Wrapped Progressive Sampling Search for Optimizing Learning Algorithm Parameters

Antal van den Bosch

ILK / Computational Linguistics and AI, Tilburg University
P.O. Box 90153, NL-5000 LE Tilburg, The Netherlands

Abstract

We present a heuristic meta-learning search method for finding a set of optimized algorithmic parameters for a range of machine learning algorithms. The method, wrapped progressive sampling, is a combination of classifier wrapping and progressive sampling of training data. A series of experiments on UCI benchmark data sets with nominal features, and five machine learning algorithms to which simple wrapping and wrapped progressive sampling is applied, yields results that show little improvement for the algorithm which offers few parameter variations, but marked improvements for the algorithms offering many possible testable parameter combinations, yielding up to 32.2% error reduction with the WINNOW learning algorithm.

1 Introduction

It is common knowledge that large changes can be observed in the generalisation accuracy of a machine learning algorithm on some task when instead of its default algorithmic parameter settings, one or more parameters are given a non-default value. The fundamental problem in algorithmic parameter selection (or model selection) is that it is hard to estimate which parameter setting would lead to optimal generalization performance on new data. One can estimate it on the labeled data available for training purposes, but optimizing parameters on that easily leads to overfitting. A remedy for overfitting is to use classifier wrapping [7], which partitions the available labeled training material in internal training and test data, and which performs cross-validation experiments to estimate a training-set-internal generalization accuracy. Using this method, competitions can be held among parameter settings, to determine the average best-performing setting to be used later in the experiment on the real test data.

For many tasks and algorithms it is not feasible to test all possible combinations of parameter settings and feature selections exhaustively. By opening up a vast search space, exhaustive wrapping moves the burden of the optimization process to raw computing power. On the other hand, search methods can offer heuristic solutions to finding sufficiently good solutions in a large search space. This paper describes a method for optimizing algorithmic parameters based on wrapped progressive sampling, a heuristic search method we describe in detail in the next section. The method is based on classifier wrapping, but it does not venture into

searching among all possible combinations of parameter settings exhaustively on all training data available. Rather, it borrows a heuristic from progressive sampling [10]. The goal of progressive sampling is to iteratively seek a data set size at which generalization performance on test material converges. The method is to start at a small sample size training set, and progressively increase the training set while monitoring the convergence – and halting the process at some training set size when the error on test material has converged. In this study we do not actively monitor convergence, but we do adopt the progressive sampling method in which we test *decreasing* amounts of settings combinations with *increasing* amounts of training data.

2 Wrapped progressive sampling

The wrapped progressive sampling (henceforth WPS) algorithm takes as input a data set of labeled examples D. An example is represented by a vector of feature values, and labeled with one out of a set of possible output labels. This data set is used as the basis for sampling smaller training and test sets. In the following subsections we detail this process. We start by describing how the sizes of the progressively-sampled training and test sets are computed in Subsection 2.1. We then describe the WPS process in Subsection 2.2.

2.1 Determining progressive sample sizes

The first action of the WPS method is to determine the progressive sizes of the wrapping data set samples that will be needed. Both training sets and test sets will be needed; a single cut is made of the original data (after being suffled randomly) in 80% of the examples for internal training data, and 20% for internal test data. We generate a frequency-clipped pseudo-quadratic series, according to the following three-step procedure:

First, let n be the number of labeled examples in the 80% part of the data set designated to extract internal training material from. A quadratic sequence of d data set sizes is created by using a factor $f = \sqrt[d]{n}$. In all experiments, $d = 20$. Starting with a seed data set of one example, a sequence of $i = \{1 \dots d\}$ data sets with sizes $size_i$ is created by letting $size_1 = 1$ and for every i, $size_i = size_{i-1} * f$. We then limit the generated list of 20 sizes down to a list containing only the data sets with more than 500 examples. We also include the 500-example data set itself as the first set. This leaves a clipped pseudo-quadratic series. For each of the training sets, an accompanying test set is created by taking, from the tail of the 20% compartment of the full data set designated for test material, a set that has 20% of the size of its corresponding training set.

