Mach Learn (2011) 85:175–208
DOI 10.1007/s10994-011-5246-7

Model selection for primal SVM

Gregory Moore · Charles Bergeron · Kristin P. Bennett

Received: 28 February 2010 / Accepted: 1 March 2011 / Published online: 22 April 2011

Abstract This paper introduces two types of nonsmooth optimization methods for selecting model hyperparameters in primal SVM models based on cross-validation. Unlike common grid search approaches for model selection, these approaches are scalable both in the number of hyperparameters and number of data points. Taking inspiration from linear-time primal SVM algorithms, scalability in model selection is achieved by directly working with the primal variables without introducing any dual variables. The proposed implicit primal gradient descent (ImpGrad) method can utilize existing SVM solvers. Unlike prior methods for gradient descent in hyperparameters space, all work is done in the primal space so no inversion of the kernel matrix is required. The proposed explicit penalized bilevel programming (PBP) approach optimizes both the hyperparameters and parameters simultaneously. It solves the original cross-validation problem by solving a series of least squares regression problems with simple constraints in both the hyperparameter and parameter space. Computational results on least squares support vector regression problems with multiple hyperparameters establish that both the implicit and explicit methods perform quite well in terms of generalization and computational time. These methods are directly applicable to other learning tasks with differentiable loss functions and regularization functions. Both the implicit and explicit algorithms investigated represent powerful new approaches to solving large bilevel programs involving nonsmooth loss functions.

Keywords Model selection · Cross-validation · Nonconvex optimization · Bilevel programming · Support vector machines

Editors: Süreyya Özöğür-Akyüz, Devrim Ünay, and Alex Smola.

G. Moore (✉) · C. Bergeron · K.P. Bennett
Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180, USA
e-mail: mooregm@gmail.com

C. Bergeron
e-mail: chbergeron@gmail.com

K.P. Bennett
e-mail: bennek@rpi.edu

1 Introduction

Support vector machines (SVM) are a popular approach to machine learning. They are simple to implement, theoretically sound, and easy to customize to different tasks such as classification, regression, ranking and so forth. Primal SVM algorithms are very fast. Recent approaches directly optimize the unconstrained possibly nonsmooth SVM loss function using various subgradient-based methods including cutting plane and bundle approaches. They achieve primal SVM methods that are linearly scalable in the number of training points (Joachims 2006; Teo et al. 2007). Hence, optimal solutions to SVM problems may be calculated effectively and efficiently.

One critical difficulty that remains for a machine learning practitioner is model selection, and many of the choices that need to be made appear as one or several hyperparameters. Determining appropriate values thereof is a challenge in view of obtaining a final model that is generalizable. Cross-validating over a grid is a simple and almost universal strategy, but its computational complexity scales exponentially with the number of hyperparameters *and* the number of grid points for each hyperparameter. This discourages the formulation of sophisticated learning tasks requiring several hyperparameters. At the same time, it requires a possibly inexperienced SVM user to select a grid range and coarseness for each hyperparameter which may be sensitive with respect to finding an optimal (or almost optimal) solution. Clearly, the development of alternative model selection strategies is an important issue in machine learning today. This paper takes a step towards automating the model selection task.

T-fold cross-validation (CV) is the classic approach for estimating out-of-sample generalization error in modeling. The method involves breaking up the modeling set into T partitions. $T-1$ partitions are used to train the model, and the remaining partition is used to validate the model. This is repeated T times, so that each partition is used in validation once. The average validation error across the folds is returned. CV can be applied to arbitrary data mining problems, producing a good estimate of generalization error on new data. Model generalization is reported using new data disjoint from the modeling set.

