Jointly optimizing model complexity and data-processing parameters with mixed-input SPSA

Jim Garrett *

June 30, 2004

Abstract

When predictor selection is applied prior to modeling, and modeling performance is assessed by cross-validation (or most other methods), then that performance estimate will be biased. When the number of predictors outstrips the number of data cases, the bias can be severe, a phenomenon known as selection bias. Fundamentally, a model process is applied—of which fitting the model is only the last step—yet performance estimation does not encompass the entire process. Cross-validation that examines the entire process is free of selection bias, yet such cross-validation presents a challenging optimization problem. An efficient multiparameter optimization algorithm, Simultaneous Perturbation Stochastic Approximation ("SPSA"), can be adapted to handle loss functions having both continuous and ordered discrete inputs. SPSA is relatively efficient, handles noisy loss functions, and is unlikely to become trapped in inferior local optima, particularly for noisy loss functions. This "mixed-input" SPSA can jointly optimize data-processing parameters (including feature-selection) and model-complexity parameters, and also provide a type of cross-validation performance estimate that is free of selection bias.

1 The problem

1.1 Why joint optimization?

Predictive modeling generally proceeds through a chain of data-processing steps, each locally optimized in some sense. Raw data may arise from an instrument, such as a mass spectrometer or gene-array scanner, which applies algorithms thought to be nearly optimal. Raw data is further processed: peaks on spectra are identified, baselines are subtracted, etc. Growth curves may be smoothed and derivatives measured. Data ready for statistical analysis in the traditional sense may then be scaled, averaged, transformed, etc. Dimension reduction

*jim_garrett@bd.com

and/or variable selection may occur, and finally a model may be fitted, whose performance is the final object of the entire chain.

Steps in the chain typically are optimized without consideration of the final output of the chain, yet such local optimization can be hard and may not optimize the total chain even when done well. Consider for instance building a model to discriminate growth of microorganisms in suspension from non-growth. The maximum first derivative of a growth curve can be a reasonable biological growth metric, as mentioned in [13]. Estimating derivatives of smooth curves is a challenging problem that has been explored in [2] and [10], but always with a view to optimizing some loss function related to squared-error loss. While this is a natural and reasonable criterion to consider, and parameters optimal for such criteria are unlikely to give very poor values as classification inputs, they may not be optimal for the larger classification problem because the bias-variance tradeoff optimal for squared-error loss may not be optimal for subsequent classifying. In particular, high bias (induced by oversmoothing) may matter little in subsequent classification, provided all cases are biased similarly; meanwhile the gain in reduced variance could be valuable. Why not then choose as a smoothing-parameter criterion the downstream classification performance (as estimated by cross-validation)? The criterion cannot be evaluated independently of classification parameters, so the optimization problem effectively becomes one of jointly opimizing data-processing and classification.

Optimizing some portion of a data-processing chain (in addition to the final model-fitting portion) also has implications for statistical inference. Ambroise and McLachlan [1] describe how screening variables prior to fitting classification models can severely bias cross-validation performance estimates. Variable screening is properly seen as part of a modeling process, and since (in their examples) cross-validation does not encompass this important step, it yields biased performance estimates. The remedy: perform variable screening at each iteration of cross-validation, so that cross-validation assesses the entire modeling process, not just the "model-fitting" at the end; Ambroise and McLachlan refer to this as "external" cross-validation.

Other data-processing steps may be more benign than variable selection, but any such step can induce some bias if performance estimates do not account for it. Hence joint optimization of a data-processing chain *and* model complexity can remove bias and perhaps give better performance.

Most models have a small number of parameters governing complexity, so ad-hoc search methods may be adequate. However, adding data-processing parameters to the problem will quickly make ad-hoc methods insufficient. Design of Experiment (DOE) methods [9], typically carried out in a sequence of screening and then response-surface experiments, optimize noisy loss functions of many variables very effectively if approximately, while also providing surface estimates in the neighborhood of the optimum. In fact, DOE methods are probably underutilized for model optimization. However, DOE methods are interactive, and often a more unsupervised "batch-mode" process is desired.

1.2 SPSA

The Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm ([15, 17]) can optimize differentiable noisy loss functions of many parameters effectively. SPSA is relatively robust to local optima; Maryak and Chin [8] show that under certain conditions SPSA is guaranteed to converge to the global optimum; while these conditions are unlikely to be known for a given loss function, Maryak and Chin also provide anecdotal evidence that SPSA can be more robust even than simulated annealing. The web site http://www.jhuapl.edu/spsa provides a thorough bibliography. Johannsen et. al. [7] apply SPSA to optimize jointly variable selection and distance metric in a nearest-neighbor classification problem by transforming the predictor-selection problem into one using continuous parameters.

