In Armand Prieditis & Stuart Russell, eds., *Machine Learning: Proceedings of the Twelfth International Conference*, Morgan Kaufmann Publishers, San Francisco, CA.

# Automatic Parameter Selection by Minimizing Estimated Error

**Ron Kohavi**
Computer Science Dept.
Stanford University
Stanford, CA 94305
ronnyk@CS.Stanford.EDU
http://robotics.stanford.edu/~ronnyk

**George H. John**
Computer Science Dept.
Stanford University
Stanford, CA 94305
gjohn@CS.Stanford.EDU
http://robotics.stanford.edu/~gjohn

## Abstract

We address the problem of finding the parameter settings that will result in optimal performance of a given learning algorithm using a particular dataset as training data. We describe a "wrapper" method, considering determination of the best parameters as a discrete function optimization problem. The method uses best-first search and cross-validation to wrap around the basic induction algorithm: the search explores the space of parameter values, running the basic algorithm many times on training and holdout sets produced by cross-validation to get an estimate of the expected error of each parameter setting. Thus, the final selected parameter settings are tuned for the specific induction algorithm and dataset being studied. We report experiments with this method on 33 datasets selected from the UCI and StatLog collections using C4.5 as the basic induction algorithm. At a 90% confidence level, our method improves the performance of C4.5 on nine domains, degrades performance on one, and is statistically indistinguishable from C4.5 on the rest. On the sample of datasets used for comparison, our method yields an average 13% relative decrease in error rate. We expect to see similar performance improvements when using our method with other machine learning algorithms.

## 1 Introduction

A user of machine learning algorithms must decide not only which algorithm to use on a particular dataset, but also what parameter values to use for the chosen algorithm. The user wants to choose the algorithm and parameters that result in the best future performance. Although the former problem of selecting a learning algorithm for a particular task is recognized as an important issue in machine learning (Brazdil, Gama & Henery 1994, Schaffer 1993), the latter problem of finding the best parameter values has not been systematically studied. Nearly all machine learning algorithms have parameters to be set by a user, and the setting of these parameters can have a large effect on the accuracy of the induced models.

Selecting an algorithm and finding the best parameters for a particular algorithm are equivalent in the abstract, but the latter involves optimization over a much larger (possibly continuous) space, one that cannot be completely explored within reasonable time limits. Thus, issues of heuristic search arise that are not present in the algorithm selection problem.

Authors of machine learning algorithms typically give some rules of thumb regarding the setting of their algorithm's parameters or supply default parameter settings that they expect to work well on the problems of interest to the user. Although such heuristics may apply reasonably well to a variety of datasets, a better strategy would be to arrive at parameter values that work well for the particular dataset under analysis.

To determine the parameter setting, the selection method must take into account the interaction between the biases of the induction algorithm (Mitchell 1982) and the particular training set available. Intuitively, an algorithm should be able to make use of the information in the training data to guide it in a search for the best parameters. This idea motivated us to consider the wrapper method of John, Kohavi & Pfleger (1994), which does employ such information. In the wrapper method, the parameter selection algorithm exists as a wrapper around the induction algorithm (see Figure 1), conducting a search for a good parameter setting using the induction algorithm itself as part of the evaluation function.

For any induction algorithm $\mathcal{A}$, we may define an al-

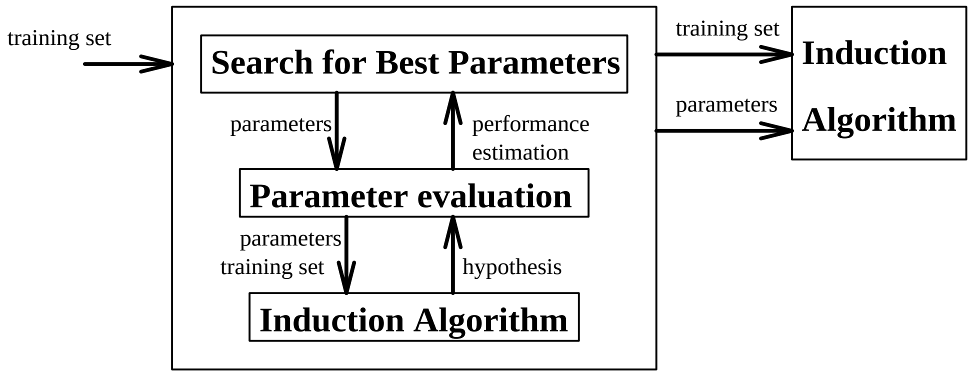


Figure 1: The Wrapper method for learning algorithm parameter selection. The algorithm itself is used to select the parameters.

