Randomized Variable Elimination

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

Abstract

Variable selection, or the process of identifying input variables that are relevant to a particular learning problem, has recently received much attention in the learning community. Methods that employ the learning algorithm as a part of the selection process (wrappers) have been shown to outperform methods that select variables independent of the learning algorithm (filters), but only at great computational expense. We present a randomized wrapper algorithm for variable elimination that runs in time only a constant factor greater than that of simply learning in the presence of all input variables, provided that the cost of learning grows at least polynomially with the number of inputs.

1. Introduction

When learning in a supervised environment, a learning algorithm is typically presented with a set of N-dimensional data points, along with the associated target outputs. The learning algorithm is then required to output a hypothesis describing the function underlying the data. In practice, the set of N input variables is carefully selected in order to improve the performance of the learning algorithm. However, 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), Falhman and Lebiere (1990), and Kivinen and Warmuth (1997) all present examples of algorithms that create a potentially large number of features during the learning process. In these situations, an automatic approach to variable selection is required.

Kohavi and John (1997) discuss an approach to finding variables that are relevant to the target function with respect to a given learning algorithm. Their *wrapper* model performs a best-first search in the space of variable subsets. The search may proceed either as a forward-selection, starting with the empty set, or a backward-elimination, starting with the set of all available variables. Operators in the search generally include adding or removing a single variable from the current set, although compound operators are also possible. Each set of variables is evaluated by running the learning algorithm and comparing the performance of the resulting hypotheses. Although this approach is capable of producing a minimal set of input variables, the cost grows exponentially in the face of many irrelevant variables. A greedy approach (Caruana & Freitag, 1994) may be more practical than a best-first search, but this may ultimately include irrelevant variables while excluding relevant variables from the final set.

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 is limited. The main computational cost associated with wrapper methods for variable selection is usually that of executing the learning algorithm. The learner must produce a hypothesis for each potential set of input variables. Even greedy selection methods that ignore large areas of the search space can produce a large number of input sets as the number of irrelevant variables grows.

Randomized variable elimination attempts to avoid the cost of evaluating many variable sets by taking the largest steps possible through the space of possible input sets. As the number of irrelevant variables increases, the size of the step also increases. 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 backward elimination approach to simplify the detection of relevant variables. Removal of any relevant variable should immediately cause the learner's performance to degrade.

Analysis of our cost function shows that the cost of removing all irrelevant variables is 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 notable improvement compared to the factor N increase experienced in removing inputs individually.

2. 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_i that have not been marked as irrelevant. We assume that the learning algorithm may be instructed to ignore the variables that have been marked irrelevant. Alternatively, we could assume the presence of a preprocessor that removes all variables marked irrelevant from x_i, presenting only the remaining variables to the learning algorithm.

We make the assumption that the number r of relevant variables is at least two and is small compared to the total number of variables N. This assumption 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 guessing r while removing variables.

3. 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 total 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 inputs. Note that n represents the number of remaining variables, while N denotes the total number of variables in the original problem.

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 during the definition and analysis of the cost function.

3.1 Definition

The first step of defining the cost function is to calculate the probability $p^+(n,r,k)$ of successfully selecting k irrelevant inputs at random and without replacement, given that there are n total and r relevant inputs. We define this probability as

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

Applying now the negative binomial distribution, let

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

represent the expected number of failures at selecting k irrelevant variables from n total given that r are relevant. $E^-(n,r,k)$ therefore yields the expected number of trials in which at least one of the r relevant variables will be randomly selected along with irrelevant variables prior to success.

We now define the cost of selecting and removing k variables, given n and r. Let $M(\mathcal{L},n)$ represent a 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. Assumptions about the nature of the learning problem should be accounted for by the bounding function. 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 definitions and discussion assume a fixed algorithm for $\mathcal{L}$. The cost of successfully removing k variables from n total given that r are relevant is given by

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

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

Given this cost of removing k variables, we can now define recursively the cost of removing all $n-r$ irrelevant variables. The goal is to minimize locally the cost of removing k inputs with respect to the remaining cost, resulting in a global minimum cost for removing all $n-r$ irrelevant variables. The use of a greedy minimization step relies on 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. Let

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

represent the cost (with respect to learning algorithm $\mathcal{L}$) of removing $n-r$ irrelevant variables. The first term in the equation represents the cost of removing the first k variables while the second term 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. We define

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

3.2 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 $M(\mathcal{L}, n)$ 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.