To optimize a p-length parameter vector θ, SPSA uses a perturbation vector Δ_k where each $[\Delta_k]_i, i = 1, \ldots, p$ is selected randomly from {-1, 1}. $[\Delta_k]_i$ refers to element i of vector Δ_k. Given a gradient-estimation step-size sequence c_k, the loss function is evaluated at $\hat{\theta}_k + c_k\Delta_k$ and $\hat{\theta}_k - c_k\Delta_k$, generating y_k^+ and y_k^-, respectively. The one-dimensional gradient estimate $\hat{g}_k(\hat{\theta}_k) = (y_k^+ - y_k^-)/(2c_k)$ is calculated, and a "steepest descent" step is taken: $\hat{\theta}_{k+1} = \hat{\theta}_k - a_k\hat{g}_k(\hat{\theta}_k)$ for step-size sequence a_k. Like all first-order optimization algorithms, step sizes $\{a_k\}$ and the relative scale of the parameters θ must be selected carefully. Further refinements, such as enforcing box constraints and limiting the update distance, can improve the odds of successful optimization in certain cases.

Gain sequences a_k and c_k are divergent sequences that converge to zero. Typically

$$a_k = \frac{a}{(k + A)^\alpha}$$

and

$$c_k = \frac{c}{k^\gamma}$$

with $\alpha = 0.602$ and $\gamma = 0.101$. Optimal choices for a and c are problem-dependent [14].

The set $\{\hat{\theta}_k \pm \Delta_k\}$ may remind statisticians of a fractional factorial design, although statisticians don't generally recommend multifactor experimental designs with two design points. In fact, $\{\hat{\theta}_k \pm \Delta_k\}$ can be viewed as a randomly-selected fraction of a $2^{p-(p-1)}$ fractional factorial design, where p is the number of parameters to optimize. As an experimental design it hardly seems informative, yet a different random fraction is selected at every iteration, in the aggregate providing enough information to converge. At any rate, it seems fitting to refer to the gradient-estimation points $\{\hat{\theta}_k \pm \Delta_k\}$ as the "design points" for the kth iteration.

Since only two points are evaluated regardless of the dimension of θ, including a parameter whose role is uncertain poses little cost to convergence properties, and may offer a benefit. If a parameter has no effect on the loss function, including it in an optimization will not change the sequence at all, and will not require more loss evaluations. (The parameters value will be set arbitrarily

by the process, which is inconsequential if the parameter has no effect.) If the parameter in question *does* have a strong effect, the optimization *will* converge more slowly, but ultimately specifying a parameter that merits specification, and possibly yielding better ultimate performance than if the parameter were specified arbitrarily. If parameters with weak effects are mixed with strong-effect parameters, the process will quickly converge to a near-optimum neighborhood (optimizing the strong-effect parameters), and then move slowly, optimizing the weak-effect parameters. In this situation following the process until apparent convergence of all parameters may not be worthwhile.

2 Mixed-input SPSA

When optimizing both data-processing and model-complexity parameters, very likely some parameters will be continuous and others discrete. As presented thus far, SPSA applies only to continuous parameters. Hill et. al. [6] discuss three approaches to adapting SPSA to purely discrete problems. Technically these extend SPSA to integer domains, but since a countable ordered set can be put in one-to-one correspondence with integers, these results can be applied to countable ordered sets. One of these approaches seems particularly convenient to the mixed problem. In this approach, Hill et. al. extend a loss function L defined on a discrete subset of $\mathbb{R}^p$ to L^* on $\mathbb{R}^p$ by elementwise linear interpolation, and base the gradient of L^* on L evaluated at adjacent discrete points, again elementwise.

Two modifications to Hill's L^* approach seem to be in order. First, "linear interpolation" suggests that in order to obtain the two L^* evaluations SPSA requires, L must be evaluated at least twice, perhaps four times if design points are widely separated. For example, consider optimizing a single parameter ν for which L is defined on the integers; for instance let $L(1) = 1, L(2) = 0.5, L(3) = 2$, and $L(4) = 2.5$. For design points $\nu^+ = 3.5$ and $\nu^- = 1.5$, we have $L^*(\nu^-) = (1.0 + 0.5)/2 = 0.75$ an $L^*(\nu^+) = (2.0 + 2.5)/2 = 2.25$. Four loss evaluations are required to obtain the gradient estimate. If instead of L^* we extend with $L^\sim(\nu) = L(\lfloor \nu \rfloor)$, we find $L^\sim(\nu^-) = 1.0$ and $L^\sim(\nu^+) = 2$, yielding a gradient estimate of $(2 - 1)/2 = 0.5$, similar in magnitude and in the same direction as L^*'s gradient but requiring only two evaluations of L.

Suppose $2 \leq \nu^- < \nu^+ \leq 3$. Since L^* extends L linearly between integers, L^*'s gradient estimate must be $(L(3) - L(2))/1 = 1$ for these design points, hence the design points may as well be 2 and 3. However, $L^\sim$'s gradient is $(L(2) - L(2))/0$ which is undefined. In this case let us add 1 to one of the (truncated) design points, making it 3. Which one? Since the identification of one point as ν^+ and the other as ν^- was originally based on a random perturbation vector, it makes no difference: select one randomly. Equivalently, generate another perturbation vector η taking on values 0 and 1 with equal probability. Add η to $\lfloor \nu^+ \rfloor$, and add its "opposite" $(\eta + 1) \mod 2$ to $\lfloor \nu^- \rfloor$. Obtain a gradient estimate with the newly-perturbed truncated design points.

