7.3±2.9	8.9	1.09×10^7
		backward	19	472	18.0	7.9±4.6	37.5	3.63×10^7
		filter	35	35	31.3	7.8±2.6	2.0	1.97×10^6
	C4.5				35	8.6±4.0	0.04	1.21×10^3
		$r_{max}=15$	118	118	16.3	9.5±4.6	13.5	2.78×10^5
		$r_{max}=25$	158	158	14.7	10.1±4.1	18.6	1.90×10^5
		$r_{max}=35$	139	139	16.3	9.1±3.7	17.3	3.86×10^5
		$k=1$	117	117	16.1	9.3±3.5	14.9	3.52×10^5
		forward	16	435	14.8	9.1±4.0	33.7	3.22×10^5
		backward	18	455	19.1	10.4±4.4	69.0	1.75×10^6
		filter	35	35	30.8	8.5±3.7	3.7	6.06×10^4

Table 8: Variable selection results using the naive Bayes and C4.5 learning algorithms.

tional tests using data with many hundreds or thousands of variables would be instructive, but may not be feasible with respect to the deterministic search algorithms.

RVE*r*S does not achieve the same economy of subset evaluations on the three smaller problems as on the larger problems. This is not surprising, since the ratio of relevant variables to total variables is much smaller, requiring RVE*r*S to proceed more cautiously. In these cases, the value of r_{max} has only a minor effect on performance, as RVE*r*S is unable to remove more than two or three variables in any given step.

One problem evidenced by both large and small data sets is that there appears to be no clear choice of a best value for r_{max}. Conservative versions of RVE*r*S tend to produce lower error rates,

Data Set	Learning Algorithm	Selection Algorithm	Iters	Subset Evals	Inputs	Percent Error	Time (sec)	Search Cost
euthyroid	Bayes				25	6.2±1.3	0.02	7.54×10^4
		$r_{max}=10$	30	30	2.0	4.6±1.5	2.1	2.65×10^6
		$r_{max}=18$	39	39	2.0	4.8±1.2	1.3	3.73×10^6
		$r_{max}=25$	46	46	2.3	5.1±1.4	1.6	4.32×10^6
		$k=1$	35	35	1.7	5.0±1.2	1.5	4.86×10^6
		forward	5	118	4.2	4.6±1.1	3.4	6.45×10^6
		backward	16	263	11.3	4.2±1.3	14.4	5.91×10^7
		filter	25	25	4.7	4.2±0.6	1.4	4.16×10^6
	C4.5				25	2.7±1.0	0.2	1.00×10^3
		$r_{max}=10$	49	49	2.9	2.4±0.8	21.0	2.98×10^4
		$r_{max}=18$	63	63	3.3	2.2±0.7	27.6	4.58×10^4
		$r_{max}=25$	55	55	2.7	2.5±0.8	25.3	4.92×10^4
		$k=1$	54	54	3.8	2.3±0.9	29.4	6.39×10^4
		forward	7	151	5.9	2.4±0.7	51.8	3.89×10^4
		backward	16	269	11.0	2.5±0.9	200.0	6.73×10^5
		filter	25	25	15.2	2.7±1.1	15.4	3.60×10^4
monks-2	Bayes				17	39.4	0.01	3.11×10^3
		$r_{max}=5$	25	25	2.6	36.1±3.2	0.02	1.10×10^5
		$r_{max}=10$	54	54	4.0	37.2±2.3	0.05	2.50×10^5
		$r_{max}=17$	74	74	6.0	37.4±3.1	0.08	4.93×10^5
		$k=1$	41	41	6.0	36.8±3.1	0.04	2.89×10^5
		forward	2	33	1.0	32.9	0.02	6.70×10^4
		backward	8	99	11.0	38.4	0.13	9.93×10^5
		filter	17	17	2.0	40.3	0.01	1.21×10^5
	C4.5				17	23.6	0.03	5.14×10^2
		$r_{max}=5$	36	36	8.2	16.7±10.9	0.8	2.34×10^4
		$r_{max}=10$	84	84	6.2	4.6±0.4	1.9	3.74×10^4
		$r_{max}=17$	79	79	6.4	6.5±4.7	1.8	4.63×10^4
		$k=1$	55	55	6.2	4.4±0.0	1.4	4.13×10^4
		forward	2	33	1.0	32.9	0.6	3.95×10^2
		backward	13	139	6.0	4.4	3.9	1.61×10^5
		filter	17	17	13.0	35.6	0.4	1.51×10^4