3.2.1 Removing Multiple Variables

We now show informally that the above bounds on the performance of the RVE algorithm hold. 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 computation is a linear program, 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.

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

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

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, $\frac{\log(n)-\log(r)}{\log(1+1/(r-1))} \leq r\log(n)$, 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 cost of running RVE in these cases is therefore bounded by $I_{sum}(n, r) \leq O(M(\mathcal{L}, n))$ (factors of r may be ignored as n will always be larger). When $M(\mathcal{L}, n)$ is sub-linear in n (e.g logarithmic), each pass of the algorithm will contribute significantly to the total cost, resulting in a bound of $O(\log(n)M(\mathcal{L}, n))$.

Table 1. Algorithm for computing k and cost values.

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 2. Randomized backward-elimination variable selection 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

3.2.2 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. The bound on the cost of RVE is therefore $I_{sum}(n,r) \leq O(nM(\mathcal{L},n))$ when $k=1$. This is true regardless of whether the variables are selected randomly or deterministicly at each step.

3.3 Computing the Cost and k-Sequence

The equations for $I_{sum}(n,r)$ and $k_{opt}(n,r)$ suggest a simple $O(n^2)$ linear program for computing both the cost and optimal k-sequence for a problem of n variables with r relevant. 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 values of n, and bootstraps these to find values for increasingly large n. The function $I(n,r,k)$ in Table 1 is computed as described above.

4. The Randomized Variable Elimination Algorithm

The algorithm begins by first computing the values of $k_{opt}(i,r)$ for all $r+1 \leq i \leq n$. Next an initial hypothesis based on all n input variables is generated. 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 hypothesis has improved upon the error of the previous hypothesis (possibly within a given tolerance), then the selected k inputs are marked as irrelevant and 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.

5. An Application of RVE

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 perceptron. The specific task examined here is to learn a Boolean function that is true when seven out of ten relevant inputs are true, given a total of 100 input variables. In order to ensure that the hypotheses generated for each selection of variables has minimal error, we use the thermal perceptron training algorithm (Frean, 1992). The pocket algorithm (Gallant, 1990) is also applicable, but we found this to be slower and less reliable.

Twenty problems were generated randomly with $N =$

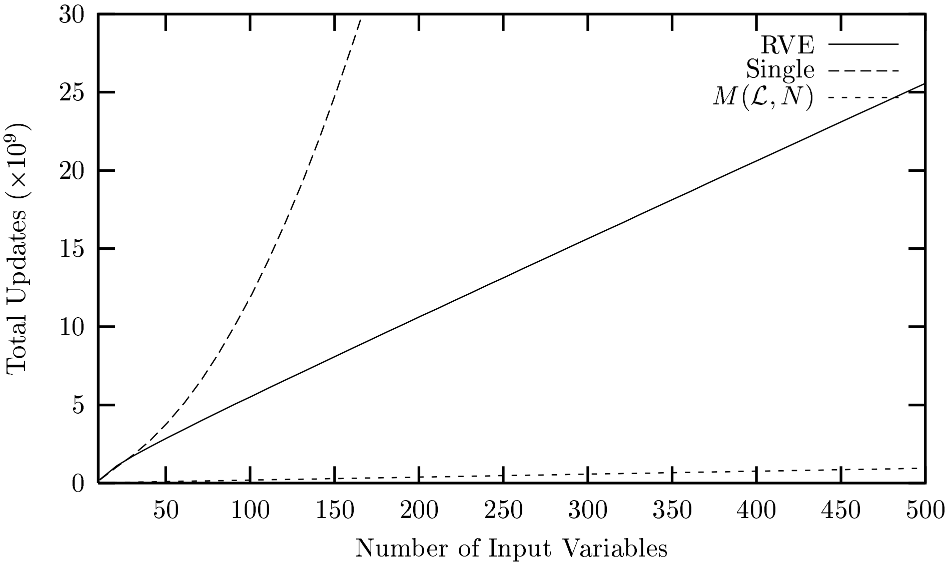

Figure 1. Plot of the cost of running RVE ($I_{sum}(N, r = 10)$) along with the cost of removing inputs individually, and the estimated number of updates $M(\mathcal{L}, N)$.

100 input variables, of which $r = 10$ are relevant. 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 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 formula for the number of updates needed by an N-input perceptrons, 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, while the cost of removing variables individually grows as N^2 and begins to explode around 150 input variables. This matches perfectly 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 as r increases.