Continuing the example, suppose $\eta = [0]$. The gradient estimate is

$$\begin{aligned}\frac{L(\lfloor \nu^+ \rfloor + 0) - L(\lfloor \nu^- \rfloor + 1)}{(\lfloor \nu^+ \rfloor + 0) - (\lfloor \nu^- \rfloor + 1)} &= \frac{L(2) - L(3)}{2 - 3} \\ &= (0.5 - 2)/-1 \\ &= 1.5\end{aligned}$$

the same as L^*'s gradient estimate for the design points ν^- and ν^+.

Generalize this algorithm to multiple parameters as follows, yielding the first modification to Hill's approach:

1. Leave continuous parameters as they are.
2. Take the integer part of nominally integer-valued parameters.
3. For those integer-valued parameters having the same integer part across design points, perturb by 0 or 1. Use the resulting design points to estimate the gradient.

The second modification to Hill's approach is motivated by the need to scale and transform parameters. In practice, ensuring that gradient elements are of similar scale can dramatically improve the efficiency of SPSA, or any other first-order algorithm. Moreover, transforming some elements nonlinearly, so that a given step-size on the transformed scale has different implications in different regions, can also improve performance. Scaling different discrete parameters differently causes their admissible values to be equally-spaced on grids of different mesh-size, while transforming nonlinearly yields a grid of admissible values that is not even of fixed mesh size. Both complicate gradient estimation.

Above, we added a perturbation when discrete elements of design points were within 1. After differentially scaling and/or transforming, on what scale is this distance evaluated? On the original, "natural" scale; call the space of "natural" parameter values Ω. Optimization mathematics (design point generation, gradient estimation, and updates) are applied to the space of transformed parameters; call this Θ. Let $f : \Omega \to \Theta$ and let f be invertible. Let design points θ^+ and θ^- be generated in Θ. The corresponding design points in Ω are $\nu^+ = f^{-1}(\theta^+)$ and $\nu^- = f^{-1}(\theta^-)$. Apply the algorithm above to modify ν^+ and ν^-; call the modified points ν_*^+ and ν_*^-. To estimate the gradient on the transformed scale, calculate corresponding modified design points in Θ, $\theta_*^+ = f(\nu_*^+)$ and $\theta_*^- = f(\nu_*^-)$. Finally, let

$$\hat{g}_k(\hat{\theta}_k) = (\theta_*^+ - \theta_*^-)\frac{L(\theta_*^+) - L(\theta_*^-)}{(\theta_*^+ - \theta_*^-)}$$

where vector multiplication and division in this expression is elementwise.

3 Implementation details

When applying mixed-input SPSA to optimize cross-validation performance estimates, a few implementation details are worth considering.

3.1 Maximum change in $\hat{\theta}$

Allowing a given maximum change in $\hat{\theta}$ limits the effect of outlying loss evaluations. This is implemented by calculating $d = \sqrt{||a_k \hat{g}_k||}$. If d exceeds the given threshold, the change vector is normalized to length 1, i.e., $\hat{\theta}$ is updated by $-a_k \hat{g}_k / d$.

3.2 Complexity penalty

When selecting a model complexity parameter to maximize cross-validation a performance estimate, it is common to select the simplest model that does not appreciably degrade performance relative to the apparent optimum. This is not easy to do when optimizing two or more parameters, however. Fortunately, much the same effect can be had by including in the optimization criterion a complexity penalty on parameters that regulate model complexity. The appropriate size of the penalty can be difficult to determine a priori, but in many situations, including that of classification, penalties can be motivated naturally by the degree of performance degradation that one would be willing to trade for a unit decrease in a complexity parameter. For instance, if one parameter is the number of predictors to use, then if one is willing to lose a percentage point of classification accuracy in order to drop one predictor, then the criterion

$$L(\theta) = \text{performance accuracy} + 0.01 * (\text{number of predictors})$$

would be justifiable.

3.3 Incomplete cross-validation

SPSA's ability to optimize noisy loss functions makes it ideal for optimizing cross-validation performance estimates. In fact complete cross-validation is generally not necessary, and in examples presented here 10% of cases is set aside at each iteration. The random selection is stratified so that the frequencies of positives and negatives in the set-aside data set match the class prevalences in the original data as closely as possible.

The same random partition of cases is used for all loss-function evaluations in that iteration. Continuing the analogy to DOE methods, this amounts to blocking on partitions.