Table 9: Variable selection results using the naive Bayes and C4.5 learning algorithms.

but there are exceptions. In some cases, r_{max} has very little effect on error. However, in most cases, small values of r_{max} have a distinct positive effect on run time.

The results suggest two other somewhat surprising conclusions. One is that backward elimination does not appear to have the commonly assumed positive effect on generalization. Step-wise forward selection tends to outperform step-wise backward elimination, although randomization often reduces this effect. The second conclusion is that the hybrid filter algorithm performs well in some cases, but worse than RVE*r*S and step-wise selection in most cases. Notice also that for problems with many variables, RVE*r*S runs as fast or faster than the filter. Additional experiments along these lines would be instructive.

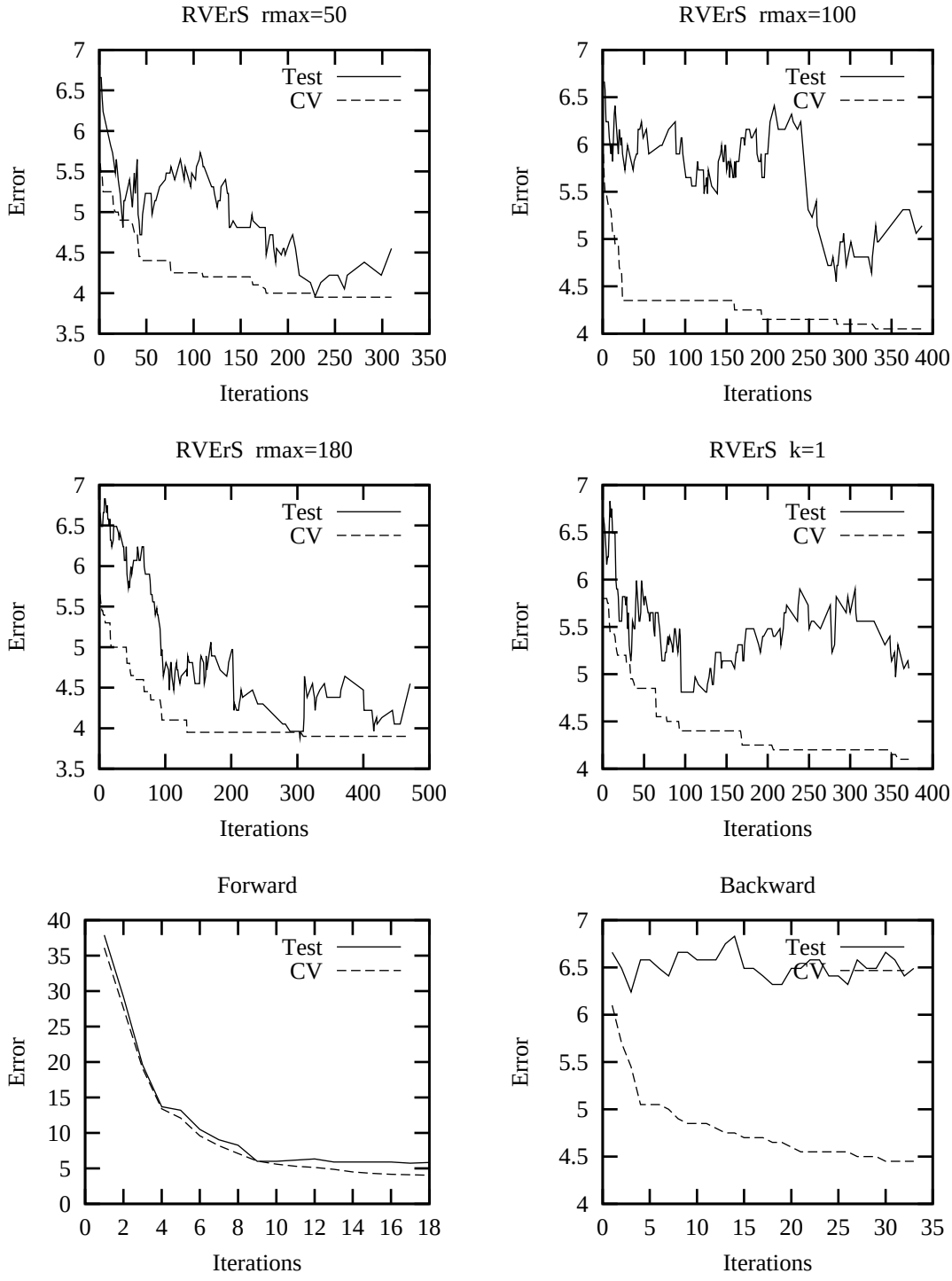

Figure 4: Naive Bayes overfitting plots for DNA data.