Once the table for $k_{opt}(n, r)$ has been created, the process of selecting and removing variables can begin. 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 necessarily produce a minimum error hypothesis.

The RVE algorithm was run using the twenty problems described above. Hypotheses based on only ten variables were produced in an average of 214.8 seconds on a 1.13 GHz Pentium III, using an average of 5.5×10^9 weight updates. A version of the RVE algorithm which removes variables individually (i.e. k was set permanently to 1) was also run, and produced hypotheses based on only ten variables in an average of 443 seconds with 12.5×10^9 weight updates. The number of weight updates required to remove all $N - r$ irrelevant inputs agrees with the estimate produced by the cost function. Interestingly, 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) that had un-

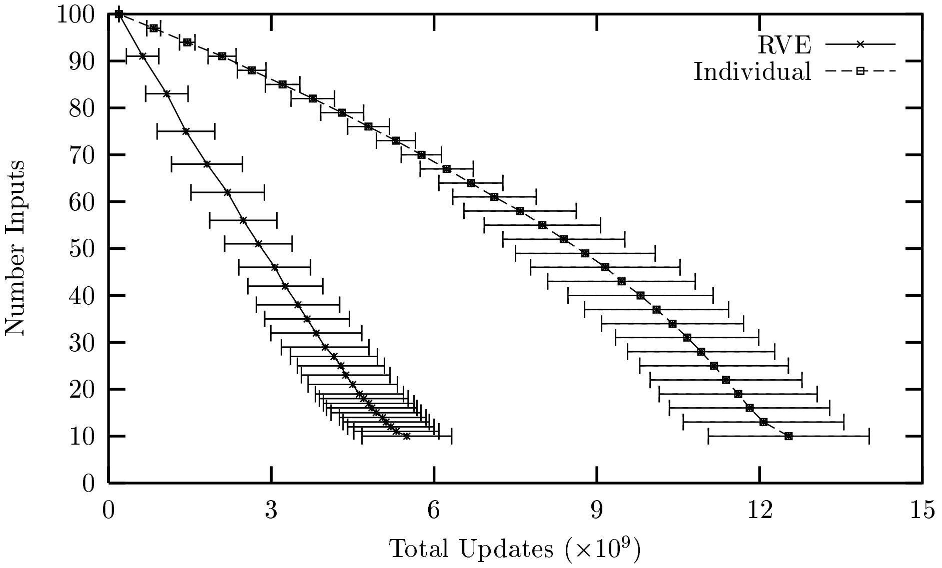

Figure 2. A comparison between the number of inputs on which the perceptrons are trained and the mean aggregate number of updates performed by the perceptrons.

usually high error (error greater than 0.18 or approximately 45 mis-classified instances).

Figure 2 shows a plot of 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. Notice both the rate of drop in inputs, and the number of selections (and therefore the number of hypotheses trained), for the RVE algorithm compared to removing variables individually. This reflects the balance between the cost of training and unsuccessful variable selections. Removing variables individually in the presence of many irrelevant variables does not account for 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 r of relevant variables is known has played a critical role in preceding discussion. In practice, this is a very strong assumption that is not easily met. We would like an algorithm that can remove irrelevant attributes efficiently even though the number of relevant variables is not known in advance. One approach would be to simply guess a value for r and see how RVE fares. This is unsatisfying however, as a poor guess can destroy the efficiency of RVE.

Guessing a single, specific value for r is a difficult task, but placing a loose bound around maximum and minimum values for r may be much easier. Applications such as pattern recognition may produce a large number of potential inputs for a simple problem because the specific few that are actually relevant are unknown. In such a task, the maximum value for r may be known to be much less than N. In the worst case, 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. 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 pre-

