Optimization Methods and Software
Vol. 19, No. 5, October 2004, pp. 577–594

Taylor & Francis
Taylor & Francis Group

MODEL BUILDING WITH LIKELIHOOD BASIS PURSUIT

MICHAEL C. FERRIS[a,*], META M. VOELKER[a,†] and HAO HELEN ZHANG[b,‡]

[a]*Department of Computer Sciences, University of Wisconsin, 1210 West Dayton Street, Madison, Wisconsin 53706, USA;* [b]*Department of Statistics, North Carolina State University, Raleigh, NC 27613, USA*

(Received 17 December 2003; Revised 09 March 2004; In final form 23 March 2004)

Dedicated to Olvi Mangasarian on the occasion of his 70th birthday

We consider a non-parametric penalized likelihood approach for model building called likelihood basis pursuit (LBP) that determines the probabilities of binary outcomes given explanatory vectors while automatically selecting important features. The LBP model involves parameters that balance the competing goals of maximizing the log-likelihood and minimizing the penalized basis pursuit terms. These parameters are selected to minimize a proxy of misclassification error, namely, the randomized, generalized approximate cross validation (ranGACV) function. The ranGACV function is not easily represented in compact form; its functional values can only be obtained by solving two instances of the LBP model, which may be computationally expensive.

A grid search is typically used to find appropriate parameters, requiring the solutions to hundreds or thousands of instances of the LBP model. Since only parameters (data) are changed between solves, the resulting problem is a non-linear slice model in the parameter space. We show how slice-modeling techniques significantly improve the efficiency of individual solves and thus speed-up the grid search. In addition, we consider using derivative-free optimization algorithms for parameter selection, replacing the grid search. We show how, by seeding the derivative-free algorithms with a coarse grid search, these algorithms can find better solutions with fewer function evaluations.

Our interest in this area comes directly from the seminal work that Olvi and his collaborators have carried out designing and applying optimization techniques to problems in machine learning and data mining.

Keywords: Model building; Slice models; Parameter selection; Basis pursuit

1 INTRODUCTION

A large body of research focuses on medical applications involving large databases of raw data. A common goal in such applications is to use the raw data for classification purposes in order to predict outcomes. Another goal is variable selection (feature selection in machine language terminology), where important variables related to the outcome are identified. In classification, we use features (such as blood pressure, height, weight, age, etc.) to classify patients into classes related to the outcomes (such as recurrent cancer or not, disease susceptible or not, death likely

* Corresponding author. E-mail: ferris@cs.wisc.edu
† E-mail: voelker@cs.wisc.edu
‡ E-mail: hzhang2@stat.ncsu.edu

ISSN 1055-6788 print; ISSN 1029-4937 online
DOI: 10.1080/1055678042000221719

or not, etc.). In variable selection, we look for those features which influence the outcome the most. A variety of models have been developed to achieve these goals.

One such approach makes use of linear support vector machines (SVMs) [1]:

$$\begin{aligned} \min_{w,\gamma,z} \quad & \frac{1}{2}\|w\|_2^2 + C_0 e'z \\ \text{subject to} \quad & M(Aw - \gamma) + z \geq e, \quad z \geq 0. \end{aligned} \tag{1}$$

Here, A is a matrix containing the training data (subjects by features) and M is a diagonal matrix with values ± 1, denoting the two classification labels. C_0 is a parameter weighting the importance of maximizing the margin between the classes versus minimizing the misclassification error (z) and e is a vector of ones. The solution (w, γ) is used to define a separating hyperplane $\{x | w'x = \gamma\}$ that divides the two classes from each other. Non-linear separators can also be found through the use of kernels [1,2]. Note that the classical SVM is able to select the influential observations but is not able to select important features.

Other approaches make use of statistical methods to predict outcomes. For example, consider a linear model which looks for coefficients b such that, for the ith observation,

$$y_i = b'x_i + \varepsilon_i. \tag{2}$$

Here, y_i is the outcome, x_i is the vector of observations, and ε_i represents random variations. Traditionally, b is found by a least squares fit, but link functions that describe how the expected outcome is related to $b'x_i$ can also be used to define generalized linear models [3].

These approaches focus on classification. However, they can be modified to also consider variable selection. For example, using the 1-norm linear support vector machine, we can find the k most influential features by restricting w to be nonzero in only k components; this results in a mixed-integer program. We could put similar restrictions on b in the linear model (2), like the LASSO proposed in Ref. [4]. Another approach for the linear model (2) measures the importance of each observation j after (full) classification by considering the averaged norm of $b_j x_{\cdot j}$ (see Ref. [5], where this is done for observations under a more complicated model).

In this article, we consider a statistical approach called likelihood basis pursuit (LBP). Unlike SVMs that look for a classifier, LBP derives a probability estimate for the outcome by maximizing the penalized likelihood of a non-parametric model. The penalty terms come from basis pursuit ideas and use the 1-norm of the model's variables. Similar penalization ideas can be found in Refs. [6,7]. Section 2, reviews the LBP procedure and two resulting models, a main effects model and a two-factor interaction model. Section 3 discusses fitting the model by selecting appropriate parameters under a grid search. We show how this procedure can be improved through the use of slice-modeling techniques [8,9]. Finally, Section 4 describes alternate procedures for parameter selection that make use of derivative-free optimization algorithms. Given appropriate starting information, we show that these algorithms can obtain better solutions than the grid search with fewer function evaluations.

2 LIKELIHOOD BASIS PURSUIT

(Note that in the following, the notation is chosen to be consistent with Ref. [5].) We begin with a training set of n observations, (x_i, y_i) $(i = 1, \ldots, n)$, where

$$x_i = (x_i^1, \ldots, x_i^d) \in \mathbb{R}^d$$

is the explanatory vector and

$$y_i \in \{0, 1\}$$

is the classification label (typically 1 represents occurrence of some event, and 0 represents non-occurrence). For any explanatory vector, x, we would like to determine the corresponding label y (either 1 or 0). If we knew

$$p(x) = \Pr\{y = 1|x\},$$

then the label could be determined by applying Bayes' rule for minimizing the expected misclassification rate [10]:

$$y = \begin{cases} 1 & \text{if } p(x) > \dfrac{1}{2}, \\ 0 & \text{otherwise.} \end{cases}$$

Our goal is to estimate $p(x)$ using the training data (x_i, y_i). To do this, we use the log odds ratio:

$$f(x) = \log\left(\frac{p(x)}{1 - p(x)}\right).$$

Note that p can be recovered from f by:

$$p(x) = \frac{\exp(f(x))}{1 + \exp(f(x))}.$$

2.1 Smoothing Spline Analysis of Variance Decomposition

To model f, we employ smoothing spline analysis of variance (SS-ANOVA). Smoothing spline analysis of variance is a general technique for building multivariate, non-parametric regression models. The idea behind SS-ANOVA is to decompose a function into components that satisfy generalized ANOVA side conditions [11]. Specifically, we search for f in a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ by considering the decomposition [5,12]:

$$f = b_0 + \sum_{\alpha=1}^{d} f_\alpha(x^\alpha) + \sum_{\alpha,\beta=1,\beta>\alpha}^{d} f_{\alpha\beta}(x^\alpha, x^\beta) + \text{higher-order interactions.} \tag{3}$$

Here, b_0 is constant, f_αs are main effects, and $f_{\alpha\beta}$s are two-factor interactions. Following Refs. [5,12], we assume that $f_\alpha \in \mathcal{H}^{(\alpha)}$, where $\mathcal{H}^{(\alpha)}$ is a RKHS, generated by some specified kernel. For the interaction terms, $f_{\alpha\beta} \in \mathcal{H}^{(\alpha)} \otimes \mathcal{H}^{(\beta)}$, where $\otimes$ denotes the tensor product. Then

$$\mathcal{H} = [1] \oplus \sum_{\alpha=1}^{d} \mathcal{H}^{(\alpha)} \oplus \sum_{\alpha,\beta=1,\beta>\alpha}^{d} [\mathcal{H}^{(\alpha)} \otimes \mathcal{H}^{(\beta)}] \oplus \cdots,$$

where $\oplus$ denotes the Kronecker (direct) sum.