3.4 Performance estimation

SPSA-optimized cross-validation performance as described above yields a series of nearly-independent small-sample performance estimates, evaluated in points in optimization space that are near in sequence. We desire a performance estimate (as precise as possible) for the last point in optimization space visited. This problem is amenable to Generalized Additive Modeling [5]. The examples presented here use the mgcv package for R ([19, 20, 11]) because its automated smoothing-parameter selection is convenient.

Classification performance estimates are often based on a two-way table of actual vs. predicted classes for cases not included in model-fitting, i.e., a confusion matrix [12]. For two-class classification, let the four cells in the 2×2 confusion matrix be denoted d_1, d_2, d_3, and d_4 (order is irrelevant, and generalization to larger matrices is straightforward). At each SPSA iteration, evaluate $L(\hat{\theta}_k)$ (in addition to the two gradient-estimation points) and in the process obtain a small-sample confusion matrix for that iteration. Arrange counts with which to estimate $P(d_1)$, $P(d_2|d_1)$, and $P(d_3|d_1, d_2)$ (only three need be estimated; the fourth follows from the unit-sum constraint). After fitting a logistic-regression GAM to each constructed data set, estimate probabilities $P(d_1), \ldots, P(d_3|d_1, d_2)$ for the final iteration and then multiply to obtain the confusion-matrix probabilities. These probability estimates support any other calculation (agreement, sensitivity, specificity, etc.).

4 Limitations

Although this article extends SPSA to deal with certain types of loss functions in discrete and continuous parameters, some cautions are in order.

First, if the algorithm can handle discrete parameters, shouldn't it solve the feature-selection problem directly? Simply let an indicator vector define which variables to include, and feed this vector in as an additional set of parameters to optimize. This will work in principle. However, if there are very many features, this will be a very high-dimensional problem that may take very many iterations to converge. Also, at every iteration mixed-input SPSA fits two models, each using approximately half the variables. For models that do not allow more predictors than cases, this will be simply impossible. For models that can accommodate more predictors than cases, contrasting models based on roughly-equal partitions of the predictors will be informative early in optimization, and SPSA will move towards the optimum subset reasonably efficiently. Then, if the optimum subset is much smaller than half the total set, continuing to include apparently worthless predictors at every iteration will add needless computational burden (although any robust algorithm will probably need to visit such predictors occasionally).

Depending on the number of predictor variables and the type of model, a better strategy might be to implement a variable-screening algorithm that is governed by one or a few parameters, and optimize those parameters. For instance, one might apply a location test to each variable and select those with p-values falling below a threshold; the parameter to optimize would be threshold.

SPSA easily accommodates independent ("box") constraints by resetting offending elements of $\hat{\theta}_{k+1}$ to their respective limits. To handle gradient-estimation steps that extend beyond limits, the loss function can be programmed to replace offending parameters with the nearest admissible value. Dealing with dependent constraints is harder: Wang and Spall [18] discuss enforcing general inequality constraints with penalty functions, an approach that requires a penalty sequence, hence further algorithm parameters to tune.

Consequently, mixed-input SPSA can be applied to integer-valued parameters (and by extension, ordered discrete parameters) but not to unordered discrete parameters with more than two levels. Identifying one level among three or more levels is exclusive—the other two or more levels cannot be picked—implying complex constraints.

5 Examples

Consider for example a simple data-processing chain, implemented in R [11] using the R package gbm:

1. Rank predictors by p-value using Wilcoxon's signed-rank test.
2. Select the k most significant predictors.
3. Fit a classifying gradient-boosting machine (GBM) [3, 4].

GBM's use regression trees, which select predictors in the course of fitting, so predictor preselection is not required. However, they have three fitting parameters requiring specification, and so pose an interesting optimization example here. A GBM is based on a sequence of trees; the sequence takes the GBM from a model of constant predicted value (0 trees) through models that deviate from the constant value in progressively more complex ways. A parameter referred to as "shrinkage" dampens the contribution from each tree. The number of splits allowed in each tree determines the maximum degree of interaction the GBM can exhibit. k, the number of trees, and the degree of interaction are discrete, while shrinkage is continuous.

Like all models, GBM's can exhibit selection bias. When applied to a simulated data set with 50 cases (25 "positive" and 25 "negative") and 1000 predictors, all independent standard random normals, this author was able to obtain apparent 84% classification accuracy in 10-fold cross-validation (after selecting predictors) by taking $k = 10$, shrinkage $= 0.05$, number of trees $= 1000$, and allowing no interaction effects. This was completely due to predictor preselection: using all predictors, the highest apparent accuracy for any set of fitting parameters was 44%, consistent with the expected 50%.