2.2 The wrapped progressive sampling procedure

The WPS procedure is an iterative procedure over the clipped list of data set sizes. The procedure operates on a pool of settings, S, where one setting is a unique

combination of algorithmic parameter values of the chosen machine-learning algorithm A (next to data set D, A is given as input to the WPS procedure). On the outset, S contains all possible combinations of values of algorithm A's parameters. We refer to them as $s_1 \ldots s_c$, c being the total number of possible combinations.

The first step of WPS is to perform experiments with all settings in $s_1 \ldots s_c$. Each of these experiments involves training algorithm A on the first training set (500 examples) and testing the learned model on the first test set (100 examples), and measuring A's test accuracy, viz. the percentage of correctly classified test examples. This produces a list of accuracies, $acc(s_1) \ldots acc(s_c)$. As the second step, badly-performing settings from the current set are removed on grounds of their low score. Any such selection should be performed with some care, since it is unknown whether a currently badly performing setting would perform relatively better as compared to other settings when trained on more examples. WPS, therefore, does not simply sort the list of accuracies and cuts away the lower-performing half or some other predefined fraction. Rather, it attempts to estimate at each step the subset of accuracies that stands out as the best performing group, whichever portion of the total set of accuracies that is. To this end, a simple linear histogram is computed on all accuracies, dividing them in ten equally-sized bins, $b_1 \ldots b_{10}$ (the notation for the size of a bin, the number of accuracies in the bin, is $|b_i|$).

Without assuming any distribution over the bins, WPS enacts the following procedure to determine which settings are to be selected for the next step. This procedure produces a selection of bins, which in turn represents a set of settings represented by the selected bins. First, the bin with the highest accuracies is taken as the first selected bin. Subsequently, every preceding bin is also selected that represents an equal number of settings or more than its subsequent bin, $|b_i| \geq |b_{i+1}|$. This is determined in a loop that halts as soon as $|b_i| < |b_{i+1}|$.

Next, all non-selected settings are deleted from S, and the next step is initiated. This involves discarding the current training set and test set, and replacing them by their next-step progressively sampled versions. On this bigger-sized training and test set combo, all settings in S are tested through experiments, a histogram is computed on the outcomes, etcetera. The process is iterated until either one of these stop conditions is met: (1) After the most recent setting selection, only one setting is left. Even if more training set sizes are available, these are not used, and search halts, returning the one selected setting. Or, (2) after the last setting selection on the basis of experiments with the largest training and test set sizes, several settings are still selected. First, it is checked whether A's default setting is among them (we discuss default settings in the next Section). If it is, this default setting is returned. If not, a random selection is made among the selected settings, and the randomly chosen setting is returned.

3 Customizing wrapped progressive sampling to classifiers

The WPS procedure can operate on any supervised machine learning algorithm (classifier) that learns from labeled examples and is able to label new data after

Algorithm	Parameters varied	Combinations
RIPPER	`-F` (1, **2**, 5, 10, 20, 50); `-a` (`-freq`, **+freq**); `-n` (**-n**, `-!n`), `-S` (**0.5**, 1.0, 2.0); `-O` (0, 1, **2**); `-L` (0.5, **1.0**, 2.0)	648
C4.5	`-m` (1, **2**, 5, 10, 20, 50, 100, 200, 500); `-c` (5, 10, 15, 20, **25**, . . ., 90, 95, 100); `-g` (**not set**, `-g`)	360
MAXENT	`--gis` or –**lb-fgs**, `-g` (**0.0**, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, only with `--lb-fgs`)	11
IB1	`-k` (**1**, 3, 5, 7, 9, 11, 13, 15, 19, 25, 35); `-w` (0, **1**, 2, 3, 4); `-m` (**O**, `M`, `J`); `-d` (**Z**, `IL`, `ID`, `ED1`, only with $k > 1$); `-L` (**1**, 2, only with `-mM` or `-mJ`)	925
WINNOW	`promotion` (1.1, 1.2, **1.3**, 1.5); `demotion` (0.5, 0.7, **0.8**, 0.9); threshold (1.0, 2.0, 3.0, **4.0**, 5.0); `-r` (**2**, 3, 5); `-S` (**0.0**, 0.1, 0.2, 0.3, 0.4)	1200