Following Refs. [5,12], $\mathcal{H}^{(\alpha)}$ can be decomposed into a (finite-dimensional) parametric part and a smooth part (the orthogonal complement of the parametric part):

$$\mathcal{H}^{(\alpha)} = \mathcal{H}_\pi^{(\alpha)} \oplus \mathcal{H}_{\text{s}}^{(\alpha)}.$$

Similarly,

$$\mathcal{H}^{(\alpha)} \otimes \mathcal{H}^{(\beta)} = [\mathcal{H}_{\pi}^{(\alpha)} \otimes \mathcal{H}_{\pi}^{(\beta)}] \oplus [\mathcal{H}_{\pi}^{(\alpha)} \otimes \mathcal{H}_{\mathrm{s}}^{(\beta)}] \oplus [\mathcal{H}_{\mathrm{s}}^{(\alpha)} \otimes \mathcal{H}_{\pi}^{(\beta)}] \oplus [\mathcal{H}_{\mathrm{s}}^{(\alpha)} \otimes \mathcal{H}_{\mathrm{s}}^{(\beta)}].$$

Thus, we obtain an orthogonal decomposition for $\mathcal{H}$ into sums of products of finite-dimensional parametric subspaces, plus smooth main effects subspaces, plus two-factor interaction subspaces (of forms parametric $\otimes$ parametric, smooth $\otimes$ parametric, and smooth $\otimes$ smooth), and higher-order interactions subspaces [5,12]. To make the model more manageable, we can truncate the higher-order interaction terms. Then, $\mathcal{H}$ is the direct sum of a finite number of component subspaces, say Q. If R_l denotes the kernel of each component subspace, then

$$\mathcal{H} = \oplus_{l=1}^{Q} \mathrm{span}\{R_l\}.$$

Suppose the reproducing kernel of each $\mathcal{H}^{(\alpha)}, \alpha = 1, \ldots, d$, is K. Then for the 'main effect' component subspace $\mathcal{H}^{(\alpha)}$, we have $R(x_i, \cdot) = K(x_i^{(\alpha)}, \cdot), i = 1, \ldots, n$. For the 'two-way interaction' component subspace $\mathcal{H}^{(\alpha)} \otimes \mathcal{H}^{(\beta)}$, $R(x_i, \cdot)$ takes the form either $K(x_i^{(\alpha)}, \cdot)K(x_i^{(\beta)}, \cdot)$ or $K(x_i^{(\alpha)}, \cdot)K(x_j^{(\beta)}, \cdot), i, j = 1, \cdots, n; i \neq j$.

When n is large, the searching of $\mathcal{H}$ can be computationally intensive. To reduce the load, we limit the number of basis functions for the components of $\mathcal{H}$. This parsimonious basis approach has been used extensively in non-parametric regression, see for example Refs. [13–15]. In addition, Lee and Mangasarian [16] proposed the reduced SVM (RSVM) by applying a similar idea to SVM and showed that the classification accuracy, computational times, and memory usage of RSVM are better than a conventional SVM, which explicitly depends on the entire dataset (Though RSVM and LBP are similar in that both methods randomly select a subset of observations for model building and making predictions, there is subtle difference in their procedures. In RSVM, the sub-sampling technique directly targets the kernel matrix K, while LBP selects a number of basis terms to generate some proper subspace for functional estimation).

In Ref. [5], it is noted that the number of basis terms can be much smaller than n without degrading performance. As in Ref. [5], we apply a random sampling to draw $N \leq n$ observations, denoted $\{x_{1*}, \ldots, x_{N*}\}$, for use in defining $\mathcal{H}$ (Other methods for choosing the N observations include clustering [13].). Then, rather than searching $\mathcal{H}$ for f, we search an approximated function space

$$\mathcal{H}_* = \oplus_{l=1}^{Q} \mathrm{span}\{R_l(x_{j*}, \cdot), j = 1, \ldots, N\}.$$

Initially, we consider only the main effects. This results in dropping two-factor and higher-order interaction terms from consideration in Eq. (3) and truncating $\mathcal{H}_*$ to direct sums of products of parametric subspaces plus smooth effects subspaces:

$$\mathcal{H} = \oplus_{\alpha=1}^{d} \mathrm{span}\{k_1(x^{\alpha}), K_1(x^{(\alpha)}, x_{j*}^{(\alpha)}), j = 1, \ldots, N\}.$$

Here, k_1 represents the (one-dimensional) basis for the parametric portion and K_1 represents the kernel for the smooth portion. For our examples, we take $m = 2$ in Ref. [17, equation (10.2.4)] to obtain:

$$k_1(t) = t - \frac{1}{2} \tag{4}$$

and

$$K_1(s, t) = k_2(s)k_2(t) - k_4(|s - t|), \tag{5}$$

where

$$k_2(t) = \frac{1}{2}\left(k_1^2(t) - \frac{1}{12}\right)$$

and

$$k_4(t) = \frac{1}{24}\left(k_1^4(t) - \frac{1}{2}k_1^2(t) + \frac{7}{240}\right).$$

Then, the main effects model, also referred to as the 'additive' model, for f becomes:

$$f(x) = b_0 + \sum_{\alpha=1}^{d} b_\alpha \left(x^{(\alpha)} - \frac{1}{2}\right) + \sum_{j=1}^{N}\sum_{\alpha=1}^{d} C_{j,\alpha}^{\mathrm{s}} K_1(x^{(\alpha)}, x_{j*}^{(\alpha)}). \tag{6}$$

Adding the two-factor interaction effects, the parametric portion now consists of the parametric main effects $\{k_1(x^{(\alpha)})|\alpha = 1, \ldots, d\}$ and the parametric–parametric interaction terms $\{k_1(x^{(\alpha)})k_1(x^{(\beta)})|\alpha = 1, \ldots, d; \beta > \alpha\}$. The smooth portion consists of the smooth main effects, the parametric–smooth effects, and the smooth–smooth effects. Together, this yields the 'full' model:

$$\begin{aligned} f(x) = b_0 &+ \sum_{\alpha=1}^{d} b_\alpha \left(x^{(\alpha)} - \frac{1}{2}\right) + \sum_{\alpha=1}^{d}\sum_{\alpha,\beta=1,\beta>\alpha}^{d} b_{\alpha,\beta}\left(x^{(\alpha)} - \frac{1}{2}\right)\left(x^{(\beta)} - \frac{1}{2}\right) \\ &+ \sum_{j=1}^{N}\sum_{\alpha=1}^{d} C_{j,\alpha}^{\mathrm{s}} K_1(x^{(\alpha)}, x_{j*}^{(\alpha)}) + \sum_{j=1}^{N}\sum_{\alpha=1}^{d}\sum_{\beta=1,\beta\neq\alpha}^{d} C_{j,\alpha,\beta}^{\pi\mathrm{s}} K_1(x^{(\alpha)}, x_{j*}^{(\alpha)}) \\ &\left(x^{(\beta)} - \frac{1}{2}\right)\left(x_{j*}^{(\beta)} - \frac{1}{2}\right) + \sum_{j=1}^{N}\sum_{\alpha=1}^{d}\sum_{\beta=\alpha+1}^{d} C_{j,\alpha,\beta}^{\mathrm{ss}} K_1(x^{(\alpha)}, x_{j*}^{(\alpha)}) K_1(x^{(\beta)}, x_{j*}^{(\beta)}). \end{aligned} \tag{7}$$