Table 1: Notation used in the rest of the paper.

| Symbol | Meaning |
|---|---|
| $\vec{x} \in \mathcal{X}$ | An instance. A vector of attribute (or feature) values. |
| $y \in \mathcal{Y}$ | The label or output value of an instance. |
| $(\vec{x}, y) \in \mathcal{X} \times \mathcal{Y}$ | A training instance. |
| $f \in \mathcal{F}$ | A target function mapping instances to output values. $f : \mathcal{X} \mapsto \mathcal{Y}$. $f$ may be probabilistic, in which case we will be concerned with $E(f\|\vec{x})$ instead of $f(x)$ when we discuss expected error. |
| $h \in \mathcal{H}$ | A hypothesis function mapping instances to output values. $h : \mathcal{X} \mapsto \mathcal{Y}$. |
| $T \subseteq \mathcal{T}$ | A training set. $\mathcal{T} = \{(\vec{x}, y) \| \vec{x} \in \mathcal{X}, y = f(\vec{x})\}$. |
| $\theta \in \Theta$ | A vector of parameters for a machine learning algorithm. |
| $\mathcal{A}(T, \theta)$ | A machine learning algorithm. Input: a training set and parameter vector. Output: a function. $\mathcal{A} : (\mathcal{T} \times \Theta) \mapsto \mathcal{H}$. |
| $L(y_1, y_2)$ | A loss function. $L : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}$. For the classification problems considered here, $L(y, y') = 0$ if $y = y'$, 1 otherwise. |
| $\mathrm{err}(h, f)$ | The error of a function $h$ with respect to $f$. Given some probability distribution $P_X$ over $\mathcal{X}$, $\mathrm{err}(h) = \int_{\vec{x}} L(h(\vec{x}), f(\vec{x})) P_X(\vec{x}) d\vec{x}$. |

gorithm $\mathcal{A}$-AP that, when given a set of training data, uses the wrapper to choose parameters and then runs $\mathcal{A}$ on the training data using the chosen parameters. The *bias* of the learning algorithm $\mathcal{A}$ is thus carried into the $\mathcal{A}$-AP algorithm but modified by the wrapper's choice of parameters.

In Section 2, we formally define the problem we wish to solve: finding the paramters that minimize expected loss. Section 3 discusses cross-validation and best-first search, the two components of our wrapper. Section 4 gives an example of our approach, describing how the C4.5-AP algorithm uses a wrapper around C4.5. Section 5 discusses our experimental methodology and results comparing C4.5 to C4.5-AP. We also discuss lessons about C4.5 learned from C4.5-AP's choices of parameters. Section 6 discusses related work in machine learning and statistics. Finally, Section 7 gives our conclusions and suggested directions for further work.

## 2 The Parameter Selection Problem

The task of the induction algorithm $\mathcal{A}$ is to take a training set $T$ and induce a function $h$ (*e.g.*, a decision tree or neural network) which minimizes $err(h, f)$ (see Table 1). The output of $\mathcal{A}$ is determined by both $T$ and the parameter vector $\theta$. We wish to find the parameter vector $\theta^*$ that will result in minimal expected loss, given a distribution over $X$, a learning algorithm $\mathcal{A}$, and training set $T$. Formally, the optimal parameter setting $\theta^*$ satisfies

$$\theta^* = \underset{\theta \in \Theta}{\operatorname{argmin}} \, err(\mathcal{A}(T, \theta), f) \quad , \tag{1}$$

that is, $\theta^*$ is the element in $\Theta$ which results in minimal error.

Finding $\theta^*$ satisfying Equation 1 is impossible because we do not know $P_X(\vec{x})$ or $f$, which are required to calculate $err$. The only information we have about the distribution of X is the training set $T$. Thus, we must somehow estimate $err$ using $T$. Even if we did know $P_X(\vec{x})$, as a practical problem $\Theta$ may be too big to search exhaustively. When some elements of $\theta$ are

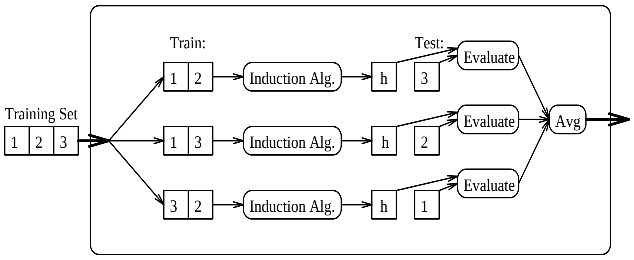


Figure 2: Cross-validation methodology (3-fold CV shown).

real-valued, or when $\theta$ has many discrete-valued elements of high cardinality, it is impractical to search the entire space $\Theta$. Since both obstacles prevent us from finding $\theta^*$, we must instead be content to find some $\widehat{\theta}$ approximating $\theta^*$ which also results in low expected loss.

In the next section we discuss error estimation method for estimating *err*, and a heuristic search method for exploring only a subset of $\Theta$.

## 3 The Wrapper Method

Figure 1 depicts the wrapper method, which requires two components: a search component and an evaluation component. The search component repeatedly suggests parameter settings. The evaluation component evaluates these settings by running the induction algorithm several times and getting an estimate of the resulting accuracy. (In some domains, users may also be concerned about the interpretability of the induced models. In such cases some interpretability metric should be included in the evaluation component.) In this section we describe the error estimation and search algorithms used.

### 3.1 Error Estimation by Cross-Validation

Given a learning algorithm $\mathcal{A}$ with parameters $\theta$ and a training set $T$, we need to estimate $err(\mathcal{A}(T,\theta),f))$. *Error estimation* methods give approximations to *err* without requiring knowledge of $P_X$. We use cross-validation (Stone 1974, Breiman, Friedman, Olshen & Stone 1984, Weiss & Kulikowski 1991, Kohavi 1995*b*) to estimate *err* and denote the estimate by $err_{\text{CV}}$.