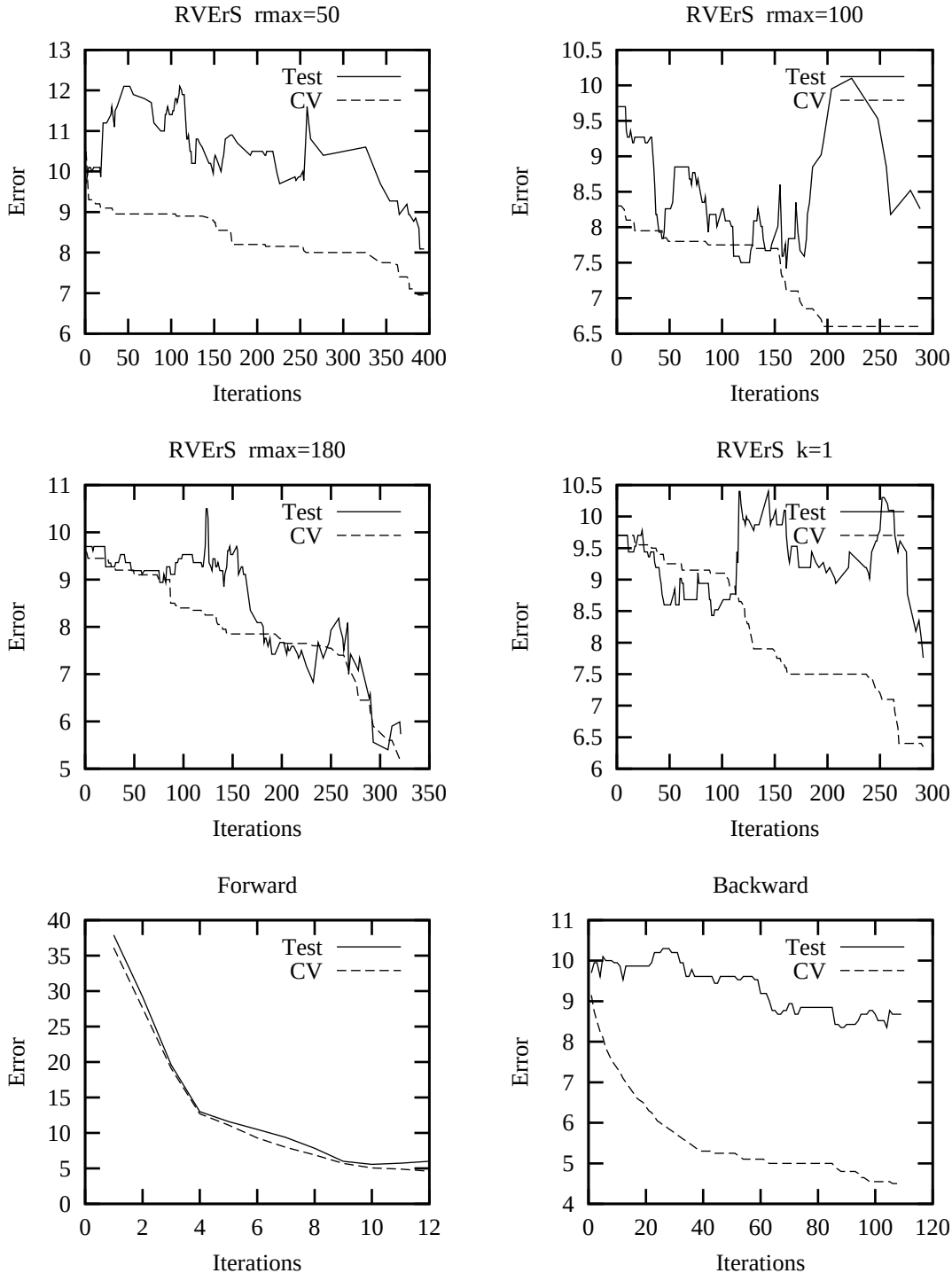

Figure 5: C4.5 overfitting plots for DNA data.

Overfitting is sometimes a problem with greedy variable selection algorithms. Figures 4 and 5 show both the test and inner (training) cross-validation error rates for the selection algorithms on naive Bayes and C4.5 respectively. Solid lines indicate test error, while dashed lines indicate the inner cross-validation error. Notice that the test error is not always minimized with the final selections produced by RVE*r*S. The graphs show that RVE*r*S does tend to overfit naive Bayes, but not C4.5 (or at least to a lesser extent). Trace data from the other data sets agree with this conclusion.

There are at least two possible explanations for overfitting by RVE*r*S. One is that the tolerance level either causes the algorithm to continue eliminating variables when it should stop, or allows elimination of relevant variables. In either case, a better adjusted tolerance level should improve performance. The monks-2 data set provides an example. In this case, if the tolerance is set to zero, RVE*r*S reliably finds variable subsets that produce low-error hypotheses with C4.5.

A second explanation is that the stopping criteria, which becomes more difficult to satisfy as the algorithm progresses, causes the elimination process to become overzealous. In this case the solution may be to augment the given stop criteria with a hold-out data set (in addition to the validation set). Here the algorithm monitors performance in addition to counting consecutive failures, returning the best selection, rather than simply the last. Combining this overfitting result with the above performance results suggests that RVE*r*S is capable of performing quite well with respect to both generalization and speed.

8. Discussion