For notational convenience, we let

$$f_i = f(x_i)$$

and

$$[f_1, \ldots, f_n]' = f = Tb + KC, \tag{8}$$

where T contains the parametric effects and K contains the smooth effects. T and K will differ depending on whether we are considering the additive model or the full model.

Once values for the b and C variables are determined in either the additive or the full model, $f(x)$ and $p(x)$ can be found. To find the b and C values, we maximize likelihood L over the training data:

$$\log(L) = \sum_{i=1}^{n} \left[y_i f_i - \log\left(1 + \exp(f_i)\right)\right]. \tag{9}$$

Note that we would have difficulty in directly maximizing Eq. (9): when $y_i = 0$, the corresponding term of the sum is always decreasing; when $y_i = 1$, the corresponding term of the sum is always increasing. To deal with this situation, we place restrictions on the b and C variables. In such a situation, we would like to select these restrictions so that we would find the 'best' solution. To do this, we follow basis pursuit ideas and define the 'best' to mean the solution

where the b and C variables have the smallest 1-norms. This is done because the 1-norm often produces coefficients that are exactly 0 and therefore gives sparse solutions.

2.2 Basis Pursuit

Basis pursuit was originally developed by Chen *et al.* [18] for signal decomposition in overcomplete dictionaries. In the environment of overcomplete dictionaries, multiple decompositions can exist for the same signal. Basis pursuit offers a way to find a 'best' decomposition, where Chen *et al.* [18] defined 'best' to mean the solution whose coefficients had the smallest 1-norms. Using 1-norms, Chen *et al.* [18] were able to formulate linear programming problems to find a single ('the best') signal decomposition. Not only are these signal decomposition problems under basis pursuit 'easy' to solve in comparison to other methods since they use linear programming, but they also result in sparse decompositions.

Extending basis pursuit ideas to SS-ANOVA decomposition, Zhang *et al.* [5,12] penalize the log-likelihood function with the 1-norms of the variables b and C. This allows us to take advantage of the sparsity that Chen *et al.* [18] observed for signal decompositions – under basis pursuit ideas, we typically find sparse solutions. Since we obtain a sparser solution than we might otherwise, we also have some idea of which features are most important in influencing the outcomes. In Ref. [5], the importance of a particular input is measured by the size of its functional 1-norm, calculated as the average of the function values estimated at all the data points. For the main effect, we have

$$L_1(f_\alpha) = \frac{1}{n}\sum_{i=1}^{n} |f_\alpha(x_i^{(\alpha)})| = \frac{1}{n}\sum_{i=1}^{n} \left| b_\alpha k_1(x_i^{(\alpha)}) + \sum_{j=1}^{N} C_{\alpha,j} K_1(x_i^{(\alpha)}, x_{j*}^{(\alpha)}) \right|$$

(for $\alpha = 1, \ldots, d$) and for the two-factor interactions, we have

$$\begin{aligned} L_1(f_{\alpha,\beta}) &= \frac{1}{n}\sum_{i=1}^{n} \quad |f_{\alpha,\beta}(x_i^{(\alpha)}, x_i^{(\beta)})| \\ &= \frac{1}{n}\sum_{i=1}^{n} \quad \left| b_{\alpha,\beta} k_1(x_i^{(\alpha)}) k_1(x_i^{(\beta)}) + \sum_{j=1}^{N} C^{\pi s}_{j,\alpha,\beta} K_1(x_i^{(\alpha)}, x_{j*}^{(\alpha)}) k_1(x_i^{(\beta)}) k_1(x_{j*}^{(\beta)}) \right. \\ &\quad + \sum_{j=1}^{N} C^{\pi s}_{j,\beta,\alpha} K_1(x_i^{(\beta)}, x_{j*}^{(\beta)}) k_1(x_i^{(\alpha)}) k_1(x_{j*}^{(\alpha)}) \\ &\quad \left. + \sum_{j=1}^{N} C^{ss}_{j,\alpha,\beta} K_1(x_i^{(\alpha)}, x_{j*}^{(\alpha)}) K_1(x_i^{(\beta)}, x_{j*}^{(\beta)}) \right| \end{aligned}$$

(for $\beta < \alpha$). The larger the functional 1-norm, the more important that input is. (The 2-norm was found to work just as well.) When the solutions for b and C are zero, we have no contribution to the corresponding L_1 value, resulting in a smaller L_1 and hence a less important input. Thus, applying basis pursuit to the maximum likelihood problem from Eq. (9) enhances our ability to perform variable selection.

2.3 Likelihood Basis Pursuit Models

Combining the basis pursuit penalty terms with the log-likelihood objective function (9), we obtain the complete LBP models:

$$\min_{b,C} \frac{1}{n} \sum_{i=1}^{n} \left[-y_i f_i + \log(1 + \exp(f_i)) \right] + \lambda_\pi \sum_{\alpha=1}^{d} |b_\alpha| + \lambda_{\mathrm{s}} \sum_{j=1}^{N} \sum_{\alpha=1}^{d} |C_{j,\alpha}| \tag{10}$$

for the additive model with f given by Eq. (6), or

$$\begin{aligned} \min_{b,C} \frac{1}{n} \sum_{i=1}^{n} \left[-y_i f_i + \log(1 + \exp(f_i)) \right] &+ \lambda_\pi \sum_{\alpha=1}^{d} |b_\alpha| + \lambda_{\pi\pi} \sum_{\alpha=1}^{d} \sum_{\beta=1,\beta>\alpha}^{d} |b_{\alpha,\beta}| \\ &+ \lambda_{\mathrm{s}} \sum_{j=1}^{N} \sum_{\alpha=1}^{d} |C^{\mathrm{s}}_{j,\alpha}| + \lambda_{\pi\mathrm{s}} \sum_{j=1}^{N} \sum_{\alpha=1}^{d} \sum_{\beta=1,\beta\neq\alpha}^{d} |C^{\pi\mathrm{s}}_{j,\alpha,\beta}| + \lambda_{\mathrm{ss}} \sum_{j=1}^{N} \sum_{\alpha=1}^{d} \sum_{\beta=1,\beta>\alpha}^{d} |C^{\mathrm{ss}}_{j,\alpha,\beta}| \end{aligned} \tag{11}$$

for the full model with f given by Eq. (7). The LBP models are generalized versions of the LASSO penalty for non-parametric models. In these programs, λ_π, $\lambda_{\pi\pi}$, λ_{s}, $\lambda_{\pi\mathrm{s}}$, and λ_{ss} are regularization parameters that balance the trade-off between maximizing the likelihood and minimizing the penalty terms. Here, we have chosen to group terms of similar types (parametric and smooth) and to allow distinct λs for different groups. Of course, we could also choose to set the λs equal to each other. To solve Eq. (11), we use the standard technique of replacing the absolute value terms with non-negative variables constrained to be the absolute values of the b and C variables. This results in programs with non-linear objective functions and linear constraints. Using standard approaches, it can be shown that the resulting mathematical programs are bounded below by zero and convex.