Figure 2 depicts how 3-fold cross-validation calculates $err_{\text{CV}}(\mathcal{A},\theta,T)$. In $k$-fold CV, one partitions the original training set $T$ into $k$ subsets $T_i$, such that each instance $(\vec{x}_i, y_i)$ in $T$ appears in exactly one subset and all subsets have cardinality approximately $|T|/k$. $err_{\text{CV}}$ is defined as

$$err_{\text{CV}}(\mathcal{A},\theta,T) = \frac{1}{k}\sum_{i=1}^{k} err_{\text{h}}(\mathcal{A}(T-T_i,\theta),T_i) \ , \quad (2)$$

where $err_{\text{h}}(h,T) = \frac{1}{|T|}\sum_{i=1}^{|T|} L(h(x_i),y_i)$, the error on an independent holdout set of instances. The appeal of the cross-validation approach is that all instances are used $k-1$ times as training instances and once as test instances, so that no data are wasted.

One problem with using cross-validation as an error estimate for parameter selection is that it has high variance. Thus, we must use some care in using $err_{\text{CV}}$ to select a parameter setting. We should not select one parameter setting over another without being reasonably confident that we are making the correct choice. Letting $c_i = err_{\text{h}}(\mathcal{A}(T-T_i,\theta),T_i)$, then $err_{\text{CV}} = \bar{c}$. The variance of $err_{\text{CV}}$ is $\frac{1}{k}\frac{1}{k-1}\sum_{i=1}^{k}(c_i-\bar{c})^2$. Since the variance of the mean of a sample is inversely proportional to the number of samples, we may achieve lower variance of the mean of $err_{\text{CV}}$, by repeating the entire cross-validation procedure $t$ times. Let $c_{i,j}$ be the holdout error on the $i$th fold of the $j$th run of cross-validation. Then $err_{\text{CV}} = \bar{c} = \frac{1}{tk}\sum_{j=1}^{t}\sum_{i=1}^{k} c_{i,j}.$ , and the variance of $err_{\text{CV}}$ is $\frac{1}{tk}\frac{1}{tk-1}\sum_{j=1}^{t}\sum_{i=1}^{k}(c_{i,j}-\bar{c})^2$. (Since the $kt$ samples of $C$ are not independent, this variance is likely to be optimistic.) Thus, at a computational cost (running CV $t$ times) we can increase the chance (by reducing the variance of our estimate by $1/t$) that the final selected parameters will result in low expected error.

### 3.2 Optimization using Best-First Search

The problem of finding the best value for each parameter of a given learning algorithm is a function optimization problem. We chose to view the problem as state-space search (Kohavi 1994). We chose the best-first search algorithm (Ginsberg 1993, Nilsson 1980), which works by repeatedly expanding the most promising unexpanded state (Table 2). (Note that this is not simply hill-climbing.) Search problems can be characterized by five distinct components. We describe these components below, describing how each is involved in our wrapper method.

- **State Space $\mathcal{S}$** The set of states explored by the search. The goal of the search is to find the $s^* \in \mathcal{S}$ satisfying some set of desired properties. Here, $\mathcal{S} \subseteq \Theta$, so we either search the entire space of parameters or some subset.
- **Heuristic Function $f : \mathcal{S} \mapsto \mathbb{R}$** A function mapping a state to the real numbers. We use $f(s)$ as both the *cost* of the state $s$, which is to be minimized, and as the heuristic function. Here, $f = err_{\text{CV}}$.

Table 2: The Best-First Search algorithm, modified for function optimization

```
Best-First-Search
   1. Put the initial state on the OPEN list, CLOSED list <- empty, BEST <- initial state
   2. Let v = the state in OPEN with minimal f(v).  Remove v from OPEN, add v to CLOSED.
   3. if f(v) + epsilon < f(BEST),  BEST <- v
   4. Expand v: Apply all operators to v.  Evaluate all resulting states, add each
      to OPEN list unless already in CLOSED list
   5. If no change in f(BEST) during the last k expansions, exit and return BEST
   6. Goto 2
```

- **Operators $op : \mathcal{S} \mapsto \mathcal{S}$** A function mapping states into states. States and operators form a graph with states represented by nodes and operators labeling the directed arcs between nodes. Here, the operators transform one parameter setting to another.
- **Initial State $s_0$** The initial state, or starting point of the search.
- **Termination Condition** The criterion for stopping the search. In our wrapper method, the desired property is that $s = \mathrm{argmin}_{\mathcal{S}} err_{\mathrm{CV}}(\mathcal{A}, s, T)$, but this condition cannot be efficiently evaluated. Instead we stop and accept $s$ as the best state if some reasonably thorough further search in the space does not uncover a better state: when $k$ consecutive expansions fail to yield a state $s'$ that is at least $\epsilon$ better than the current best state $s$. All function optimization methods must specify some stopping criterion, and ours is similar to those used in numerical optimization methods (Press, Teukolsky, Vetterling & Flannery 1992).

## 4 Automatic Parameter Selection for C4.5

To make our ideas more concrete, we describe the application of our method to the C4.5 decision tree algorithm (Quinlan 1993). C4.5 is an extremely robust algorithm that performs well on a wide variety of domains. It is very difficult to consistently outperform C4.5 on a variety of datasets. Thus, improving C4.5 should yield an interesting learning algorithm. Our C4.5-AP algorithm is a "wrapper" around Quinlan's C4.5 algorithm (Figure 1).