CV is naturally and precisely formulated as a bilevel program (BP), as shown in Fig. 1. In the bilevel CV problem (BCP), the inner-level problems are each of the T-fold CV training problems which must be optimal for the hyperparameters. The objective of the outer-level is the validation error. Early work using BCP included generalized CV which looks at model selection of a single parameter in Ridge Regression (Golub et al. 1979). The extensive literature on models selection is too large to review. Therefore we focus on optimization-based approaches for BCP problems involving many hyperparameters and typical SVM-type losses that may involve nonsmooth functions such as absolute value and maximum.

The most widely used strategies for solving the BCP (like grid search) treat SVM model parameters as an **implicit** function of the hyperparameters. The BCP is transformed to a

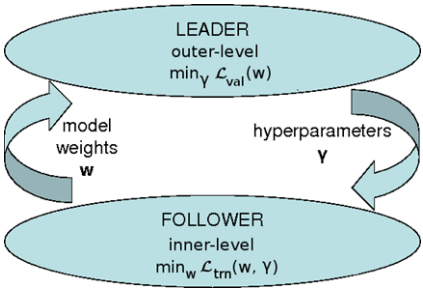

Fig. 1 The bilevel cross-validation problem for a single validation and training split. The outer-level leader problem selects the hyperparameters, γ, to perform well on a validation set. The follower problem trains an optimal inner-level model for the given hyperparameters, and returns a weight vector $\mathbf{w}$ for validation

minimization problem of the implicit validation function over the hyperparameter space and any desired optimization algorithm can be used to optimize this function. However, each evaluation of the implicit validation function is very costly because it involves solving the inner training problems to optimality. In contrast, many general purpose bilevel programming algorithms in the mathematical programming literature (Colson et al. 2007) treat a bilevel program as a function of all its variables and implicit strategies are known but not as widely used. A very successful bilevel programming algorithm strategy is to simultaneously solve for the hyperparameters and inner parameters (Dempe 2003). We will call these simultaneous strategies **explicit** methods to contrast them with their implicit counterparts. Intuitively, explicit strategies that simultaneously solve for the learning models and hyperparameters are much more appealing than grid or other implicit function strategies because there is no need to fully solve the inner training problems when the hyperparameters are far from optimal.

Implicit methods work by alternating between searching the hyperparameter space and solving the SVM problem. Grid search, discussed in the previous paragraph, does this by choosing a discrete number of hyperparameter combinations and solving the SVM model for each one. The optimal hyperparameters are based on which combination achieves optimal performance in terms of validation; the grid point that gives the best average validation error is chosen to be the best model. One advantage of grid search is that it can exploit state-of-the-art solvers to solve the SVM subproblems at each grid point. But this method has obvious theoretical weaknesses (working from a discrete sampling of possible hyperparameters, the best model found is undoubtedly not optimal) and computational inefficiencies (high accuracy requires a fine grid). Moreover, its computational complexity grows exponentially with the number of hyperparameters, rendering this method intractable for more complex modeling tasks. Despite its heuristic nature and high computational complexity, it remains widely used due to its simplicity and the fact that it can use any SVM solver on the base subproblems.

Other implicit methods use a more intelligent search over the grid hyperparameters which involves alternating between solving a problem with fixed hyperparameters, taking a hyperparameter step in a direction of decrease, and then re-solving the problem using the new fixed hyperparameters. The direction of decrease can be determined by using gradient or gradient-free approaches. Gradient-free approaches (Eitrich and Lang 2006; Momma and Bennett 2002) have worked well in practice for small numbers of hyperparameters, but, as in grid search, the computational complexity grows exponentially in the hyperparameter size. Primal gradient-based methods have been successful for BCP with many points, many hyperparameters, and twice differentiable inner loss functions (Do et al. 2008). For SVM type problems with loss functions that are not twice differentiable, dual SVM implicit gradient methods have worked very well for problems with many hyperparameters (Chapelle et al. 2002; Keerthi et al. 2006). Unfortunately these algorithms require inversion of the kernel matrix, and thus have limited scalability in the dataset size. In this paper, we introduce a new implicit primal gradient approach (ImpGrad) for loss functions that are not twice differentiable. Further ImpGrad eliminates the need for inversion of the kernel, thus it is scalable in the dataset size.