3 FITTING THE MODELS

Given values for the regularization parameters, programs (10) and (11) return corresponding values for the b and C variables, and thus define f (and p). Therefore, in order to find p, we first need to choose values for the regularization parameters that give the smallest misclassification rate.

With a slight abuse of notation, let λ represent the set of regularization parameters for each model. In other words, for the additive model (10), $\lambda = (\lambda_\pi, \lambda_{\mathrm{s}})$; while for the full model (11), $\lambda = (\lambda_\pi, \lambda_{\pi\pi}, \lambda_{\mathrm{s}}, \lambda_{\pi\mathrm{s}}, \lambda_{\mathrm{ss}})$.

3.1 Generalized Approximate Cross Validation and Randomized Generalized Approximate Cross Validation

In order to find the good values for λ, we use generalized approximate cross validation (GACV). The GACV is a proxy for the comparative Kullback–Liebler (CKL) measure and is mainly used in smoothing spline models for non-Gaussian regression [see Refs. 13,19,20]. In Refs. [5,12,21], the GACV is derived for selecting λ in LBP models. Let $p_\lambda(x)$ be the estimated conditional probability function using the parameter λ and $f_\lambda(x)$ be its corresponding log odds ratio. Let $\mu_\lambda(x)$ and $\sigma^2_\lambda(x) = p_\lambda(x)(1 - p_\lambda(x))$ be, respectively, the estimated mean

and variance functions. Then, the second-order Taylor expansion of the leaving-out-one cross validation (CV) for CKL gives

$$\mathrm{GACV}(\lambda) = \frac{1}{n}\sum_{i=1}^{n}[-y_i f_\lambda(x_i) + b(f_\lambda(x_i))] + \frac{\mathrm{tr}(H)}{n}\frac{\sum_{i=1}^{n} y_i(y_i - \mu_\lambda(x_i))}{\mathrm{tr}[I - W^{1/2}HW^{1/2}]}, \tag{12}$$

where $W = \mathrm{diag}[\sigma_\lambda^2(x_1), \ldots, \sigma_\lambda^2(x_n)]$. The matrix H satisfies $f_\lambda^{y+\varepsilon} - f_\lambda^{y} \approx H\varepsilon$, where $f_\lambda^{y+\varepsilon}$ and f_λ^{y} are, respectively, the minimizers of Eq. (10) or (11) using the perturbed data $y + \varepsilon$ and the original data y. Refer to Refs. [5,21] for more details.

In practice, it is expensive to compute GACV directly since the calculation of H depends on the inversion of some large-scale matrix. We can produce randomized estimates of $\mathrm{tr}(H)$ and $\mathrm{tr}[I - W^{1/2}HW^{1/2}]$ without having any explicit calculation of these matrices based on the following theorem (which has been exploited by numerous authors, see, e.g., Ref. [22]).

> If A is any square matrix and ε is a zero mean random n-vector with independent components with variance σ_ε^2, then $(1/\sigma_\varepsilon^2)E(\varepsilon^{\mathrm{T}}A\varepsilon) = \mathrm{tr}(A)$.

In the context of ordinary smoothing spline regression for binary response data, Refs. [13,19] argued that the approximation $f_\lambda^{y+\varepsilon} - f_\lambda \approx H\varepsilon$ suggests that $(1/\sigma_\varepsilon^2)\varepsilon'(f_\lambda^{y+\varepsilon} - f_\lambda)$ provides an estimate of $\mathrm{tr}(H)$, and $(1/\sigma_\varepsilon^2)\varepsilon' W(f_\lambda^{y+\varepsilon} - f_\lambda)$ gives an estimate of $\mathrm{tr}(W^{1/2}HW^{1/2}) = \mathrm{tr}(WH)$. For the LBP models, we follow Ref. [5] by defining the randomized GACV (ranGACV) as a computable proxy for GACV:

$$\begin{aligned}\mathrm{ranGACV} = {} & \frac{1}{n}\sum_{i=1}^{n}\left[-y_i f_\lambda(x_i) + \log(1 + \exp(f_\lambda(x_i)))\right] \\ & + \frac{\varepsilon'(f_\lambda^{y+\varepsilon} - f_\lambda^{y})}{n}\frac{\sum_{i=1}^{n} y_i(y_i - \mu_\lambda(x_i))}{\varepsilon'\varepsilon - \varepsilon' W(f_\lambda^{y+\varepsilon} - f_\lambda^{y})}.\end{aligned} \tag{13}$$

Thus, to calculate ranGACV, we only need the solution to two problems: minimizing Eq. (10) once with y and once with $y + \varepsilon$. Here $\varepsilon = (\varepsilon_1, \cdots, \varepsilon_n)'$ is a zero-mean random vector of independent components with variance σ_ε^2. The ranGACV is an established procedure for selecting the good parameters in smoothing spline SS-ANOVA regression, first proposed in Ref. [13] for SS-ANOVA models in the case of binary response and further explored in Refs. [19,20].

Unlike the original problem, which is bounded below by zero, the perturbed problem may not be bounded below. Consider the case where $y_i = 1$ and $\varepsilon_i > 0$ (so that $y_i + \varepsilon_i > 1$). The ith term of the sum in $-\log(L)$ becomes:

$$-(1 + \varepsilon_i) f_i + \log(1 + \exp(f_i)) \overset{f_i \to \infty}{\longrightarrow} -\varepsilon_i f_i \longrightarrow -\infty.$$

If the penalty terms are not large enough (depending on T and K) to cancel $-\varepsilon_i f_i$, then it is possible for the perturbed problem to become unbounded. This has happened in our tests on simulated data. In such cases, we need to decrease ε_i.

In addition, two ideas help to reduce the variance of the second term in Eq. (13).

1. Choose ε as Bernoulli(0.5) taking values in $\{+\sigma_\varepsilon, -\sigma_\varepsilon\}$. This guarantees that the randomized trace estimate has the minimal variance given a fixed σ_ε^2, see Ref. [23].
2. Generate U independent perturbations $\varepsilon^{(u)}$, $u = 1, \ldots, U$, and compute U replicated ranGACVs. Their average is then used to compute the GACV estimate.

3.2 Choice of Solver

The programs (10) and (11) are not difficult programs to solve, as all of the constraints are linear and the programs are convex. However, finding ranGACV for real instances can be difficult due to the size of the data and the number of individual solves that must be performed.