The speed of randomized variable elimination stems from two aspects of the algorithm. One is the use of large steps in moving through the search space of variable sets. As the number of irrelevant variables grows, and the probability of selecting a relevant variable at random shrinks, RVE attempts to take larger steps toward its goal of identifying all of the *irrelevant* variables. In the face of many irrelevant variables, this is a much easier task than attempting to identify the relevant variables.

The second source of speed in RVE is the approach of removing variables immediately, instead of finding the best variable (or set) to remove. This is much less conservative than the approach taken by step-wise algorithms, and accounts for much of the benefit of RVE. In practice, the full benefit of removing multiple variables simultaneously may only be beginning to materialize in the data sets used here. However, we expect that as domains scale up, multiple selections will become increasingly important. One example of this occurs in the STL algorithm (Utgoff and Stracuzzi, 2002), which learns many concepts over a period of time. There, the number of available input variables grows as more concepts are learned by the system.

Consider briefly the cost of forward selection wrapper algorithms. Greedy step-wise search is bounded by $O(rNM(\mathcal{L},r))$ for forward selection and $O(N(N-r)M(\mathcal{L},N))$ for backward elimination, provided it does not backtrack or remove (or add) previously added (or removed) variables. The bound on the backward approach reflects both the larger number of steps required to remove the irrelevant variables and the larger number of variables used at each call to the learner. The cost of training each hypothesis is small in the forward greedy approach compared to RVE, since the number of inputs to any given hypothesis is much smaller (bounded roughly by r). However, the number of calls to the learning algorithm is polynomial in N. As the number of irrelevant variables increases, even a forward greedy approach to variable selection becomes quickly unmanageable.