Explicit methods simultaneously solve for the model weights and hyperparameters. The first explicit bilevel programming approach used the smoothed version of the SVM problem and treated the problem as a mathematical program with equilibrium constraints (MPEC) (Bennett et al. 2006; Kunapuli et al. 2008). The MPEC approach can handle many hyperparameters and generates good generalization errors when compared with grid search. However the MPEC method introduces dual variables and constraints for each of the data

points, thus it is not scalable in the dataset size. Inspired by linear-time nonsmooth primal SVM algorithms, we rewrite the bilevel program as a polyhedrally constrained nonsmooth problem using a penalty strategy. We call the resulting algorithm penalized bilevel program (PBP) and reported preliminary results in Moore et al. (2009).

This paper examines how to optimize multiple hyperparameters (2 or more) using cross-validation for primal linear SVM-type problems. The ultimate goal is to produce algorithms that are scalable both in the number of hyperparameters and the number of training points. Taking our queue from recent nonsmooth approaches for SVM, we produce both nonsmooth implicit and explicit methods for hyperparameter selection. Implicit Primal Gradient Descent (ImpGrad) generalizes an alternating implicit algorithm for the dual problem (Keerthi et al. 2006) to the primal nonsmooth SVM. Penalized Bilevel Programming (PBP) uses a partial linearization strategy to produce an effective and efficient explicit method. PBP represents a significant advance in finding local solutions of general bilevel programs.

We compare ImpGrad and PBP against alternative implicit and explicit methods, grid search and MPEC respectively on least squares support vector regression problems with up to ten hyperparameters. Experiments are performed on cheminformatics datasets. Using a support vector regression loss function that treats subgroups of the modeling set differently, we analyze the benefits and drawbacks to this many-hyperparameter model. We show that grid search is not applicable to this problem because it is too slow and show the advantages and disadvantages of the ImpGrad and PBP algorithms.

The remainder of this paper is outlined as follows: Sect. 2 introduces the notation used throughout this paper. Section 3 introduces the bilevel cross-validation framework. Section 4 details support vector regression. Section 5 describes ImpGrad and give details for implementing SVR. Section 6 summarizes PBP. Section 7 details the nonsmooth bilevel approach given by this paper and gives details for implementing a multiple hyperparameter version of SVR. Sections 8 and 9 detail the experimental design, results and discussion for the datasets considered in this paper. Finally, Sect. 10 concludes.

2 Notation and definitions

Let $\mathbb{R}$ denote the set of real numbers, with $\mathbf{x} \in \mathbb{R}^n$ a column vector and $\mathbf{x}'$ its transpose. The family of vector q-norms is defined as $\|\mathbf{x}\|_q = \sqrt[q]{\sum_{j=1}^{n} |\mathbf{x}_j|^q}$ for $\infty \geq q \geq 1$. When unspecified, $q = 2$ is assumed. Define $x_+ = \max(0, x)$. The standard deviation of numbers assembled in vector $\mathbf{x}$ is written $\mathrm{std}(\mathbf{x})$.

Let $\mathcal{X}$ denote a set and let $|\mathcal{X}|$ denote its cardinality. Let $\mathbf{x} \in \arg\min_{\mathbf{x}} f(\mathbf{x})$ be an element in the set of optimal solutions to the mathematical program $\min_{\mathbf{x}} f(\mathbf{x})$. The convex hull written over a vector set $\mathcal{X}$ is denoted $\mathrm{conv}\{\mathcal{X}\}$.