The size of individual programs depends on the number of observations (n), the number of dimensions (features) for the observations (d), and the number of dimensions for the basis of the kernel (N). For testing purposes, we use simulated data, where $n = 1000$, $d = 10$, and $N = 50$. Since both matrices T and K in the calculation of f (from Eq. (8)) are dense, this results in a large number of values that must be stored to calculate f: for the additive model, T is 1000×11 and K is 1000×500; for the full model, T is 1000×56 and K is 1000×7250 (requiring approximately 58 MB of storage). Real-life data is typically even larger: n often varies between 2000 and 4000, with 10,000 being considered a large data set; d often varies between 20 and 50; and N is around $n/20$. Owing to this, we would like to avoid storing T and K – in fact, in the full model, we often cannot store K entirely.

The data that is stored also affects the number of variables and constraints in the programs. Not storing K entirely in the full model means that the f variables cannot be explicitly defined in the model but must be implicitly used. This reduces the number of variables and constraints by n but makes the definition of the objective function more complicated. We have moved the issue from one of storage to one of model formulation.

To find ranGACV, program (10) or (11) must be solved twice for every grid point – once for the original problem and once for the perturbed problem. The number of times this must be done depends on how we choose λ.

These requirements mean that we must choose a solver that does not require an explicit definition for f. In addition, we would like one that is fast and consistent since we are potentially doing many solves (depending on the choices of λ). We use MINOS [24] as the underlying solver. This choice is based on tests performed under GAMS using the original additive model on simulated data. Two GAMS models were written, one using explicit f-variables and one using only the b, C variables. Both models were submitted to GAMS for various λ under five different non-linear solvers: CONOPT, CONOPT2, MINOS, MINOS5, and SNOPT. Refer to Tables I and II for the results. It is interesting to note the varying results under the different solvers. Even different versions of the same solver (CONOPT and CONOPT2, MINOS and MINOS5) returned different results under sometimes significantly different solution times.

TABLE I Objective values obtained by solving the original additive model with explicit f-variables

λ *values*	*CONOPT*	*CONOPT2*	*MINOS*	*MINOS5*	*SNOPT*
$\lambda_\pi = 2^{-20}$ $\lambda_s = 2^{-20}$	0.541399	0.540633	0.540831	0.540616	0.540905
$\lambda_\pi = 2^{-20}$ $\lambda_s = 2^{-19}$	0.541441	0.540927	0.540995	0.540924	0.540938
$\lambda_\pi = 2^{-19}$ $\lambda_s = 2^{-20}$	0.541404	0.540630	0.540760	0.540620	0.540784
$\lambda_\pi = 2^{-19}$ $\lambda_s = 2^{-19}$	0.541446	0.540929	0.540940	0.540928	0.540943
Solve time	9.292	58.967	16.214	15.680	19.275
Total time	27.14	77.08	34.60	33.97	37.49

TABLE II Objective values obtained by solving the original additive model without explicit f-variables

λ *values*	*CONOPT*	*CONOPT2*	*MINOS*	*MINOS5*	*SNOPT*
$\lambda_\pi = 2^{-20}$ $\lambda_s = 2^{-20}$	0.541399	0.540620	0.540831	0.540831	0.540905
$\lambda_\pi = 2^{-20}$ $\lambda_s = 2^{-19}$	0.541441	0.540925	0.540995	0.540995	0.540940
$\lambda_\pi = 2^{-19}$ $\lambda_s = 2^{-20}$	0.541404	0.540628	0.540760	0.540836	0.540782
$\lambda_\pi = 2^{-19}$ $\lambda_s = 2^{-19}$	0.541446	Overflow	0.540940	0.541000	0.540945
Solve time	22.047	164.560	124.358	73.530	99.537
Total time	486.87	628.90	589.82	538.65	565.38

Only CONOPT and MINOS returned the same objective values for both GAMS models, suggesting only these two solvers were consistent on our problems. Further, MINOS's results were smaller than CONOPT's, suggesting that MINOS found a better point.

The last two rows in Tables I and II give the total solution times (resource usage) and the total overall times spent in GAMS. These results strongly suggest that using explicit f-variables improves solution efficiency. However, as noted above, including explicit f-variables increases the total number of variables and the total number of constraints (both by n) and requires that K be stored in its entirety. This causes difficulty primarily for the full model, where often K cannot be fully stored and only K_1 is stored. Further, many of the gains in solution time seen in Table I as compared to Table II result from eliminating iterations that generate non-sensical values for f. To obtain the values in Table I, we restricted the f-values to be less than or equal to 30 in absolute value. However, this was not possible for those results in Table II since f was not explicitly present in the model. However, when using the solvers directly, an intermediate comparison of the f-value can be made in the routine for evaluating the objective function, and infinity can be returned for values of f outside of reasonable ranges. Thus, we develop models that do not use explicit f-variables but garner the benefits in solution time shown in Table I. We utilize MINOS in our subsequent work with the modifications outlined above.

3.3 Parameter Search: Grid Search

Our goal is to choose the λ values that minimize the ranGACV measure. For each set of λs, we need to solve two non-linear programming problems (an original and a perturbed problem) in order to determine the corresponding value for ranGACV. Thus, we have a nested set of minimization problems, where the inner minimization (solving model (10) or (11)) is a straight non-linear problem, and the outer minimization (minimizing ranGACV) makes use of these results. Note that the outer minimization is almost constraint-free: the only constraint is that $\lambda > 0$ since λ represents weight on penalty parameters.

The simplest procedure to find ranGACV is to divide the λ-space into a grid and calculate ranGACV at each grid point. Once the calculations are complete, we can choose the λ values by selecting the grid point for which ranGACV achieved a minimum. Essentially, we change the problem to a minimization over a finite set. Because of this, we may not find the true minimum. However, this procedure is always implementable although costly depending on the

grid size. We use a grid of varying step size: we move from one set of λ values to the next by dividing the λ values in half. This provides a coarse grid for large values of λ (where the focus of the model is on minimizing the penalty parameters), and a fine grid for small values of λ (where the focus is on maximizing the likelihood). We consider λ ranging from 2^{-1} to 2^{-20}.

Under the grid search, we may have hundreds or thousands of individual solves depending on the range for and number of λ. In order to obtain solutions in a reasonable amount of time, we need to employ an efficient solution approach. As in the problems presented in Ref. [8], we can consider the values of λ and the choice of y to be individual slices of data, as only these values change between solves. Thus, LBP can be reduced to an example of non-linear slice modeling [9]. We have a series of programs, one for each type (original or perturbed) and grid point, that only differ in the data used to define them (λ, y). By applying slice modeling ideas from Ref. [8], namely, maintaining program structure and common data between solves and using previous solutions as starting points for later solves, we can improve efficiency and make the grid search usable.

Under MINOS, non-linear programs are specified in three pieces: the linear portion, the non-linear objective function, and the non-linear constraints. Originally, MINOS required the linear portion of the program to be specified by an MPS file; later versions of MINOS include the subroutine `minoss`, that reads the linear portion from parameters. Using `minoss`, we are able to specify and store the linear portion of the LBP programs internally, eliminating the need to write a new MPS file every time we move to a new grid point (i.e., change the λs). Besides saving us time in accessing files, it also enables us to hold the program structure constant throughout all solves.

The non-linear objective function is specified by the subroutine `funobj`. Besides including the non-linear objective portion, we also include all f-related terms in `funobj` since we do not model f explicitly. Not only does this provide for easy changes between the original and perturbed solves (simply specify which y values to use), this also simplifies the f-calculations: we only need to calculate f inside of `funobj` rather than in the linear portion as well.