Up to this point we have specified the wrapper as much as possible independent of any learning algorithm. The rest of this section details choices we made that were specific to the C4.5 algorithm.

- **State Space** We chose to automatically set all of the C4.5 tree-building parameters ($m$,$c$,$g$, and $s$) shown in Table 3.

  For example, a parameter vector $\theta$ might be $(10, 25, \text{off}, \text{on})$, indicating $m = 10, c = 25, g =$ off (use gain-ratio), $s =$ on (use subset tests). $m$ controls when tree-building is stopped, $c$ is a pruning parameter, $g$ specifies the use of information-gain or gain-ratio during tree construction, and $s$ specifies subset tests on multi-valued nominal attributes. For the $m$ and $c$ parameters we only considered integer values $1-5, 10, 15, 20, 25, 30, 40, \ldots, 100$. This decision was based on our observations that fine granularity was not important for large values of the parameters. There are a total of 1156 states.
- **Heuristic Function** We used 10-fold cross-validation as the estimate of the expected error for each state. When the estimated standard deviation of $err_{\mathrm{CV}}$ is larger than 1%, we run cross-validation again, up to a maximum of three times. We used a 10% trimmed mean (John 1994); that is, if we run 10-fold CV three times, then out of the 30 resulting estimates we remove the lowest and highest 3 and take the average of the rest.
- **Operators** For the binary parameters, we try the opposite values. For each of the numeric parameters, we consider moving up or down circularly either one or five positions in the array of possible values. (*E.g.*, from a current state where $m = 3$ we might try $m = 2, m = 4, m = 80, m = 20$.) In our first version we only moved up or down one position, but found the algorithm getting trapped in local minima. We added the option of moving 5 steps and moving circularly to allow escape from local minima. (No other step sizes were tried.)
- **Initial State** We used the default setting for C4.5: $(2, 25, \text{off}, \text{off})$.
- **Termination Condition** We use the condition described in Section 3.2, setting $k$ to 5 and $\epsilon$ to 0.1%.

## 5 Experiments with C4.5-AP

Our hypothesis is that, given some learning algorithm $\mathcal{A}$ that has parameters, we can use our wrapper method to create an $\mathcal{A}$-AP algorithm which performs better than $\mathcal{A}$ on real-world domains. However,

Table 3: Parameters to the C4.5 algorithm.

| Name | Possible Values | Default Value | Description |
|---|---|---|---|
| `-m`$m$ | $1\ldots\infty$ | 2 | Stopping parameter in tree construction. Halts the recursive partitioning process when no partition of the current node results in children all having weight $\geq m$. (*Weight* is equal to the number of instances unless there are missing values.) |
| `-c`$c$ | $[0, 100]$ | 25 | Confidence level parameter in tree pruning. Small values of $c$ cause much pruning, large values cause little pruning. |
| `-g` | on,off | off | Splitting criterion: information gain or gain ratio. When `-g` is specified, information gain is used as the splitting criterion in tree construction. When not specified, gain ratio is used. |
| `-s` | on,off | off | Subset splits. When `-s` is specified, subset splits are considered during tree construction. When unspecified, all splits on $k$-valued nominal variables result in $k$ children. |

as noted by Schaffer (1994) and Wolpert (1994), we cannot hope to improve $\mathcal{A}$ over *all* domains. We first describe our methodology for testing this hypothesis and then present the results and our analysis.

### 5.1 Methodology

To test our hypothesis, we compared the performance of the C4.5 and C4.5-AP algorithms on 33 datasets gathered from the UCI (Murphy & Aha 1994) and StatLog (Michie, Spiegelhalter & Taylor 1994) collections. The datasets represent all of the available StatLog datasets except the Shuttle database (which was too large), all of the UCI datasets used by Holte (1993), all of the Monks datasets (Thrun et al. 1991), and Corral which is an artificial dataset presented in John et al. (1994).

For some datasets, a single test set was specified by the contributors of the dataset (which includes the entire space for the artificial datasets). In such cases we used the holdout method (defined in Section 3.1) to evaluate C4.5 and C4.5-AP so that results would be comparable to those reported in the literature. For all other datasets, 10-fold cross-validation was used to evaluate C4.5 and C4.5-AP. The exact same folds were used for both algorithms so that the ten resulting performance numbers for C4.5 and C4.5-AP are pairwise comparable. At this point it is best to consider the C4.5-AP algorithm as a black box; although it uses cross-validation itself to set its own parameters, the outer cross-validation estimates we report in this section are completely separate from the inner cross-validation. We are now interested only in comparing the performance of the two algorithms.

### 5.2 Experimental Results

Table 4 gives 10-fold CV estimates of the accuracies for the C4.5 and C4.5-AP algorithms on the natural domains. Table 5 gives results for those datasets with a specified test set. Both tables also give performance for C4.5* "algorithm." These are lower bounds on the performance of C4.5 with optimal parameter settings. C4.5* results were obtained by modifying the C4.5-AP algorithm: instead of using cross-validation as the evaluation method, it uses holdout error on the same test set that we later use to evaluate its performance. The C4.5* results might never be achieved in practice. The point is to show the limits of our approach—even if we were to use the world's greatest error estimation method instead of cross-validation, we could never surpass the C4.5* results. We believe that best first search conducts a reasonably thorough search of the space, and therefore we conjecture that the results of C4.5* cannot be significantly improved upon for any settings of C4.5 parameters.