The cost of a best-first search using compound operators (Kohavi and John, 1997) is somewhat harder to analyze. Their approach combines the two best operators (e.g. add variable or remove

variable) and then checks whether the result is an improvement. If so, the resulting operator is combined with the next best operator and tested, continuing until there is no improvement. Theoretically this type of search could find a solution using approximately $2r$ forward evaluations or $2(N-r)$ backward subset evaluations. However, this would require the algorithm to make the correct choice at every step. The experimental results (Kohavi and John, 1997) suggest that in practice the algorithm requires many more subset evaluations than this minimum.

Compare the above bounds on forward and backward greedy search to that of RVE given a fixed $k = 1$, which is $O(rNM(\mathcal{L}, N))$. Notice that the number of calls to the learning algorithm is the same for RVE with fixed k and a greedy *forward* search (the cost of learning is different however). The empirical results support the conclusion that the two algorithms produce similar cost, but also show that RVE with $k = 1$ requires less CPU time. The source of this additional economy is unclear, although it may be related to various overhead costs associated with the learning algorithms. RVE requires many fewer total learner executions, thereby reducing overhead.

In practice, the $k = 1$ version of RVE*r*S often makes fewer than rN calls to the learning algorithm. This follows from the very high probability of a successful selection of an irrelevant variable at each step. In cases when N is much larger than r, the algorithm with $k = 1$ makes roughly N calls to the learner as shown in Tables 6 and 7. Additional economy may also be possible when k is fixed at one. Each variable should only need to be tested once, allowing RVE*r*S to make exactly N calls to the learner. Further experiments are needed to confirm this intuition.

Although the RVE algorithm using a fixed $k = 1$ is significantly more expensive than the optimal RVE or RVE*r*S using a good guess for r_{max}, experiments and analysis show that this simple algorithm is generally faster than the deterministic forward or backward approaches, provided that there are enough irrelevant variables in the domain. As the ratio r/N decreases, and the probability of selecting an irrelevant variable at random increases, the benefit of a randomized approach improves. Thus, even when no information about the number of relevant variables is available, a randomized, backward approach to variable selection may be beneficial.

A disadvantage to randomized variable selection is that there is no clear way to recover from poor choices. Step-wise selection algorithms sometimes consider both adding and removing variables at each step, so that no variable is ever permanently selected or eliminated. A hybrid version of RVE*r*S which considers adding a single variable each time a set a variables is eliminated is possible, but this would ultimately negate much of the algorithm's computational benefit.

Step-wise selection algorithms are sometimes parallelized in order to speed the selection process. This is due in large part to the very high cost of step-wise selection. RVE mitigates this problem to a point, but there is no obvious way to parallelize a randomized selection algorithm. Parallelization could be used to improve generalization performance by allowing the algorithm to evaluate several subsets simultaneously and then choose the best.

9. Future Work

There are at least three possible directions for future work with RVE. The first is an improved method for choosing k when r is unknown. We have presented an algorithm based on a binary search, but RVE*r*S still wastes a great deal of time deciding when to terminate the search, and can quickly degenerate into a one-at-a-time removal strategy if bad decisions are made early in the search. Notice however, that this worst-case performance is still better than stepwise backward elimination, and comparable to stepwise forward selection, both popular algorithms.

A second direction for future work involves further study of the effect of testing very few of the possible successors to the current search node. Testing all possible successors is the source of the high cost of most wrapper methods. If a sparse search, such as that used by RVE, does not sacrifice much quality in general, then other fast wrapper algorithms may be possible.

A third possible direction involves biasing the random selections at each step. If a set of k variables fails to maintain evaluation performance, then at least one of the k must have been relevant to the learning problem. Thus, variables included in a failed selection may be viewed as more likely to be relevant. This "relevance likelihood" can be tracked throughout the elimination process and used to bias selections at each step.