Applying mixed-input SPSA to optimize the data-processing chain yields a performance estimate, and parameter settings, consistent with the conclusion that no predictor is useful. k, shrinkage, and the number of trees were optimized on scales transformed to $\sqrt{k}$, $\log_{10}(\text{shrinkage})$, and $\sqrt{\text{number of trees}}$, respectively. The optimization process was initiated at $k = 5$, shrinkage $= 0.1$, number of trees $= 20$, and interaction level $= 1$, and parameters were limited to $1 \leq k \leq 100$, $0.01 \leq$ shrinkage ≤ 1, $1 \leq$ number of trees ≤ 1000, and $1 \leq$ interaction level ≤ 2 (1 indicates main effects, 2 indicates two-way interaction). The loss function was

$$L(\theta) = \text{classification error rate} + 0.005(\text{number of trees}) + 0.01k.$$

The optimization process was run for 100 iterations with tuning parameters $a = 50$, $A = 100$, and $c = 4$. A maximum change of 3 was enforced.

The top plot in figure 1 indicates that the loss function is indeed very noisy. Loss generally trended downwards, although (probably by chance) the final value was quite high. Much of the decrease occurs before the 40th iteration. Change in loss was small relative to the noise, possibly because the initial parameter values were not far from optimal, but Figure 2 indicates that parameters changed substantially nevertheless. In particular, k quickly reached 1.0, the smallest allowed value, and stayed there except for occasional excursions. The number of trees also reached 1.0, its smallest allowable value. Among the last 20 iterations, one predictor was selected 13 times, another 3 times, and four others once each. Therefore the optimal model either has no predictors (hence zero trees, but zero lies outside the permitted optimization space) or one predictor. Using the methods of 3.4, cross-validation prediction accuracy is estimated to be 45.4%, so we may conclude that the ideal model has no predictors.

Evidently mixed-input SPSA applied to the joint optimization of predictor selection and model-fitting parameters can give reasonable results free of selection bias. But can it successfully identify useful predictors, and yield reasonable classification estimates? Suppose in the pure noise data set above we substitute 3 informative predictors for the first 3 noise predictors as follows: Let x_1 and x_2 be jointly multivariate normal with mean 20, standard deviation 1, and covariance (and correlation) 0.75. Indepndently of x_1 and x_2, Let x_3 be gamma with shape 1 and scale 10 in the parameterization used by R (yielding mean 10 and variance 100). Create a class indicator variable with the ith case distributed Bernoulli with probability p_i, where $\text{logit}(p) = x_1 + x_2 + \log(x_3)$. Data drawn from this model will likely not have clear class separation in the informative predictors, but classification significantly better than noise levels will be possible. In one simulation realization examined here, a logistic regression model fitted to x_1, x_2, and $\log(x_3)$ (using all cases) yielded classification accuracy of 76%, where 0.5 was applied as a classification threshold for predicted class probabilities. Stepwise model selection using AIC yielded a model with x_1 and $\log(x_3)$.

Using the same parameter transformations and boundaries as above but beginning the optimization with $k = 100$, shrinkage = 0.1, 100 trees, and interaction level = 1, the process with this data yielded clear decrease in loss (see figure 1) and reasonable though conservative parameter values. (k was initialized to a high value to verify that the process could in fact move through the optimization space.) Figure 3 presents parameter histories during optimization. In 100 iterations $\hat{\theta}$ reached $k = 1$, shrinkage = 1.0, number of trees = 2.96 (implying 2—recall that the data-processing chain is specified by the integer part), and interaction level = 1.82 (implying 1). Among the last 20 iterations, x_1 was selected 12 times and x_2 was selected 9 times (on one of these iterations both were selected). The small number number of trees, the high shrinkage value, and the lack of interaction indicate the apparent optimal model makes two splits in one predictor, a relatively simple model. Prediction accuracy was estimated