For each dataset in Table 4, we used a one-tailed paired $t$-test (Casella & Berger 1990) to test the hypothesis that the accuracy of the C4.5-AP algorithm is higher than the C4.5 algorithm (versus the null hypothesis that the algorithms perform equally). When we run 10-fold cross-validation, we get ten accuracy estimates that we average to give the final estimated accuracy. In our experiments we treated each of the ten accuracy estimates as samples from a population, and used these as the estimates to be paired in the $t$-test. The $P(t)$ values in the table should be interpreted as supporting our hypothesis at the $P(t)$ confidence level (*e.g.*, when $P(t) \geq .95$ we may state with 95% confidence that C4.5-AP has higher accuracy than C4.5).

From these tests we conclude with 90% confidence that the C4.5-AP algorithm outperforms the C4.5 algorithm on nine of the 33 datasets, and is outperformed by C4.5 on a single dataset. At a 95% confidence level, C4.5-AP outperforms C4.5 on six datasets, and is never outperformed by C4.5. In the one case where C4.5-AP loses at a 90% confidence level, the

Table 4: Experimental results: Accuracies for the C4.5 and C4.5-AP algorithms and the C4.5* upper bounds, all using 10-fold cross-validation. $P(t)$ gives the probability of the paired $t$-statistic that C4.5-AP has higher accuracy than C4.5 on each dataset.

| Dataset | Size | Accuracy | | | $P(t)$ | C4.5-AP |
|---|---|---|---|---|---|---|
| | | Default C4.5 | C4.5-AP | C4.5* | | Better (90% CI) |
| australian | 690 | 85.36±1.13 | 85.07±1.47 | 88.70±1.58 | .372 | |
| breast | 699 | 95.42±0.70 | 96.29±0.77 | 96.98±0.74 | .961 | ✓ |
| breast-cancer | 286 | 73.87±2.76 | 73.87±2.76 | 77.68±1.90 | .500 | |
| chess | 3196 | 99.50±0.13 | 99.62±0.10 | 99.75±0.08 | .800 | |
| cleve | 303 | 72.30±2.17 | 75.61±2.43 | 84.82±1.42 | .931 | ✓ |
| crx | 690 | 85.94±1.37 | 85.07±1.26 | 88.84±0.87 | .186 | |
| diabetes | 768 | 71.75±1.02 | 75.26±1.26 | 82.03±1.42 | .984 | ✓ |
| german | 1000 | 72.50±1.41 | 72.70±1.65 | 79.42±1.45 | .553 | |
| glass | 214 | 65.48±3.22 | 68.20±2.92 | 76.17±2.69 | .847 | |
| glass2 | 163 | 70.55±2.00 | 74.85±3.94 | 87.61±2.96 | .789 | |
| heart | 270 | 80.00±2.77 | 82.22±2.19 | 86.11±1.90 | .703 | |
| hepatitis | 155 | 80.04±3.65 | 84.50±2.47 | 89.67±2.98 | .924 | ✓ |
| horse-colic | 368 | 85.05±1.16 | 84.24±0.78 | 88.86±0.85 | .139 | |
| hypothyroid | 3163 | 99.11±0.18 | 99.27±0.08 | 99.40±0.21 | .894 | |
| iris | 150 | 95.33±1.42 | 95.33±1.42 | 95.33±1.42 | .500 | |
| labor-neg | 57 | 85.67±3.48 | 80.67±3.87 | 89.00±3.02 | .097 | × |
| lymphography | 148 | 78.38±1.65 | 74.14±3.19 | 86.48±1.41 | .103 | |
| mushroom | 8124 | 100.00±0.00 | 100.00±0.00 | 100.00±0.00 | .500 | |
| pima | 768 | 71.60±1.93 | 76.67±2.05 | 79.55±1.55 | .999 | ✓ |
| segment | 2310 | 96.36±0.33 | 96.80±0.37 | 97.75±0.36 | .846 | |
| sick-euthyroid | 3163 | 97.69±0.25 | 97.63±0.46 | 98.20±0.19 | .444 | |
| soybean | 47 | 100.00±0.00 | 100.00±0.00 | 100.00±0.00 | .500 | |
| tic-tac-toe | 958 | 85.59±1.08 | 93.73±0.52 | 96.34±0.65 | 1.00 | ✓ |
| vehicle | 846 | 69.84±1.77 | 72.44±1.73 | 78.18±0.94 | .973 | ✓ |
| vote | 435 | 95.64±0.52 | 95.41±0.47 | 97.71±0.68 | .172 | |
| vote1 | 435 | 88.02±1.77 | 87.58±1.52 | 92.40±1.20 | .342 | |

labor-negotiation dataset, note that the entire dataset is very small with only 57 instances. Because of the small size, the state evaluations in C4.5-AP had high variance, and the search did not find good parameter values.