This paper makes extensive use of subgradients, we briefly review them here. Consider a nonsmooth locally Lipschitz function $f : \mathbb{R}^n \to \mathbb{R}$ that is differentiable almost everywhere. The subdifferential of f at $\bar{\mathbf{x}}$ is a set (Clarke 1990; Ruszczyński 2006):

$$\partial f(\bar{\mathbf{x}}) = \mathrm{conv}\{\mathbf{g} \in \mathbb{R}^n | \mathbf{x} \in \mathbb{R}^n, \nabla f(\mathbf{x}) \to \mathbf{g}, \mathbf{x} \to \bar{\mathbf{x}}, \mathbf{x} \notin \Omega\} \tag{1}$$

with Ω being the set of points where f is not differentiable. A subgradient $\mathbf{g}$ of f at $\mathbf{x}$ is any element of $\partial f(\bar{\mathbf{x}})$.

We assume the subdifferential is polyhedral. This is a mild assumption that would be satisfied for any function whose nonsmooth components result from absolute value, minimum, maximum, or step functions. Detecting the necessary conditions of optimality for

nonsmooth functions is more difficult than for smooth functions. In the nonsmooth case, all directional derivatives must be considered, an expensive task. Finding the minimum norm subgradient in the convex case is another strategy for detecting optimality. If this subgradient is zero, then the solution is optimal, but the search for such a subgradient may also be nontrivial. Finally, in this work, we frequently refer to subgradients as gradients. However, strictly speaking the gradient of a function is unique and nonzero gradients define directions of decrease of the function. Subgradients do not necessarily possess these properties.

3 Bilevel cross-validation problem

A bilevel program is a mathematical program that optimizes an objective subject to constraints which are also mathematical programs. These constraints are referred to as the *inner-level problems* whereas the overall problem is called the *outer-level problem*. Bilevel optimization problems were first studied by Bracken and McGill (1973). A survey by Dempe (2003) provides a general background.

Model selection via T-fold cross-validation can be written compactly as single bilevel program (Bennett et al. 2006; Kunapuli et al. 2008). The inner-level problems minimize the regularized training error to determine the best function for the given hyperparameters for each fold. The hyperparameters are the outer-level control variables. The objective of the outer-level is to minimize the validation error based on the optimal parameters returned for each fold.

Formally, let $\Omega := \{\mathbf{x}_j, y_j\}_{j=1}^{\ell} \in \mathbb{R}^{n+1}$ denote a given labeled data set consisting of ℓ samples, where each sample is described by an n-dimensional feature vector $\mathbf{x}_j$ and a response label y_j. Set Ω is broken into T equally sized divisions to perform T-fold cross-validation. For fold $t = 1, \ldots, T$, one of the divisions is used as the validation set, Ω_{val}^t, and the remaining $T - 1$ divisions are assigned to the training set, Ω_{trn}^t. Let $\gamma \in \Gamma \in \mathbb{R}^m$ be the set of m model hyperparameters and $\mathbf{w}_t$ be the model weights for the tth fold. Let $\mathcal{L}_{trn}^t(\mathbf{w}_t, \boldsymbol{\gamma} | (\mathbf{x}_j, y_j) \in \Omega_{trn}^t)$ be the inner-level training function given the tth fold training dataset and $\mathcal{L}_{val}^t(\mathbf{w}_t, \boldsymbol{\gamma} | (\mathbf{x}_j, y_j) \in \Omega_{val}^t$ be the tth outer-level validation loss function given its validation dataset. Note we omit the bias term b for simplicity, but all results remain valid with its inclusion. The bilevel program for T-fold cross-validation is:

$$\begin{aligned} \min_{\mathbf{w}_1,\ldots,\mathbf{w}_T,\boldsymbol{\gamma}} \; & \frac{1}{T}\sum_{t=1}^{T} \mathcal{L}_{val}^t\left(\mathbf{w}_t, \boldsymbol{\gamma} | (\mathbf{x}_j, y_j) \in \Omega_{val}^t\right) \\ & \qquad \textbf{(outer-level)} \\ \text{subject to } & \boldsymbol{\gamma} \in \Gamma \\ & \text{for } t = 1, \ldots, T: \\ & \mathbf{w}_t \in \arg\min_{\mathbf{w}} \left\{ \mathcal{L}_{trn}^t\left(\mathbf{w}_t, \boldsymbol{\gamma} | (\mathbf{x}_j, y_j) \in \Omega_{trn}^t\right) \right\}. \\ & \qquad \textbf{(inner-level)} \end{aligned} \tag{2}$$