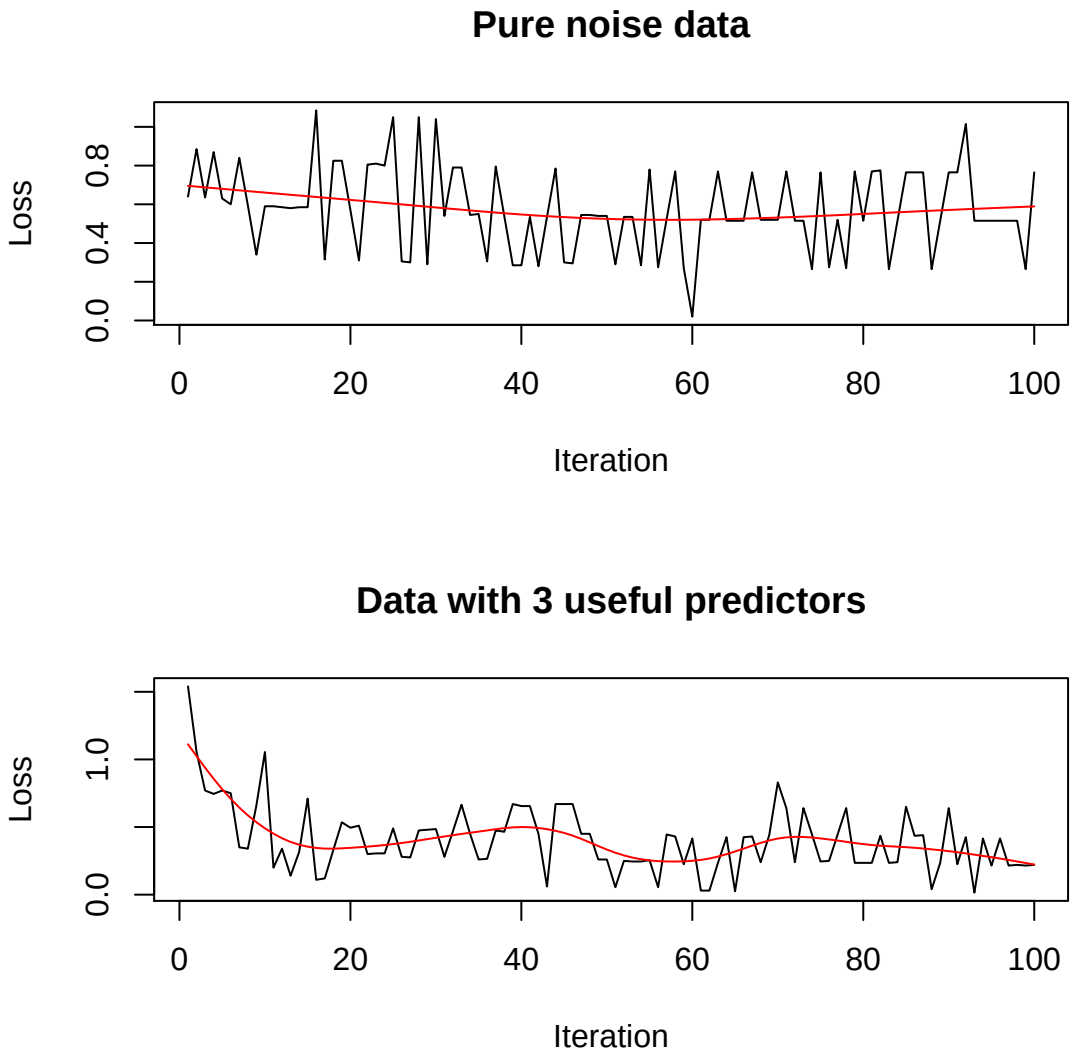

Figure 1: Observed loss values during data-processing chain optimization, with overlaid smoothing spline. Top, applied to noise data; bottom, applied to data with 3 useful predictors.