Table 1: Varied parameters (as they appear on the command line) with their tested values and constraints between parentheses. Default values are printed in bold. The right column lists the total number of combinations of these parameter values, for the five tested algorithms.

learning. The procedure is only effective, naturally, to those algorithms which have algorithmic parameters that change some significant bias aspect of the learning algorithm.

Provided that the chosen algorithm has one or more parameters which can all take two or more values, the WPS method can be applied to all possible settings, each representing one particular combination of possible parameter values. However, this simple procedure may not be applied blindly. Some algorithmic parameters may be mutually exclusive; some parameters may only be selected in combination with another parameter or some other parameter's value. This implies that the method has to be customized for each classifier, using background knowledge about the algorithm's mutual parameter constraints (which are usually known and specified by the algorithm's developers) and about the algorithms' parameter value constraints (which may only be known as rules of thumb).

We customized WPS to the following five well-known machine-learning algorithms. (1) RIPPER [4] is a rule-induction algorithm that compresses a data set of labeled examples into an ordered rule list. We employed implementation *V1 Release 2.5 patch 1.* (2) C4.5 [11] is an algorithm for the top-down induction of decision trees. It compresses a data set of labeled examples into a decision tree that has class labels at its nodes, and tests on feature values on its branches. We employed implementation *Release 8.* (3) MAXENT [6] is a probabilistic learner in which the central probability matrix between feature values and class labels is smoothed towards a state of maximum entropy. We used the implementation by [8], version *20040315.* (4) IB1 [1] is an instance-based classifier based on the k-nearest neighbor (k-NN) classification rule. We used an implementation that supports a range of k-NN kernel plugins, TiMBL [5], version *5.0.0 patch 3.* (5) WINNOW [9] is a linear-threshold classifier which learns weights on the model parameters (features) in a learning phase through an error-based weight update rule, which at the side removes weights that end up below a threshold. We employed SNoW [3], a sparse implementation of Winnow classifiers, version *3.1.3.* Table 1 lists the parameters with their values that were varied, and the total number of combinations of parameter settings tested in the first pseudo-exhaustive round of

Task	Number of examples	features	classes	Class entropy
audiology	228	69	24	3.41
bridges	110	7	8	2.50
soybean	685	35	19	3.84
tic-tac-toe	960	9	2	0.93
votes	437	16	2	0.96
car	1,730	6	4	1.21
connect4	67,559	42	3	1.22
kr-vs-kp	3,197	36	2	1.00
splice	3,192	60	3	1.48
nursery	12,961	8	5	1.72

Table 2: Basic statistics of the six UCI repository data sets. On the top five tasks normal wrapping is performed rather than WPS.

WPS.

4 Experimental setup

For our experiments we used ten benchmark data sets from the UCI repository [2]. Table 2 lists some data set statistics for the ten selected data sets, which all have nominal attributes. Note that "soybean" is short for "soybean-large", and "car" is short for "car evaluation". As the table illustrates, the selection displays a wide variance in numbers of examples, features, classes, and entropy of the classes.

The top five data sets in Table 2 have less than 1000 examples. We apply straightforward wrapping to these five, and apply WPS to the bottom five tasks, to allow for some comparison between the two approaches. The rationale for this split is twofold; (1) it is feasible to run pseudo-exhaustive wrapping 10-fold cross-validation experiments on the smaller datasets, while this is gradually more infeasible with 10 or 100 times as much instances, as in the bottom five tasks; (2) little progressive sampling is possible with under 500 or just over 500 instances, which is the fixed size of the initial training set of WPS.