We can speed up individual solves in MINOS using a hot start. On the first solve, we use a cold start. The cold start instructs MINOS to use a crash procedure to determine an initial basis [24]. All subsequent solves use a hot start. Under the hot start, not only is the basis from the previous call maintained (as in a warm start), but also are the LU factors of the basis, the approximate reduced Hessian, and the column and row scales [24]. This is particularly advantageous for the models (10) and (11) since the constraint set does not change between solves. This significantly speeds up subsequent individual solves.

Table III compares the times for solving the additive model (10) on three simulated problems of increasing size. For all problems, we let λ_π and λ_s range from 2^{-1} to 2^{-5}, for a total of 25

TABLE III Time comparison on additive problems

Problem	*Original*	*Slice techniques*	*Separate solves*
$n = 500, d = 5,$ $N = 25$	1.5 h	12 s	6 s
$n = 1000, d = 10,$ $N = 50$	23 h	4 min	40 s
$n = 2000, d = 20,$ $N = 100$	72+ h	17 min	7 min

Note: n is the number of observations, d is the number of features, and N is the number of dimensions for the basis of the kernel.

solves. The first column (original) displays the times for solving the model using the original code. This code uses the subroutine `minos1`, which requires an MPS file to written for each program. In addition, for each set of λ values, it solves the original and perturbed versions together followed by the ranGACV calculation. Note that for the largest problem, we never actually completed all 25 solves.

The second column (slice techniques) displays the results for applying the slice modeling approach to LBP. In this code, we use the subroutine `minoss`. This eliminates the need for MPS files, allowing us to store the programs internally and maintain program structure between solves. In addition, we move all f-related terms to `funobj`, and we request hot starts for all but the first solve. Like the original code, this version solves the original program, followed immediately by the perturbed program for each set of λ values. As can be seen from Table III, moving to an internal problem representation gives a significant time improvement, requiring only minutes instead of hours to solve all 25 programs.

We note that we can further improve efficiency by considering the order of the solves. Once we have the solutions for the original and perturbed problems at a particular grid point, ranGACV can be calculated. This suggests the approach of solving the original and perturbed problems together for each grid point, as we have done. However, the slice modeling approach suggests the opposite: because fewer changes take place moving from one grid point to another while maintaining the problem type, previous solutions will have greater impact on future solves if the original and perturbed solves are separated. Such separation requires extra storage: we must store the b and C variable values for each solve so that ranGACV can be calculated later. If these solution files become too large, multiple solves must be done, each time using only a piece of the grid. However, we do improve the solution time, as can be seen in Table III. This also allows for much easier parallel implementation – now the original and perturbed problem can be run in parallel.

The final column (separate solves) displays the times for ordering the solves according to the slice modeling approach. The only difference between this approach and that of the second (slice techniques) approach is that the original and perturbed programs are solved separately (The times displayed include the times to calculate the results, after all solves have finished.). Solving the problems separately reduces the improved times by half or more. This timesavings enabled us to solve more complicated models, such as the full model (11) or the categorical models discussed in Ref. [5].

4 ALTERNATIVE PARAMETER SELECTION METHODS

Using the simple grid search for the parameter selection, we are guaranteed to find a minimum over the set of λs that we consider. Considering a different set of λs may result in a better or worse ranGACV value. Thus, we must consider a large set of λs in order to guarantee a relatively good solution. The approach we use above, varying the size of the steps between grid points, allows us to focus more on areas where ranGACV is small. When λ is large (near 2^{-1}), the penalty terms dwarf the likelihood terms, and the b, C variables are set equal to zero in the solution (the only non-zero variable is the model's constant term, b_0, which is not penalized). Thus, we end up ignoring the likelihood calculation completely and so ranGACV is likely to be large (since maximum likelihood is small). When λ is small, the penalty parameters are small enough that likelihood is considered and so ranGACV decreases. By using a finer grid for smaller λ, we are more likely to obtain a small value of ranGACV. Although this grid search technique reduces the number of ranGACV calculations we must do, we are still not guaranteed a minimum ranGACV – another set of λs may produce a smaller ranGACV.

Further under a grid search, we consider all combinations of λ values. We cannot adapt the search to concentrate on areas where ranGACV is likely to be small because we have no mechanism for determining such areas. This results in a number of wasted calculations. To reduce the overall solution time, we would like to use an efficient and fast search method to determine the λ values that give the minimum ranGACV. In other words, we would like an alternative method for performing the outer minimization.

4.1 Derivative-Free Search Methods

Rather than using a grid search, we consider applying optimization approaches to select appropriate parameters for our models. We employ three derivative-free search methods: Nelder–Mead simplex method [25,26], Powell's UOBYQA method [27], and the wedge trust region method [28]. All three methods minimize an unconstrained function of several variables. Under these methods, the λ values are the variables and the ranGACV calculation is the objective function θ. Because these methods are derivative-free, we do not need an explicit functional formulation for the objective function; rather, we can specify the objective function as a subroutine. Input to our objective function subroutine are the λ values; output is the corresponding ranGACV value. The two non-linear MINOS solves are contained within this subroutine, one for the original model and one for the perturbed model.

The Nelder–Mead simplex method is a direct search method. It attempts to minimize a function of p real variables using only function values, not derivative information. To do this the method maintains a simplex, the convex hull of $p + 1$ distinct points (vertices). We use the implementation provided in the routine `fminsearch` in MATLAB [29].

UOBYQA stands for unconstrained optimization by quadratic approximation. This method forms quadratic models for the objective function θ by interpolating $m = (p + 1)(p + 2)/2$ values [27]. Usually, an iteration consists of finding a new vector of variable values by either the minimization of the quadratic model subject to a trust region bound (a trust region step), or a procedure that is used to improve the accuracy of the quadratic model (a model step). For implementation, we use the Fortran code provided by Powell [30].

In a similar fashion to the UOBYQA method, the wedge trust region method generates a model (either linear or quadratic) for θ by interpolating the function at a series of points. The difference lies in the trust region problem. To ensure that the model is well-defined, the geometric condition that the interpolation points are linearly independent (non-degenerate) must be satisfied. Other methods (like Powell's UOBYQA) include a model step that improves the accuracy of the model if the geometric condition fails. Marazzi and Nocedal [28] instead incorporate the geometric condition explicitly in the trust region step. When the model is linear, this constraint takes on the form of a wedge (hence the name). To implement this algorithm, we use the MATLAB code for the quadratic model provided by Marazzi [31], with (default values) $\alpha = 0.25$, $\beta = 0.75$, and $\gamma = 0.4$.

4.2 Results

We compare the three derivative-free search methods to each other and to grid searches on three problems under the additive model (10). The first problem uses simulated data with $n = 1000$, $d = 10$, and $N = 50$. Figure 1 shows ranGACV from a 20×20 grid search ($\lambda = 2^{-1}, 2^{-2}, 2^{-3}, \ldots, 2^{-20}$). Displayed are plots for the log–log (base 2) scales. The other two problems use data drawn from medical applications with $n = 669$, $d = 6$, $N = 40$ and $n = 668$, $d = 6$, and $N = 40$, respectively. Figures 2 and 3 show ranGACV from the 20×20 grid search on these two (real) problems under log–log (base 2) scales.