10. Conclusion

The randomized variable elimination algorithm uses a two-step process to remove irrelevant input variables. First, a sequence of values for k, the number input variables to remove at each step, is computed such that the cost of removing all $N - r$ irrelevant variables is minimized. The algorithm then removes the irrelevant variables by randomly selecting inputs for removal according to the computed schedule. Each step is verified by generating and testing a hypothesis to ensure that the new hypothesis is at least as good as the existing hypothesis. A randomized approach to variable elimination that simultaneously removes multiple inputs produces a factor N speed-up over approaches that remove inputs individually, provided that the number r of relevant variables is known in advance.

When number of relevant variables is not known, a search for r may be conducted in parallel with the search for irrelevant variables. Although this approach wastes some of the benefits generated by the theoretical algorithm, a reasonable upper bound on the number of relevant variables still produces good performance. When even this weaker condition cannot be satisfied, a randomized approach may still outperform the conventional deterministic wrapper approaches provided that the number of relevant variables is small compared to the total number of variables. A randomized approach to variable selection is therefore applicable whenever the target domain is believed to have many irrelevant variables.

Finally, we conclude that an explicit search through the space of variable subsets is not necessary to achieve good performance from a wrapper algorithm. Randomized variable elimination provides competitive performance without incurring the high cost of expanding and evaluating all successors of a search node. As a result, randomized variable elimination scales well beyond current wrapper algorithms for variable selection.

Acknowledgments

The authors thank Bill Hesse for his advice concerning the analysis of RVE. This material is based upon work supported by the National Science Foundation under Grant No. 0097218. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

D. W. Aha and R. L. Bankert. Feature selection for case-based classification of cloud types: An empirical comparision. In *Working Notes of the AAAI-94 Workshop on Case-Based Reasoning*, pages 106–112, Seattle, WA, 1994. AAAI Press.

H. Almuallim and T. G. Dietterich. Learning with many irrelevant features. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, CA, 1991. MIT Press.

C. L. Blake and C. J. Merz. Uci repository of machine learning databases. Technical report, University of California, Department of Information and Computer Science, 1998.

C. Cardie. Using decision trees to improve case-based learning. In *Machine Learning: Proceedings of the Tenth International Conference*, Amherst, MA, 1993. Morgan Kaufmann.

R. Caruana and D. Freitag. Greedy attribute selection. In *Machine Learning: Proceedings of the Eleventh International Conference*, New Brunswick, NJ, 1994. Morgan Kaufmann.

K. J. Cherkauer and J. W. Shavlik. Growing simpler decision trees to facilitate knowledge discovery. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1996.

W. W. Cooley and P. R. Lohnes. *Multivariate data analysis*. Wiley, New York, 1971.

P. A. Devijver and J. Kittler. *Pattern recognition: A statistical approach*. Prentice Hall/International, 1982.

P. Domingos. Context sensitive feature selection for lazy learners. *Artificial Intelligence Review*, 11:227–253, 1997.

G. H. Dunteman. *Principal components analysis*. Sage Publications, Inc., Newbury Park CA, 1989.

B. Efron, T. Hastie, I. Johnstone, and R. Tibsharini. Least angle regression. Technical Report TR-220, Stanford University, Department of Statistics, 2003.

S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. *Advances in Neural Information Processing Systems*, 2:524–532, 1990.

W. Finnoff, F. Hergert, and H. G. Zimmermann. Improving model selection by nonconvergent methods. *Neural Networks*, 6:771–783, 1993.

M. Frean. A "thermal" perceptron learning rule. *Neural Computation*, 4(6):946–957, 1992.

J. H. Friedman and J. W. Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, C-23(9):881–889, 1974.

S. I. Gallant. Perceptron-based learning. *IEEE Transactions on Neural Networks*, 1(2):179–191, 1990.

D. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Reading, MA, 1989.

M. A. Hall. *Correlation-based feature selection for machine learning*. PhD thesis, Department of Computer Science, University of Waikato, Hamilton, New Zealand, 1999.