All data sets were concatenated into one full set (if there were originally disjoint training and test sets provided in the UCI repository), shuffled randomly, and subsequently partitioned into ten 10% partitions. On these partitions 10-fold cross-validation (CV) experiments were performed. All experiments had two variants: in one variant, the default setting of the particular implementation was used for the 10-fold CV experiment. In the other variant, WPS was performed. The difference between the two variants was subject to a one-tailed unpaired t-test.

5 Results

Table 3 displays the effects of normal wrapping and WPS on generalization accuracy on the ten UCI benchmark data sets by RIPPER, C4.5, MAXENT, IB1, and WINNOW, respectively. All tables give an average and the standard deviation for both the default setting and the settings found by normal wrapping on the top five tasks, and WPS on the bottom five tasks, measured in 10-fold CV experiments. For

Task	% Correct test instances default		wrapping / WPS		% Error reduction	One-tailed t-test	
RIPPER							
audiology	75.4	± 8.1	76.3	± 9.0	3.6		$t = 0.233$
bridges	58.2	± 14.8	55.5	± 12.5	-6.5		$t = 0.445$
soybean	91.8	± 2.5	91.5	± 2.5	-3.5		$t = 0.258$
tic-tac-toe	97.5	± 1.3	**99.8**	± 0.3	93.2	$p < 0.001$	$t = 5.330$
votes	95.4	± 2.5	95.2	± 2.8	-4.8		$t = 0.187$
car	87.8	± 3.2	**98.0**	± 0.6	84.0	$p < 0.001$	$t = 10.122$
connect4	75.4	± 0.8	**77.0**	± 1.3	6.6	$p < 0.01$	$t = 3.373$
kr-vs-kp	99.2	± 0.4	98.9	± 0.8	-30.9		$t = 0.884$
splice	93.4	± 1.6	94.3	± 1.0	13.3		$t = 1.447$
nursery	96.6	± 0.7	**98.9**	± 0.5	66.4	$p < 0.001$	$t = 7.732$
C4.5							
audiology	78.0	± 8.0	82.4	± 7.0	19.9		$t = 1.302$
bridges	49.1	± 6.0	52.7	± 7.9	7.1		$t = 1.156$
soybean	91.2	± 2.9	92.4	± 2.5	13.4		$t = 0.892$
tic-tac-toe	84.7	± 4.8	85.9	± 3.1	8.1		$t = 0.689$
votes	95.9	± 2.6	95.4	± 2.5	-11.5		$t = 0.410$
car	91.9	± 2.2	93.2	± 1.9	16.3		$t = 1.450$
connect4	**80.9**	± 0.4	79.4	± 1.7	-7.8	$p < 0.01$	$t = 2.750$
kr-vs-kp	99.5	± 0.3	99.6	± 0.2	20.0		$t = 0.799$
splice	94.0	± 1.6	93.7	± 1.7	-5.5		$t = 0.442$
nursery	97.0	± 0.4	**97.5**	± 0.7	15.3	$p < 0.05$	$t = 1.889$
MAXENT							
audiology	82.8	± 9.3	81.5	± 8.3	-7.6		$t = 0.332$
bridges	56.5	± 16.7	57.3	± 15.2	2.1		$t = 0.126$
soybean	92.6	± 2.3	92.3	± 2.3	-3.8		$t = 0.269$
tic-tac-toe	98.1	± 1.0	98.2	± 0.9	5.4		$t = 0.228$
votes	96.1	± 2.3	96.3	± 3.0	5.6		$t = 0.186$
car	93.5	± 1.7	93.2	± 1.7	-5.4		$t = 0.463$
connect4	75.7	± 1.4	74.7	± 1.7	-4.1		$t = 1.437$
kr-vs-kp	97.8	± 0.9	97.1	± 1.0	-29.3		$t = 1.545$
splice	90.7	± 2.2	**94.5**	± 2.8	41.3	$p < 0.01$	$t = 3.437$
nursery	92.5	± 0.5	92.5	± 0.4	-0.7		$t = 0.296$
IB1							
audiology	79.3	± 9.7	79.8	± 8.7	2.4		$t = 0.119$
bridges	54.6	± 8.1	50.9	± 13.6	-8.0		$t = 0.726$
soybean	91.8	± 3.1	**94.6**	± 2.9	34.0	$p < 0.05$	$t = 2.053$
tic-tac-toe	89.5	± 2.6	**99.2**	± 0.6	92.1	$p < 0.001$	$t = 11.423$
votes	93.8	± 3.7	95.9	± 3.2	33.7		$t = 1.346$
car	94.0	± 0.9	**96.6**	± 1.2	43.3	$p < 0.001$	$t = 5.569$
connect4	71.0	± 0.3	**77.9**	± 2.1	24.7	$p < 0.001$	$t = 10.171$
kr-vs-kp	**97.7**	± 0.5	96.8	± 0.7	-42.5	$p < 0.001$	$t = 3.649$
splice	91.9	± 2.0	**95.4**	± 1.0	43.7	$p < 0.001$	$t = 4.932$
nursery	94.7	± 0.5	**99.3**	± 0.2	86.9	$p < 0.001$	$t = 28.407$
WINNOW							
audiology	71.3	± 14.7	77.0	± 11.8	19.9		$t = 0.959$
bridges	58.2	± 15.9	56.0	± 13.2	-5.3		$t = 0.340$
soybean	84.3	± 5.9	88.2	± 6.1	24.9		$t = 1.457$
tic-tac-toe	90.7	± 3.0	**94.3**	± 2.1	38.2	$p < 0.01$	$t = 3.057$
votes	95.0	± 2.7	95.4	± 2.9	9.2		$t = 0.372$
car	94.3	± 2.1	**96.4**	± 1.7	36.4	$p < 0.05$	$t = 2.396$
connect4	47.2	± 5.8	**69.5**	± 2.5	42.2	$p < 0.001$	$t = 11.087$
kr-vs-kp	97.2	± 0.9	97.0	± 1.0	-5.3		$t = 0.360$
splice	92.1	± 2.0	**94.7**	± 1.6	33.2	$p < 0.01$	$t = 3.272$
nursery	96.4	± 0.7	**98.4**	± 0.4	54.6	$p < 0.001$	$t = 7.653$