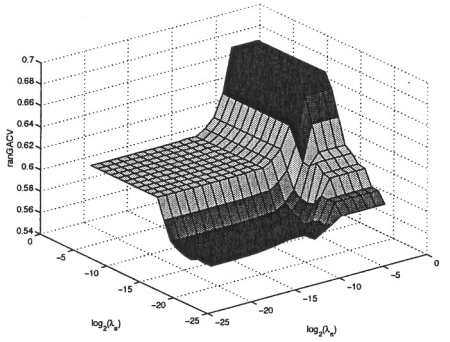

FIGURE 1 ranGACV for the simulated data problem.

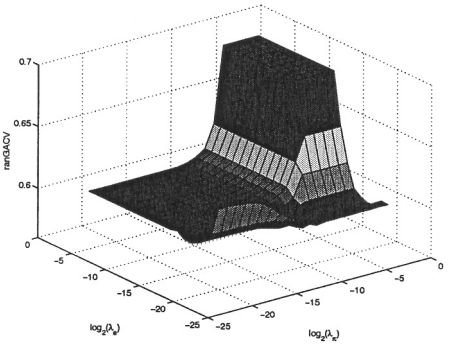

FIGURE 2 ranGACV for the first real data problem.

We use the log–log (base 2) scale to obtain a nicer function to minimize. This results in a change of variables:

$$x_1 = \log_2(\lambda_\pi) \quad \text{and} \quad x_2 = \log_2(\lambda_s).$$

Note that due to this change of variables, we do not have to worry about maintaining $\lambda > 0$: since we apply the algorithms in the log–log (base 2) space, all corresponding λ values will be $2^x > 0$.

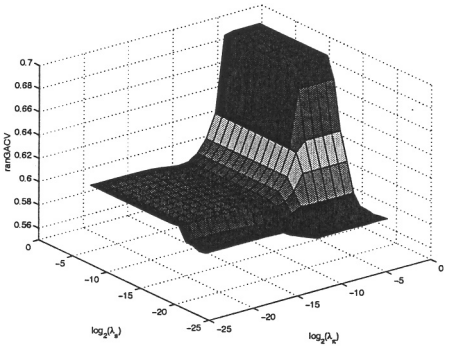

FIGURE 3 ranGACV for the second real data problem.

To obtain appropriate starting points for the derivative-free methods, we consider multiple grid searches. We start with a coarse 5×5 grid:

$$x_1, x_2 \in \{-4, -8, -12, -16, -20\}.$$

Then, we try a finer 10×10 grid:

$$x_1, x_2 \in \{-2, -4, -6, \ldots, -18, -20\}$$

Finally, we consider the typical 20×20 grid:

$$x_1, x_2 \in \{-1, -2, -3, \ldots, -19, -20\}.$$

We start the derivative-free methods at the best grid point found under each grid search.

Initial trust region radii for the UOBYQA and wedge methods are set large for the coarse grids and are decreased on the finer grids: for the 5×5 grid, the radii are set to 4; for the 10×10 grid, the radii are set to 2; and for the 20×20 grid, the radii are set to 1. This is done because we expect the grid search to come closer to the minimum as the number of grid points is increased.

All three search methods are allowed to run until the tolerances were satisfied, so any maximum iteration and maximum function evaluation limits are set large enough to be ignored. For the Nelder–Mead simplex method, we set tol x = tol $f = 1 \times 10^{-8}$. For the trust region methods, the terminating trust region radii are also set to 1×10^{-8}.

Tables IV–VI display the ranGACV values obtained. In addition, the total number of function evaluations are displayed as a sum of two numbers. The first number represents the number of function evaluations for the initial grid search and the second number represents the number of function evaluations to reach the solution, starting from the point found by the grid search. The best ranGACV value is emphasized in each table.

In all cases, the derivative-free search methods improved upon the corresponding grid search value. In almost all cases (except for UOBYQA on the coarse grid in the second real problem), the methods also improved upon all of the grid searches. This suggests that even a coarse grid search is sufficient for obtaining appropriate starting points (Note that starting the methods cold from no grid search point resulted in worse solutions, so some initial search must be done to obtain a reasonable starting point.). Further, the derivative-free search methods on coarser grids required significantly fewer *total* function evaluations than the 400 needed for the finest 20×20 grid search. The maximum total function evaluations on the 5×5 grid was $25 + 122 = 147$; the maximum total function evaluations on the 10×10 grid was $100 + 136 = 236$. Based

TABLE IV Search method results for the simulated data problem

	5 × 5 Grid	*10 × 10 Grid*	*20 × 20 Grid*
Best grid search point	(−12, −16)	(−12, −16)	(−12, −16)
Grid search	0.55029982 (25 func. evals)	0.55029982 (100 func. evals)	0.55029982 (400 func. evals)
Nelder–Mead	0.54981480 (25 + 122 func. evals)	0.54981480 (100 + 122 func. evals)	0.54981480 (400 + 122 func. evals)
UOBYQA	0.54978822 (25 + 86 func. evals)	0.54978498 (100 + 85 func. evals)	0.54982361 (400 + 77 func. evals)
Wedge	0.54979829 (25 + 45 func. evals)	0.54986966 (100 + 38 func. evals)	*0.54968777* (400 + 40 func. evals)

TABLE V Search method results for the first real data problem

	5 × 5 Grid	*10 × 10 Grid*	*20 × 20 Grid*
Best grid search point	(−8, −16)	(−10, −16)	(−10, −16)
Grid search	0.58289425 (25 func. evals)	0.58005256 (100 func. evals)	0.58005256 (400 func. evals)
Nelder–Mead	*0.57862580* (25 + 112 func. evals)	0.57934657 (100 + 80 func. evals)	0.57934657 (400 + 80 func. evals)
UOBYQA	0.57872897 (25 + 82 func. evals)	0.57936501 (100 + 88 func. evals)	0.57899692 (400 + 111 func. evals)
Wedge	0.57931261 (25 + 68 func. evals)	0.57871935 (100 + 56 func. evals)	0.57883227 (400 + 54 func. evals)

TABLE VI Search method results for the second real data problem

	5 × 5 Grid	*10 × 10 Grid*	*20 × 20 Grid*
Best grid search point	(−8, −16)	(−10, −16)	(−9, −16)
Grid search	0.57265086 (25 func. evals)	0.57103974 (100 func. evals)	0.56970404 (400 func. evals)
Nelder–Mead	*0.56792204* (25 + 98 func. evals)	0.56811498 (100 + 73 func. evals)	*0.56792204* (400 + 102 func. evals)
UOBYQA	0.57113598 (25 + 95 func. evals)	0.56792960 (100 + 136 func. evals)	0.56792235 (400 + 80 func. evals)
Wedge	0.56793007 (25 + 48 func. evals)	0.56792988 (100 + 51 func. evals)	0.56814486 (400 + 46 func. evals)

on these results, we suggest using a coarse 5 × 5 grid search to obtain a good starting point, then applying one of the derivative-free methods to finish the outer optimization. Any of the three methods could be used. However, the Nelder–Mead method found the best point in the two real data problems. The wedge method though, required fewer function evaluations (generally around half).

As the tables show, improved starting points do not always result in better solutions. For example, the UOBYQA method is worse at the 10 × 10 grid starting point than at the 5 × 5 grid starting point on the first real problem. Similarly, the Nelder–Mead method is worse at the 10 × 10 grid starting point than at the 5 × 5 grid starting point on the second real data problem. In addition, on this problem, the wedge method actually obtains the worst solution of the derivative-free methods at the 20 × 20 grid starting point. These results suggest that a starting point farther from the solution may result in a better model being built and thus a better final value (The trust region methods may also be improved by larger initial trust region radii.).