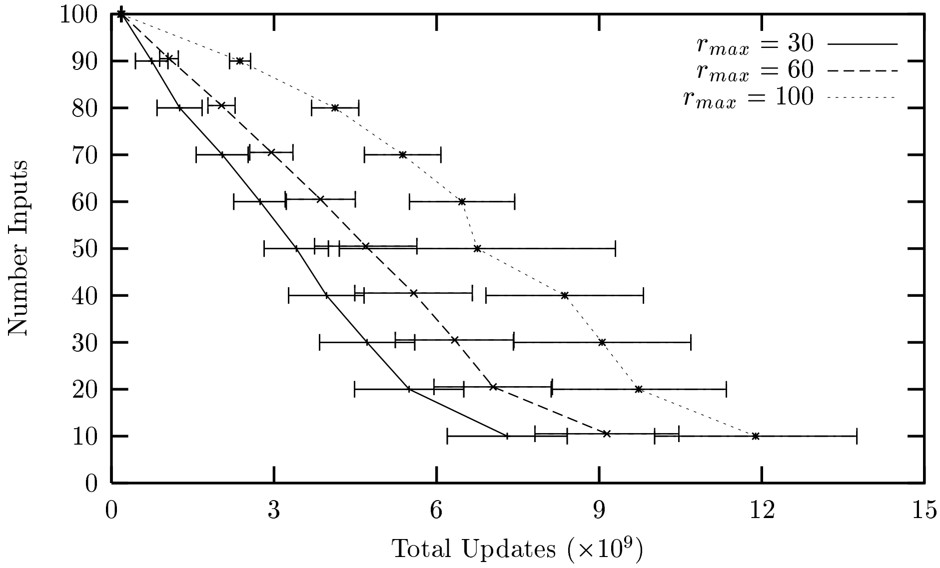

Figure 3. A comparison between the number of inputs on which the perceptrons are trained and the aggregate number of updates performed using the RVErS algorithm.

scribed 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, but more extensive tests are needed.

The algorithm for 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 number of variables to remove at each selection is chosen according to the current value of r. Each time the bound on the maximum number of successful selections is exceeded, r_{max} is reduced to r and r is recalculated as $\frac{r_{max}+r_{min}}{2}$. Similarly, when the bound on consecutive failures is exceeded, r_{min} is increased to r and r is recalculated. A check is also needed 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 terminates when r_{min} and r_{max} have converged and $cE^-(n,r,k)$ consecutive variable selections have failed.

The RVErS algorithm was applied to the seven-of-ten problems using the same conditions as the experiments with RVE. Figure 3 shows a plot of the number of variables used by the perceptron compared to the average total number of weight updates at certain times for $r_{max} = 30$, 60 and 100. The value for r_{min} was set at 2 for all three trials. When $r_{max} = 30$ RVErS produces results quite similar to those of RVE, with an average of 7.3×10^9 updates and requiring an average of 296.7 seconds. As r_{max} increases however, the performance of RVErS degrades slowly until it is roughly the same as removing inputs individually. All three versions found the correct variable subsets for most of the problems. As with RVE, cases in which the initial hypothesis had high error produced incorrect variable subsets, including too many or too few variables.

Analysis of RVErS is complex. Although the algorithm can produce good performance without finding the exact value of r, how close the estimated value must come to the actual value is not clear. A more important factor in determining the complexity of RVErS is the question of 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 quickly, and the bound on complexity for RVE will apply. In the worst case, the search for r will proceed slowly over values of r that are too high, causing RVErS to behave like the individual removal algorithm.

7. Discussion

The strength of randomized variable elimination stems from 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.

Consider briefly the cost of forward-selection wrapper algorithms. In the case that the algorithm performs an optimal search, such as best-first, the cost is ultimately exponential in N, and clearly becomes intractable for all but small values of N. The cost of a greedy search is bounded by $O(rNM(\mathcal{L},r))$ for forward selection and $O(N^2M(\mathcal{L},r))$ for backward selection, provided it does not backtrack or remove previously added variables. 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 greedy approach to variable selection becomes quickly unmanageable.

The Las Vegas Filter algorithm (LVF) (Liu & Setino, 1996) is an approach to variable selection in which variable subsets are generated independently at random during each round of execution. If the size of the subset is smaller than the current best and has a lower inconsistency rate, then the new subset replaces the current best. The inconsistency rate is defined as the total number of inconsistencies in the data (number of instances that are equivalent given the selected variables but have different labels) divided by the total number of instances. LVF is similar to RVE in that both algorithms are randomized, but while LVF makes selections totally at random (both set size and membership), RVE performs its search along a calculated trajectory.

8. 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. The assumption that the number of relevant inputs r is known raises several issues, but the overall result is positive. A randomized approach to variable elimination that simultaneously removes multiple inputs produces a factor N speed-up over approaches that remove inputs individually.

Acknowledgments

This work was supported by NSF Grant IRI-0097218. The authors thank Bill Hesse and Neil Immerman for their help in analysing the general cost function. We also thank Jen Neville for her suggestion of a binary search when r is unknown, and Tino Tamon for helpful comments.

References

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

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

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

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

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

Kohavi, R., & John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence, 97*, 273-324.

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

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

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

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