With a significance of 0.90 in the t-test, if the algorithms were equivalent in their prediction performance, one would expect three or four significant results, about two for each algorithm. Tables 4 and 5 show ten significant results, nine indicating the performance of C4.5-AP is superior and and only one indicating inferior performance. We therefore conclude that C4.5-AP is indeed significantly superior.

C4.5-AP is often significantly outperformed by C4.5*, so it is important to consider the factors that play a role here. Fortunately, there is only one: the heuristic evaluation function. C4.5-AP and C4.5* are both instantiations of the wrapper method (Figure 1). They share the same search method and induction algorithm, only differing on the evaluation function. C4.5* "cheats" and uses the test set for evaluation and thus its parameters are tuned to fit that single test set later used for evaluation. Since the test sets are small and do not provide a good estimate of the true accuracy of the induction algorithm, the C4.5* accuracy estimates are strongly optimistically biased.

We analyzed the parameters chosen by C4.5*. For the $c$ parameter, in almost all cases a wide range of values yielded the same accuracy. The 15%–35% range was most common (10 datasets). Significantly, the value 25% (the default value in C4.5) was chosen for all datasets except tic-tac-toe (where C4.5-AP halved the error rate of C4.5) and glass2 (where C4.5-AP improved absolute accuracy by over 4%). For the $m$ parameter, 1 was often the best choice, or a tie for best choice among 1–5. Although this tends to create a larger tree that then just gets pruned back, C4.5 sometimes prunes by replacing a node's test with the test at one of its children, so perhaps $m$=1 gives more latitude in the pruning phase. Information-gain (turning the $g$ parameter on) was a big winner on several datasets: vehicle, segment, hypothyroid, heart, and cleve. Turning on the $s$ parameter helped in tic-tac-toe and monk1.

Table 5: Experimental results: Accuracies for C4.5, C4.5-AP, and C4.5* from running on the specified test set. $P(t)$ gives the probability of the paired $t$-statistic that C4.5-AP has higher accuracy than C4.5 on each dataset.

| Dataset | Train/Test Size | Accuracy Default C4.5 | C4.5-AP | C4.5* | $P(t)$ | C4.5-AP Better (90% CI) |
|---|---|---|---|---|---|---|
| corral | 32/129 | 81.2±3.44 | 100.0±0.00 | 100.0±0.00 | 1.0 | ✓ |
| dna | 2000/1186 | 92.3±0.77 | 93.2±0.73 | 93.2±0.73 | .80 | |
| letter | 15000/5000 | 86.8±0.48 | 87.0±0.48 | 87.1±0.48 | .62 | |
| monk1 | 124/432 | 75.7±2.06 | 100.0±0.00 | 100.0±0.00 | 1.0 | ✓ |
| monk2 | 169/432 | 65.0±2.29 | 62.5±2.33 | 82.4±1.83 | .22 | |
| monk3 | 122/432 | 97.2±0.79 | 97.2±0.79 | 100.0±0.00 | .50 | |
| satimage | 4435/2000 | 85.2±0.79 | 86.1±0.77 | 85.2±0.79 | .79 | |

This suggests the possibility that perhaps $m = 1$ is a better default value than $m = 2$, and begs the question as to whether C4.5 with this new default parameter setting would be a more worthy competitor to C4.5-AP. We ran C4.5 with the $m$ parameter set to 1 on the same datasets used above and have gotten worse performance on average as compared with $m = 2$ (83.8% accuracy for $m = 1$ vs. 84.4% for $m = 2$); thus, while $m = 1$ yields significant improvements in some cases, it hurts on others and the improvement cannot be attributed to this single parameter.

The time penalty for C4.5-AP can be large: letting $O(A)$ be the running time of $\mathcal{A}$, the time required to run $k$-fold cross-validation $t$ times is $O(tkA)$. We must do this for each state investigated by our search algorithm, so if we evaluate $n$ nodes in the search, the total time is $O(ntkA)$. In our experiments, $t$ was limited to three, $k$ was 10, and the number of nodes expanded was about 60. For example, on the dna dataset (which took the most time) C4.5 took just under 2 minutes, while C4.5-AP took 6.8 hours.

## 6 Related Work

In statistics, *model selection* refers to the general problem of selecting a learning algorithm $\mathcal{A}$ for a particular task (Linhart & Zucchini 1986). Though the problems are equivalent in the abstract, model selection typically refers to the choice of one from a small set of algorithms, while different issues arise in parameter selection because of the large space to be searched. Nevertheless, because of the parallels between model selection and parameter selection, we might expect to observe some of the same phenomena encountered in model selection. Schaffer (1993) explored the use of cross-validation for learning algorithm selection, and concluded that the selected algorithm was sometimes much better and sometimes worse than any given algorithm, but never much worse.