B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems 5*. Morgan Kaufmann, 1993.

I. Inza, P. Larranaga, R. Etxeberria, and B. Sierra. Feature subset selection by Bayesian network-based optimization. *Artificial Intelligence*, 123(1-2):157–184, 2000.

G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 121–129, New Brunswick, NJ, 1994. Morgan Kaufmann.

R. Kerber. Chimerge: Discretization of numeric attributes. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 123–128, San Jose, CA, 1992. MIT Press.

R. King, C. Feng, and A. Shutherland. Statlog: Comparison of classification algorithms on large real-world problems. *Applied Artificial Intelligence*, 9(3):259–287, 1995.

K. Kira and L. Rendell. A practical approach to feature selection. In D. Sleeman and P. Edwards, editors, *Machine Learning: Proceedings of the Ninth International Conference*, San Mateo, CA, 1992. Morgan Kaufmann.

J. Kivinen and M. K. Warmuth. Additive versus exponentiated gradient updates for linear prediction. *Information and Computation*, 132(1):1–64, 1997.

R. Kohavi. *Wrappers for performance enhancement and oblivious decision graphs*. PhD thesis, Department of Computer Science, Stanford University, Stanford, CA, 1995.

R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2): 273–324, 1997.

D. Koller and M. Sahami. Toward optimal feature selection. In *Machine Learning: Proceedings of the Fourteenth International Conference*, pages 284–292. Morgan Kaufmann, 1996.

M. Kubat, D. Flotzinger, and G. Pfurtscheller. Discovering patterns in Eeg signals: Comparative study of a few methods. In *Proceedings of the European Conference on Machine Learning*, pages 367–371, 1993.

P. Langley and S. Sage. Oblivious decision trees and abstract cases. In *Working Notes of the AAAI-94 Workshop on Case-Based Reasoning*, Seattle, WA, 1994. AAAI Press.

Y. LeCun, J. Denker, and S. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems 2*, pages 598–605, San Mateo, CA, 1990. Morgan Kaufmann.

N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 1988.

H. Liu and R. Setino. A probabilistic approach to feature selection: A filter solution. In *Machine Learning: Proceedings of the Fourteenth International Conference*. Morgan Kaufmann, 1996.

H. Liu and R. Setiono. Feature selection via discretization. *IEEE Transaction on Knowledge and Data Engineering*, 9(4):642–645, 1997.

O. Maron and A. W. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In *Advances in Neural Information Processing Systems*, volume 6. Morgan Kaufmann, 1994.

M. Mitchell. *An introduction to genetic algorithms*. MIT Press, Cambridge, MA, 1996.

M. Mitchell, T. *Machine learning*. MIT Press, 1997.

J. Moody and J. Utans. Architecture selection for neural networks: Application to corporate bond rating prediction. In A. N. Refenes, editor, *Neural Networks in the Capital Markets*. John Wiley and Sons, 1995.

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

J. R. Quinlan. *C4.5: Machine learning programs*. Morgan Kaufmann, 1993.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and orginization in the brain. *Psychological Review*, 65:386–407, 1958.

D. E. Rumelhart and J. L. McClelland. *Parallel distributed processing*. MIT Press, Cambridge, MA, 1986. 2 volumes.

L. Thurstone. Multivariate data analysis. *Psychological Review*, 38:406–427, 1931.

P. E. Utgoff and D. J. Stracuzzi. Many-layered learning. *Neural Computation*, 14(10):2497–2529, 2002.

H. Vafaie and K. De Jong. Genetic algorithms as a tool for restructuring feature space representations. In *Proceedings of the International Conference on Tools with AI*. IEEE Computer Society Press, 1995.

L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

P. Werbos. Backpropagation: Past and future. In *IEEE International Conference on Neural Networks*. IEEE Press, 1988.

J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik. Feature selection for SVMs. In *Advances in Neural Information Processing Systems 13*, pages 668–674. MIT Press, 2000.