The bilevel CV program is challenging to solve in this form, thus both the implicit and explicit methods reformulate it into a nonlinear programming problem. To facilitate the discussion, we explain the reformations as a generic bilevel programming problem with a single inner-level problem. Let the bilevel program in its entirety be referred to as the *outer-level* problem with objective $\mathcal{L}_{\text{out}}$. Define control variables $\boldsymbol{\gamma}$ (e.g. the hyperparameters) and state variables $\mathbf{w}$ (e.g. the model weights). A constraint that is a mathematical problem is

called an *inner-level* problem $\mathcal{L}_{\text{in}}$ and it optimizes the state variables, given a fixed set of control variables. We formulate the following general bilevel program:

$$\begin{aligned} &\min_{\mathbf{w},\boldsymbol{\gamma}} \mathcal{L}_{\text{out}}(\mathbf{w},\boldsymbol{\gamma}) \\ &\text{s.t. } A\boldsymbol{\gamma} \leq \mathbf{c} \\ &\quad \mathbf{w} \in \arg\min_{\mathbf{w}} \mathcal{L}_{\text{in}}(\mathbf{w},\boldsymbol{\gamma}). \end{aligned} \tag{3}$$

We assume the control variables are constrained to lie in a convex polyhedral set, namely $A\boldsymbol{\gamma} \leq \mathbf{c}$.

Implicit methods make $\mathbf{w}$ an implicit function of γ, namely $\mathbf{w}(\boldsymbol{\gamma})$. This creates a single level optimization problem with $\mathbf{w}$ as an implicit function:

$$\begin{aligned} &\min_{\boldsymbol{\gamma}} \mathcal{L}_{\text{out}}(\mathbf{w}(\boldsymbol{\gamma}),\boldsymbol{\gamma}) \\ &\text{s.t. } A\boldsymbol{\gamma} \leq \mathbf{c}. \end{aligned} \tag{4}$$

Implicit methods replace the difficulty of the constraints with a new more complex objective. In practice nonlinear objectives are much easier to optimize than nonlinear constraints. The function $\mathbf{w}(\boldsymbol{\gamma})$ is computed such that $\mathbf{w} \in \arg\min_{\mathbf{w}} \mathcal{L}_{\text{in}}(\mathbf{w},\boldsymbol{\gamma})$.

There are three main difficulties in implicit methods: the function is nonsmooth and nonconvex, the function evaluation is computationally complex, and determining a descent direction is difficult. The nonsmoothness comes from changes in the set of points with errors (a.k.a. the support vectors) as the hyperparameters are changed. The nonconvexity resulting from the nonlinear function $\mathbf{w}(\boldsymbol{\gamma})$, and means that local minimums and saddle points may be problems. The complexity comes from the fact that an evaluation of the implicit objective $\mathcal{L}_{\text{out}}(\mathbf{w}(\boldsymbol{\gamma}),\boldsymbol{\gamma})$ requires a solution of the inner problems. For SVM problems, the implicit objective is not continuously differentiable, but directional derivatives and subgradients exist. Recall, that subgradients may not be unique and a given subgradient may not be a direction of descent for the implicit function. In this context, the success and appeal of grid search for problems with few hyperparameters is readily apparent. Grid performs a simple somewhat global search without the need of gradient information that is quite tractable for small numbers of hyperparameters.