Selecting values for important parameters is a widely-studied problem in statistics. Breiman et al. (1984) describe the CART program, which is probably the prime example of the automatic setting of a parameter (the cost-complexity parameter) in decision tree induction. CART uses 10-fold cross-validation to set this parameter. There are still many parameters left to be set by the user, however, and it would be interesting to compare a fully automated CART to the standard CART. Nearly all statistical methods of regression contain a single *smoothing parameter* $\lambda$ (similar to the cost-complexity parameter in CART, and the `m` and `c` parameters in C4.5), which attempts to address the bias-variance dilemma: how to trade off fit to the training data with some measure of "complexity" of the model. Wahba (1990, Chapter 4) discusses cross-validation, generalized cross-validation, maximum-likelihood, and Bayesian methods of choosing $\lambda$ in the context of smoothing splines. Quinlan (1993) discusses the importance of the C4.5 parameters, and suggests that the user manually perform a search through the space of parameter values using cross-validation to evaluate each parameter. Craven & Shavlik (1993) used this method in their work, but did not report the accuracy improvement over the default parameters nor the specifics of how the search was conducted. John (1994) reports preliminary results on using cross-validation and exhaustive search to set the `m` parameter in C4.5. Feature subset selection can also be viewed as a special case of parameter selection where a bit-vector-valued parameter specifies which features to use. Recent work in machine learning has focused on the use of cross-validation for setting the feature subset parameter vector (John et al. 1994, Kohavi 1995*a*, Kohavi 1994, Caruana & Freitag 1994, Langley & Sage 1994).

Moore, Hill & Johnson (1992) discuss a method for using leave-one-out cross-validation to select many parameters in a nearest-neighbor setting. In the same scenario Moore & Lee (1994) give efficient algorithms for approximating the leave-one-out estimates, evaluating each model only partially until it becomes clear that either it is or is not the best model. Our heuristic of re-running cross-validation when the standard deviation of the estimate of the mean accuracy is above 1% is a rough rule of thumb along the same lines. Variance in $err_{\mathrm{CV}}$ is also discussed in Gasser, Kneip & Kohler (1991), who discuss the use of "plug-in" estimators in kernel regression, giving better bandwidth parameter selection than cross-validation.

In the neural network community, issues of model selection and parameter selection also arise. Since neural nets generally take much longer to train than decision trees, it is not feasible to perform a 10-fold cross validation-directed search through the space of parameters of the kind we report here. Utans & Moody (1991) give a lucid description of the model and parameter selection problem in neural networks. They also present experiments using cross-validation to select the number of hidden units to use in the network and the number of weights to prune after training. Lang, Waibel & Hinton (1990) use a holdout set to decide when to stop training. Weigend (1994) and Finnoff, Hergert & Zimmerman (1993) present methods for indirectly selecting a model parameter (the *effective* network complexity) using a holdout set.

## 7 Conclusion

We have presented a general method for setting the parameters of a learning algorithm: the wrapper method. Using best-first search and cross-validation, we have applied the wrapper method to the C4.5 algorithm yielding a version of C4.5 we call C4.5-AP that selects its own tree-building parameters automatically. We ran experiments on 33 datasets from the UCI and StatLog collections and concluded that at a 95% confidence level, our method improved C4.5 in six domains and had no significant effect in the remainder. At a 90% confidence level, our method improved C4.5 in nine domains, degraded the performance of C4.5 on one very small dataset, and had no effect on the rest. On the large sample of datasets we used to compare C4.5 and C4.5-AP, C4.5-AP's error rate improved C4.5's by a relative factor of 13%.

Our results support our earlier claims that tuning the parameters of a learning algorithm for a specific dataset is better than just using the default parameters for all datasets. We believe that fielded applications of machine learning should use the wrapper method to tune the behavior of the algorithm used.


### Acknowledgments

The work in this paper was done using the $\mathcal{MLC}$++ library, partly funded by ONR grant N00014-94-1-0448 and NSF grants IRI-9116399 and IRI-9411306. George John is supported by a National Science Foundation Graduate Research Fellowship. We would like to thank Pat Langley, J. R. Quinlan, Jerome Friedman, and Robert Tibshirani for comments and pointers to related work. We thank the anonymous referees for many suggestions which helped us clarify our results. Some datasets must be acknowledged: Mangasarian & Wolberg (1990), Zwitter & Soklic (1994*a*), and Zwitter & Soklic (1994*b*).


## References


Brazdil, P., Gama, J. & Henery, B. (1994), Characterizing the applicability of classification algorithms using meta-level learning, *in* F. Bergadano & L. D. Raedt, eds, "Machine Learning: ECML-94. European Conference on Machine Learning", Lectures Notes in Artificial Intelligence, Springer-Verlag, Catania,Italy, pp. 83–102.

Breiman, L., Friedman, J., Olshen, R. & Stone, C. (1984), *Classification and Regression Trees*, Chapman & Hall, New York.

Caruana, R. & Freitag, D. (1994), Greedy attribute selection, *in* Hirsh & Cohen (1994), pp. 28–37.

Casella, G. & Berger, R. L. (1990), *Statistical Inference*, Wadsworth & Brooks/Cole.

Craven, M. W. & Shavlik, J. W. (1993), Learning symbolic rules using artificial neural networks, *in* P. Utgoff, ed., "Proceedings of the Tenth International Conference on Machine Learning", Morgan Kaufmann, pp. 73–80.

Finnoff, W., Hergert, F. & Zimmerman, H. G. (1993), "Improving model selection by nonconvergent methods", *Neural Networks* **6**(6), 771–783.

Gasser, T., Kneip, A. & Kohler, W. (1991), "A flexible and fast method for automatic smoothing", *Journal of the American Statistical Association* **86**(415), 643–652.

Ginsberg, M. L. (1993), *Essentials of Artificial Intelligence*, Morgan Kaufmann.