Table 3: Parameter optimization effects on learning ten UCI benchmark tasks by the five tested learning algorithms.

convenience, each table displays an error reduction, measured as the percentage of error that was saved by wrapping or WPS – this quantity may be negative if the wrapping method yields worse results. The t value of the one-tailed unpaired t-test is reported, as well as the p level if it is smaller than 0.05. Accuracies in bold mark the significantly better accuracies in a pair of outcomes for one data set.

Two algorithms, IB1 and WINNOW, display marked improvements due to WPS on four or the five UCI tasks. RIPPER is able to gain significant improvements compared to the default settings on three of the five datasets. C4.5 performs significantly better with WPS on two tasks, and MAXENT shows the least effect; only one task is significantly improved due to WPS. Error reduction levels vary widely; they are negative in 17 out of the total of 50 experiments, but only in two cases the negative effect of WPS is significant. On the positive side, of the 33 measured positive effects across the 50 experiments run with all five algorithms, 17 are significant.

To gain more insight into the effects of the two wrapping methods, Table 4 lists the average error reductions for all five algorithms for each of the two wrapping methods tested (i.e. each average is taken over the error reductions measured in five

	Normal wrapping		WPS	
Algorithm	Error reduct.	Reduct./combin.	Error reduct.	Reduct./combin
RIPPER	16.4	0.025	27.9	0.043
C4.5	7.4	0.021	7.7	0.021
MAXENT	5.9	0.536	0.4	0.036
IB1	30.8	0.033	31.2	0.034
WINNOW	17.4	0.015	32.2	0.027

Table 4: Average error reduction levels yielded through normal wrapping and WPS and "gain per tested combination" statistic, for each of the five learning algorithms.

experimental outcomes). Apart from these averages, which range between 0.4% for MAXENT with WPS to 32.2% for WINNOW with WPS, the table also displays the relative contribution of each tested combination of parameter settings. For example, the 32.2% error reduction of WINNOW is due to a search by WPS among 1200 combinations; the average contribution from each combination is $32.2/1200 = 0.027\%$. Interestingly, this average contribution per setting is a fairly constant number ranging between about 0.02 and 0.04. The single outlier is the high average relative contribution of each of the 11 combinations of settings of MAXENT, which can be considered accidental and unreliable as an average.