Implicit gradient methods differ in how the subgradients are calculated. From now on, we will refer to the subgradients as gradients for simplicity. In Sect. 4.1, we review how SVM can be equivalently written as a smooth or nonsmooth minimization problem. The optimality conditions for the smooth and nonsmooth SVM are quite different. The KKT optimality conditions of the traditional SVM involve both primal and dual variables and constraints, so we call this the dual method. The KKT system of equations involves the kernel matrix, so effectively the kernel must be inverted. The optimality condition of the inner problem of nonsmooth SVM is:

$$\mathbf{0} \in \partial_{\mathbf{w}} \mathcal{L}_{\text{in}}(\mathbf{w},\boldsymbol{\gamma}). \tag{5}$$

A novel nonsmooth implicit method based on (5) is a novel contribution of this paper. It represents an entirely new approach to bilevel programs with nonsmooth inner-level problems.

The explicit methods investigated here also work by reformulation of the BCP. The general approach is to replace the inner problems with their optimality conditions. Once again different problems result in different algorithms based on if the inner-level problem is taken to be a smooth or nonsmooth formulation. If the smooth SVM is used, the KKT conditions

include complementarity constraints. These are also known as equilibrium constraints, so the problem becomes a Mathematical Programming with Equilibrium Constraints (MPEC) and this formed the basis of the prior MPEC approach by Bennett et al. (2006) and Kunapuli et al. (2008). The new nonsmooth bilevel approach replaces the inner-level problem with a nonlinear constraint. This approach assumes that the inner-level problem is differentiable.

The reformulated bilevel cross-validation problem becomes:

$$\begin{aligned} &\min_{\mathbf{w},\boldsymbol{\gamma}} \mathcal{L}_{\text{out}}(\mathbf{w},\boldsymbol{\gamma}) \\ &\text{s.t. } A\boldsymbol{\gamma} \leq \mathbf{c} \\ &\quad \mathbf{0} \in \partial_{\mathbf{w}} \mathcal{L}_{\text{in}}(\mathbf{w},\boldsymbol{\gamma}). \end{aligned} \tag{6}$$

The second constraint can be treated as a penalty term in the outer-level objective:

$$\begin{aligned} &\min_{\mathbf{w},\boldsymbol{\gamma}} \mathcal{L}_{\text{out}}(\mathbf{w},\boldsymbol{\gamma}) + \beta \|\partial_{\mathbf{w}} \mathcal{L}_{\text{in}}(\mathbf{w},\boldsymbol{\gamma})\|^2 \\ &\text{s.t. } A\boldsymbol{\gamma} \leq \mathbf{c}. \end{aligned} \tag{7}$$

In doing so, we emulate the nonsmooth primal strategy of Joachims (2006) and Teo et al. (2007) by writing a constrained mathematical program as an unconstrained one. We directly solve this nonsmooth nonconvex problem in (6) using the PBP algorithm presented in Sect. 7.

4 Support vector regression and cross-validation

This section introduces relevant aspects of support vector regression (SVR), bilevel programming (BP), and cross-validation (CV). It then details how these topics are integrated to perform model selection.

The general approach developed in this paper is applicable to general convex loss functions including the nonsmooth ones used in SVM. However, for a simpler exposition, we focus on inner-level regularization functions and training losses that are differentiable but not necessarily twice continuously differentiable. For example least squares SVR (Suykens et al. 2002) and a multi-group least squares SVR is addressed in this paper. This restriction includes many well-known linear machine learning algorithms. Future investigations will be made into handling other losses such as absolute value training errors and 1-norm regularization.

4.1 Support vector regression

We focus on ϵ-insensitive least squares support vector regression (SVR) as an example of an SVM-like learning means. SVR possesses two hyperparameters:

- Tradeoff C determines the relative weight of the structural and empirical risks in the SVM objective function.
- Insensitivity ϵ is the distance from the solution hypersurface within which errors are not assessed. Figure 2 illustrates this.