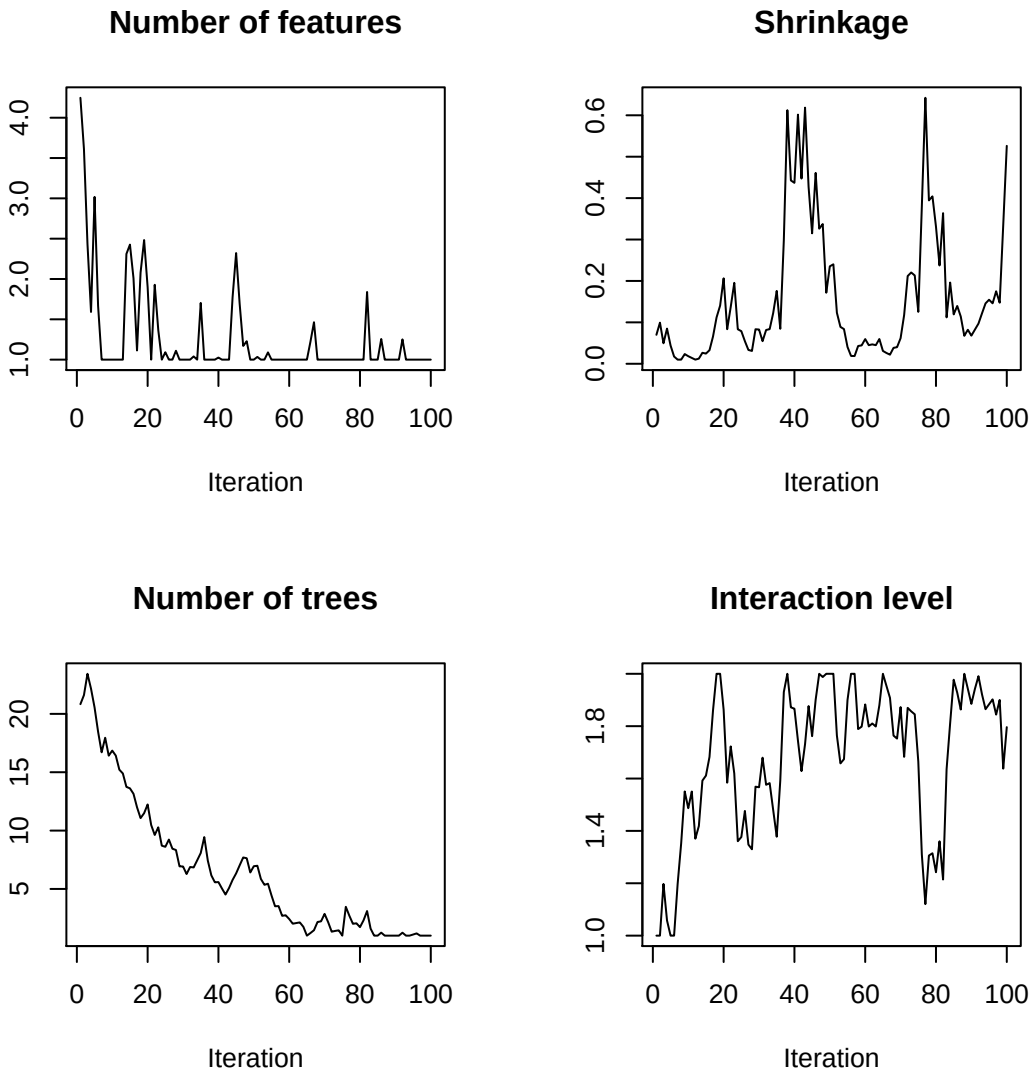

Figure 2: History of parameter values during the course of optimization for the data set in which no predictor is useful for classification.

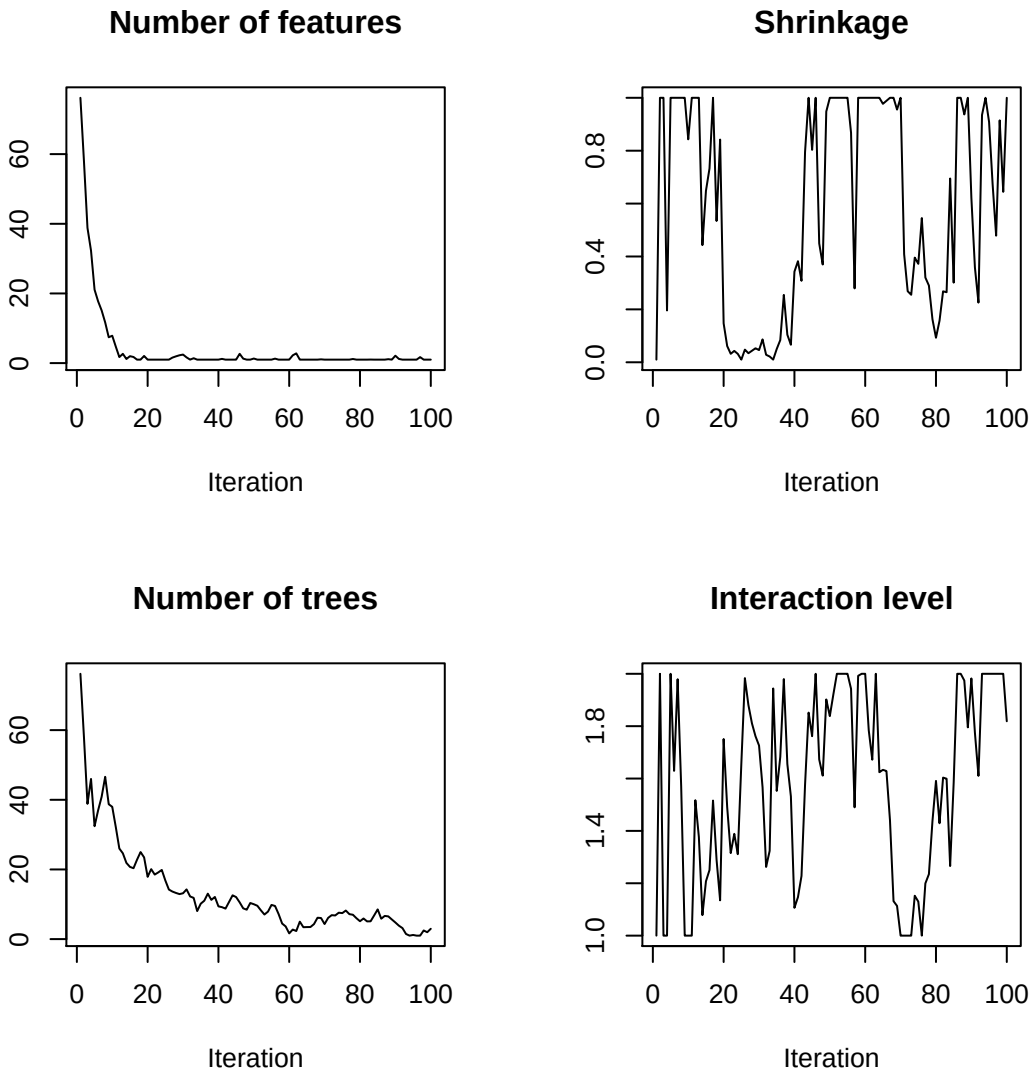

Figure 3: History of parameter values during the course of optimization for the data set having 3 useful predictors.

to be 76.8%.