6 Discussion

In this paper we demonstrated the use of wrapped progressive sampling for algorithmic parameter optimization. We customized WPS to five machine learning algorithms, and ran experiments on UCI benchmark data sets comparing the algorithms' default settings as provided by the employed implementations to the settings found by WPS for each fold of a 10-fold CV experiment. Furthermore, we ran the same five algorithms and applied normal (pseudo-exhaustive) wrapping to five smaller UCI tasks, which would not be feasible for the larger datasets on which WPS is applied.

The WPS procedure appears to have the desired function. It tends to reduce average error over the investigated UCI data sets at rates ranging from a negligible 0.4% with MAXENT to a considerable 27.2% (RIPPER), 31.2% (IB1), and 32.2% (WINNOW). In 13 out of the 25 WPS experiments, WPS performs significantly better than the default setting. As a computationally feasible alternative to normal wrapping, which is tested on the smaller five datasets, WPS offers at least as promising results. There even appears to be a constant gain in error reduction per tested combination of parameter settings, leading to higher gains when larger parameter spaces can be explored – in our experiments, about 0.02–0.04% per tested combination.

The fact that WPS sometimes produces a lower generalization accuracy than obtained with the default setting of an algorithm can partly be explained by the general fact that wrapped estimations on training data may not carry over to test data – simple wrapping also produces negative results. On the other hand, the WPS procedure is susceptible to discarding settings that would perform well on

test data, but perform badly on small amounts of training data and are therefore deleted early on in the process. More comparative experiments and deeper analyses are necessary to investigate this potential cause of negative end results.

To conclude, wrapped progressive sampling is a meta-learning approach to algorithmic parameter optimization that does not produce much effect, unsuprisingly, with algorithms that offer little variation in their parameters, such as MAXENT. It is, however, able to make more improvements when more possible combinations are available to search through.

Acknowledgements

This research is funded by the Netherlands Organisation for Scientific Research (NWO). The author wishes to thank William Cohen, Robert Stockton, Iris Hendrickx, Walter Daelemans, Zhang Le, and Dan Roth for comments, suggestions, and discussions.

References

[1] D. W. Aha, D. Kibler, and M. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

[2] C. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. `http://www.ics.uci.edu/ mlearn/MLRepository.html`.

[3] A. J. Carlson, C. M. Cumby, J. L. Rosen, and D. Roth. SNoW user guide. Technical Report UIUCDCS-R-99-2101, Cognitive Computation Group, Computer Science Department, University of Illinois, Urbana, Illinois, 1999.

[4] W. Cohen. Fast effective rule induction. In *Proceedings 12th International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[5] W. Daelemans, J. Zavrel, K. van der Sloot, and A. van den Bosch. TiMBL: Tilburg memory based learner, version 5.0, reference guide. ILK Technical Report 03-10, Tilburg University, 2003.

[6] S. Guiasu and A. Shenitzer. The principle of maximum entropy. *The Mathematical Intelligencer*, 7(1), 1985.

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

[8] Zhang Le. *Maximum Entropy Modeling Toolkit for Python and C++*. Natural Language Processing Lab, Northeastern University, China, 2004. `http://www.nlplab.cn/zhangle/software/maxent/manual/`.

[9] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[10] F. Provost, D. Jensen, and T. Oates. Efficient progressive sampling. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, pages 23–32, 1999.

[11] J.R. Quinlan. C4.5*: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.