Hirsh, H. & Cohen, W., eds (1994), *Machine Learning: Proceedings of the Eleventh International Conference*, Morgan Kaufmann.

Holte, R. C. (1993), "Very simple classification rules perform well on most commonly used datasets", *Machine Learning* **11**(1), 63–90.

John, G. H. (1994), Cross-validated C4.5: Using error estimation for automatic parameter selection, Technical Report STAN-CS-TN-94-12, Computer Science Department, Stanford University. Available by anonymous ftp from `starry.Stanford.EDU:pub/gjohn/papers/cvc45.ps`.

John, G., Kohavi, R. & Pfleger, K. (1994), Irrelevant features and the subset selection problem, *in* Hirsh & Cohen (1994), pp. 121–129. Available by anonymous ftp in starry.Stanford.EDU: pub/gjohn/papers/relevance4.ps.

Kohavi, R. (1994), Feature subset selection as search with probabilistic estimates, *in* "AAAI Fall Symposium on Relevance". Available by anonymous ftp in starry.Stanford.EDU: pub/ronnyk/aaaiSymposium94.ps.

Kohavi, R. (1995*a*), The power of decision tables, *in* N. Lavrac & S. Wrobel, eds, "Machine Learning: ECML-95 (Proc. European Conf. on Machine Learning)", Spriner Verlag, pp. 174 – 189. Available by anonymous ftp from starry.Stanford.EDU:pub/ronnyk/tables.ps.

Kohavi, R. (1995*b*), A study of cross-validation and bootstrap for accuracy estimation and model selection, *in* "Proceedings of the 14th International Joint Conference on Artificial Intelligence". Available by anonymous ftp from starry.Stanford.EDU:pub/ronnyk/accEst-long.ps.

Lang, K. J., Waibel, A. H. & Hinton, G. E. (1990), "A time-delay neural network architecture for isolated word recognition", *Neural Networks* **3**, 23–43.

Langley, P. & Sage, S. (1994), Induction of selective bayesian classifiers, *in* "Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence", Morgan Kaufmann, Seattle, WA, pp. 399–406.

Linhart, H. & Zucchini, W. (1986), *Model Selection*, Wiley.

Mangasarian, O. L. & Wolberg, W. H. (1990), "Cancer diagnosis via linear programming", *SIAM News* **23**(5), 1,18.

Michie, D., Spiegelhalter, D. J. & Taylor, C. C. (1994), *Machine Learning, Neural and Statistical Classification*, Prentice Hall.

Mitchell, T. M. (1982), "Generalization as search", *Artificial Intelligence* **18**, 203–266.

Moore, A. & Lee, M. (1994), Efficient algorithms for minimizing cross-validation error, *in* Hirsh & Cohen (1994), pp. 190–198.

Moore, A. W., Hill, D. J. & Johnson, M. P. (1992), An empirical investigation of brute force to choose features, smoothers and function approximators, *in* "Computational Learning Theory and Natural Learning Systems Conference".

Murphy, P. M. & Aha, D. W. (1994), "UCI repository of machine learning databases", Available by anonymous ftp to ics.uci.edu in the pub/machine-learning-databases directory.

Nilsson, N. (1980), *Principles of Artificial Intelligence*, Morgan Kaufmann.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. & Flannery, B. P. (1992), *Numerical Recipes in C: The Art of Scientific Computing*, second edition, Cambridge University Press.

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

Schaffer, C. (1993), "Selecting a classification method by cross-validation", *Machine Learning* **13**, 135–143.

Schaffer, C. (1994), A conservation law for generalization performance, *in* Hirsh & Cohen (1994), pp. 259–267.

Stone, M. (1974), "Cross-validatory choice and assessment of statistical predictions", *Journal of the Royal Statistical Society B* **36**, 111–147.

Thrun, S. B. et al. (1991), The monk's problems – a performance comparison of different learning algorithms, Technical Report CMU-CS-91-197, CMU School of Computer Science.

Utans, J. & Moody, J. (1991), Selecting neural network architectures via the prediction risk: Application to corporate bond rating prediction, *in* "The First International Conference on Artificial Intelligence Applications on Wall Street", IEEE Computer Society Press.

Wahba, G. (1990), *Spline Functions for Observational Data*, CBMS-NSF Regional Conference Series, SIAM, Philadelphia.

Weigend, A. S. (1994), On overfitting and the effective number of hidden units, *in* M. C. Mozer, P. Smolensky, D. S. Touretzky, J. L. Elman & A. S. Weigend, eds, "Proceedings of the 1993 Connectionist Models Summer School", Erlbaum Associates, Hillsdale, NJ, pp. 335–342.

Weiss, S. M. & Kulikowski, C. A. (1991), *Computer Systems that Learn*, Morgan Kaufmann, San Mateo, CA.

Wolpert, D. H. (1994), The relationship between PAC, the statistical physics framework, the Bayesian framework, and the VC framework, Technical report, The Santa Fe Institute, Santa Fe, NM.

Zwitter, M. & Soklic, M. (1994*a*), "Ljubljana breast cancer database", Available by anonymous ftp to ics.uci.edu in the pub/machine-learning-databases directory.

Zwitter, M. & Soklic, M. (1994*b*), "Lymphography database", Available by anonymous ftp to ics.uci.edu in the pub/machine-learning-databases directory.