5 CONCLUSION

In this article, we have considered a modeling approach called LBP. Under LBP, a model for predicting the probability of an outcome is obtained. This is done using a SS-ANOVA model to

decompose the log odds ratio into parametric and smooth components. Weights on the various components are determined by maximizing a penalized likelihood function.

As we discussed, the final LBP model can only be obtained after appropriate penalty parameters are found. We required penalty parameters that minimized the misclassification error for the current problem. To find these parameters, we initially considered using a grid search. Under the grid search, the search for penalty parameters resulted in a slice model, and slice modeling techniques from Ref. [8] were employed. We also showed that the penalty parameter search could be improved, both by being made more efficient (considering the number of calculations for misclassification) and by finding better points, through the use of derivative-free optimization techniques. These techniques could be implemented to employ some of the slice modeling techniques, namely, maintaining program structure and starting later solves from earlier solutions, as they also require multiple solutions to data-modified programs.

Acknowledgments

We would like to thank Grace Wahba and Yi Lin for introducing us to LBP and providing us with their example problems. We would also like to thank Ronald and Barbara Klein for additional example problems. This material is based on research partially supported by the National Science Foundation under grant CDA-9726385 and the Air Force Office of Scientific Research under grant F49620-01-1-0040. Part of this work was supported by Zhang's advisor Professor Grace Wahba under the National Science Foundation grant DMS-0072292 and National Institute of Health grant EY09946.

References

[1] O.L. Mangasarian (2000). Generalized support vector machines. In: A. Smola, P. Bartlett, B. Schölkopf and D. Schuurmans (Eds.), *Advances in Large Margin Classifiers*, pp. 135–146. MIT Press, Cambridge, Massachusetts.

[2] G. Fung, O.L. Mangasarian and A. Smola (2002). Minimal kernel classifiers. *Journal of Machine Learning Research*, 303–321.

[3] G. Johnston (1993). SAS software to fit the generalized linear model. In *SUGI Proceedings*. Available as PDF document from http://www.sas.com/rnd/app/papers/abstracts/genmod.html.

[4] R.J. Tibshirani (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, B*, **58**, 267–288.

[5] H. Zhang, G. Wahba, Y. Lin, M. Voelker, M. Ferris, R. Klein and B. Klein (2004). Variable selection and model building via likelihood basis pursuit. *Journal of American Statistical Association, forthcoming*.

[6] P.S. Bradley and O.L. Mangasarian (1998). Feature selection via concave minimization and support vector machines. In: J. Shavlik, (Ed.), *Machine Learning Proceedings of the Fifteenth International Conference(ICML'98)*, pp. 82–90, Morgan Kaufmann, San Francisco, California. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/98-03.ps.

[7] O.L. Mangasarian and D.R. Musicant (2000). Robust linear and support vector regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22**, 950–955.

[8] M.C. Ferris and M.M. Voelker (2002). Slice models in general purpose modeling systems: An application to DEA. *Optimization Methods and Software*, **17**, 1009–1032.

[9] M.M. Voelker (2002). *Optimization of Slice Models*. PhD thesis, University of Wisconsin, Madison, Wisconsin.

[10] Y. Lin, G. Wahba, H. Zhang and Y. Lee (2002). Statistical properties and adaptive tuning of support vector machines. *Machine Learning*, **48**, 115–136.

[11] G. Wahba, Y. Wang, C. Gu, R. Klein and B. Klein (1995). Smooth spline ANOVA for exponential families, with application to the Wisconsin epidemiological study of diabetic retinopathy. *Annals of Statistics*, **23**(6), 1865–1895.

[12] H. Zhang, G. Wahba, Y. Lin, M. Voelker, M. Ferris, R. Klein and B. Klein (2001). Variable selection via basis pursuit for non-Gaussian data. In *2001 Proceedings of the American Statistical Association, Biometrics Section [CDROM]*, American Statistical Association, Alexandria, VA.

[13] D. Xiang and G. Wahba (1998). Approximate smoothing spline methods for large data sets in the binary case. In *Proceedings of ASA Joint Statistical Meetings, Biometrics Section*, pp. 94–99.

[14] D. Ruppert and R.J. Carroll (2000). Spatially-adaptive penalties for spline fitting. *Australian and New Zealand Journal of Statistics*, **45**, 205–223.

[15] P. Yau, R. Kohn and S. Wood (2003). Bayesian variable selection and model averaging in high dimensional multinomial nonparametric regression. *Journal of Computational and Graphical Statistics*, **12**, 23–54.
[16] Y-J. Lee and O.L. Mangasarian (2000). Reduced support vector machines. Data Mining Institute Technical Report 00-07, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 2000. Appeared in CD Proceedings of the SIAM International Conference on Data Mining, Chicago, April 5–7, 2001, SIAM.
[17] G. Wahba (1990). *Spline Models for Observational Data*. SIAM, Philadelphia, PA.
[18] S.S. Chen, D.L. Donoho and M.A. Saunders (1998). Atomic decomposition by basis pursuit. *SIAM Journal on Scientific Computing*, **20**, 33–61.
[19] X. Lin, G. Wahba, D. Xiang, F. Gao, R. Klein and B. Klein (2000). Smoothing spline ANOVA models for large data sets with Bernoulli observations and the randomized GACV. *The Annals of Statistics*, **28**, 1570–1600.
[20] C. Gu (2002). *Smoothing spline ANOVA models*. Springer-Verlag.
[21] H. Zhang (2002). *Nonparametric Variable Selection and Model Building Via Likelihood Basis Pursuit*. PhD thesis, Department of Statistics, University of Wisconsin, Madison, WI.
[22] D. Girard (1998). Asymptotic comparison of (partial) cross-validation, GCV and randomized GCV in nonparametric regression. *The Annals of Statistics*, **26**, 315–334.
[23] M. Hutchinson (1989). A stochastic estimator for the trace of the influence matrix for Laplacian smoothing splines. *Commun. Statist.-Simula.*, **18**, 1059–1076.
[24] B.A. Murtagh and M.A. Saunders (1983). MINOS 5.0 user's guide. Technical Report SOL 83.20, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California. Available as PDF document from http://www.sbsi-sol-optimize.com/manuals.htm.
[25] J.A. Nelder and R. Mead (1965). A simplex method for function minimization. *Computer Journal*, **7**, 308–313.
[26] J.C. Lagarias, J.A. Reeds, M.H. Wright and P.E. Wright (1998). Convergence of the Nelder–Mead simplex method in low dimensions. *SIAM Journal on Optimization*, **9**, 112–147.
[27] M.J.D. Powell (2002). UOBYQA: unconstrained optimization by quadratic approximation. *Mathematical Programming*, **92**, 555–582.
[28] M. Marazzi and J. Nocedal (2002). Wedge trust region methods for derivative free optimization. *Mathematical Programming*, **91**, 289–305.
[29] MathWorks, Inc. FMINSEARCH. MATLAB function. Obtained from the MATLAB command 'type fminsearch' (October 2002).
[30] M.J.D. Powell. Private communication via email, November 2001.
[31] Marcelo Marazzi. Private communication via email, October 2001.