The apparently optimum parameters indicate 1 predictor is best, when in fact we know that 3 are informative; however, the loss function penalizes each selected predictor by 1 percentage point, and using two or three predictors may not improve classification performance by more than the required one or two percentage points.

In both examples, running the optimization process for more iterations could allow parameter estimates settle more. Alternatively, it would be prudent to regard the estimates as approximate, and perform some checks by hand in the neighborhood of the estimates. Like any first-order optimization method, SPSA typically moves quickly to the neighborhood of the optimum (while gradients are high), but once close to the optimum (where gradients are small) it converges slowly. Thus waiting for parameters to settle may be an inefficient use of computing time.

Spall [16] introduced a second-order analog to SPSA which speeds convergence once the process is in the neighborhood of the optimum; this approach might be adaptable to mixed inputs.

6 Conclusions

Mixed-input SPSA offers a way to jointly optimize parameters governing data processing (notably feature selection) and model complexity, while offering cross-validation performance estimates free of selection bias. In a "negative" example involving pure noise, the algorithm performed correctly, confirming it is free of selection bias. In a "positive" example involving 3 useful predictors, the algorithm returned a conservative but reasonable estimate of the number of predictors to use and yielded returned a reasonable performance estimate.

SPSA was applied for 100 iterations in each case; this number constitutes 300 loss evaluations (200 to optimize, 100 to assess progress and estimate performance). This would be sufficient to perform 30 complete 10-fold cross-validation runs, enough for almost four 2^{4-1} fractional factorial experiments. If such experiments were conducted with incomplete cross-validation runs (somehow blocking on the subset of fold labels set aside), still more designed experiments could be conducted. An analyst adept at DOE methods, designing and analyzing a series of experiments interactively, could probably reach a sound optimum, and have high confidence in the quality of that result, perhaps with less computer work than the SPSA algorithm implemented here. Indeed, DOE methods should not be neglected. However, SPSA can reach a good answer without supervision.

References

[1] Christophe Ambroise and Geoffrey J. McLachlan. Selection bias in gene extraction on the basis of microarray gene-expression data. *Proceedings*

of the National Academy of Sciences of the United States of America, 99(10):6562–6566, 2002.

[2] Jianqing Fan and Irène Gijbels. Data-driven bandwidth selection in local polynomial fitting: Variable bandwidth and spatial adaptation. *Journal of the Royal Statistical Society B*, 57(2):371–394, 1995.

[3] J. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001.

[4] J. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38(4):367–378, 2002.

[5] Trevor J. Hastie and Rob J. Tibshirani. *Generalized Additive Models*. CRC Press, 1990.

[6] Stacy D. Hill, László Gerencsér, and Zsuzsanna Vágó. Stochastic approximation on discrete sets using simultaneous perturbation difference approximations. In *Conference on Information Science and Systems*, 2003.

[7] David A. Johannsen, Edward J. Wegman, Jeffrey L. Solka, and Carey E. Priebe. Simultaneous selection of feature and metric for optimal nearest neighbor classification. *Communications in Statistics — Theory and Methods (in press)*, 2004.

[8] John L. Maryak and Daniel C. Chin. Global random optimization by simultaneous perturbation stochastic approximation. In *Proceedings of the American Control Conference*, pages 756–762, 2001.

[9] Douglas C. Montgomery. *Design and Analysis of Experiments, 5th Edition*. Wiley, 2000.

[10] H.-G. Müller, U. Stadtmüller, and T. Schmitt. Bandwidth choice and confidence intervals for derivatives of noisy data. *Biometrika*, 74:743–749, 1987.

[11] R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2003. ISBN 3-900051-00-3.

[12] Brian D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

[13] B. W. Silverman. Some aspects of the spline smoothing approach to nonparametric regression curve fitting. *Journal of the Royal Statistical Society B*, 47(1):1–52, 1985.

[14] James C. Spall. Implementation of the simultaneous perturbation algorithm for stochastic opimization. *IEEE Transactions on Aerospace and Electronic Systems*, 34(3):817–823, 1998.

[15] James C. Spall. An overview of the simultaneous perturbation method for efficient optimization. *Johns Hopkins APL Technical Digest*, 19:482–492, 1998.

[16] James C. Spall. Adaptive stochastic approximation by the simultaneous perturbation method. *IEEE Transactions on Automatic Control*, 45(10):1839–1853, 2000.

[17] James C. Spall. *Introduction to Stochastic Search and Optimization—Estimation, Simulation, and Control*. Wiley–Interscience, 2003.

[18] I-Jeng Wang and James C. Spall. Stochastic optimization with inequality constraints using simultaneous perturbations and penalty functions. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, pages 3808–3813, 2003.

[19] S. N. Wood. Modelling and smoothing parameter estimation with multiple quadratic penalties. *Journal of the Royal Statistical Society B*, 62(2):413–428, 2000.

[20] Simon N. Wood and Nicole H. Augustine. Gams with integrated model selection using penalized regression splines and applications to environmental modelling. *Ecological Modelling*, 157:157–177, 2002.