We formally state SVR. Given a training set Ω_{trn} containing data pairs $\{(\mathbf{x}_j, y_j) \in \mathbb{R}^{n+1}\}_{j\in\Omega_{trn}}$. The goal is to identify a weight vector $\mathbf{w}$ that generalizes well on unseen data

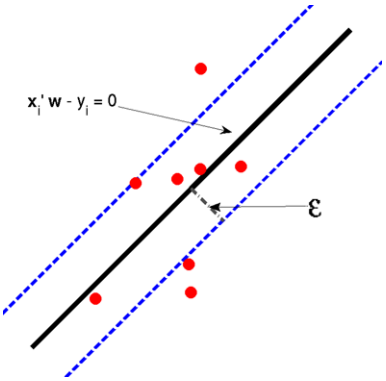

Fig. 2 (Color online) SVR with ϵ-insensitive loss. *Red dots* inside the *blue dashed* ϵ-radius tube surrounding the function $\mathbf{x}_j'\mathbf{w} - y_j = 0$ (*solid black line*) are not penalized. Only points outside the tube are penalized

$(\mathbf{x}, y)$, where y is predicted by $\mathbf{x}'\mathbf{w}$. We omit the bias b to simplify, but all results remain valid with its inclusion. Formulated as a mathematical problem, SVR is:

$$\begin{aligned} &\min_{\mathbf{w},\xi} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{2}\sum_{j\in\Omega_{trn}} \xi_j^2 \\ &\text{s.t. } \mathbf{x}_j'\mathbf{w} - y_j - \epsilon \le \xi_j \ \forall j \in \Omega_{trn} \\ &\quad -\mathbf{x}_j'\mathbf{w} + y_j - \epsilon \le \xi_j \ \forall j \in \Omega_{trn}. \end{aligned} \tag{8}$$

This is a smooth, convex, linearly-constrained quadratic optimization program. The first term involving the norm of $\mathbf{w}$ minimizes structural risk, which encourages simpler models over more complex ones. The second term minimizes errors ξ_j. The error contributed by sample j is the orthogonal distance ξ_j to the model hyperplane (defined by $\mathbf{w}$) in excess of ϵ.

The equivalent unconstrained formulation for SVR has the advantage of eliminating the constraints at the cost of making the objective only once differentiable:

$$\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{2}\sum_{j\in\Omega_{trn}} \left(|\mathbf{x}_j'\mathbf{w} - y_j| - \epsilon\right)_+^2. \tag{9}$$

It is under this convex formulation that algorithms have succeeded in training models in linear-time in the sample size (Joachims 2006; Teo et al. 2007). We chose the squared 2-norm for the structural risk and squared the errors as these choices provide richer gradient information to the objective. The optimality condition of (9) will be useful in later algorithms. It is:

$$\mathbf{w} + C\sum_{j\in\Omega_{trn}} \mathbf{x}_j\left[\left(\mathbf{x}_j'\mathbf{w} - y_j - \epsilon\right)_+ - \left(-\mathbf{x}_j'\mathbf{w} + y_j - \epsilon\right)_+\right] = \mathbf{0}. \tag{10}$$

4.2 Cross-validation for SVR

We state the objective functions $\mathcal{L}_{\text{in}}$ and $\mathcal{L}_{\text{out}}$ for SVR. For both SVR and multiSVR CV, each fold t in T-fold CV contributes a validation mean squared error:

$$\mathcal{L}_{val}^t(\mathbf{w}) = \frac{1}{|\Omega_{val}^t|}\sum_{j\in\Omega_{val}^t} (\mathbf{x}_j'\mathbf{w}_t - y_j)^2. \tag{11}$$

These are averaged across all folds to generate the outer-level objective:

$$\mathcal{L}_{\text{out}} = \frac{1}{T}\sum_{t=1}^{T} \mathcal{L}_{val}^{t}(\mathbf{w}). \tag{12}$$

For SVR, there are T inner-level objectives $\mathcal{L}_{\text{in}}$ which correspond to the unconstrained SVR loss (9) applied to fold t:

$$\mathcal{L}_{trn}^{t}(\mathbf{w}, C, \epsilon) = \frac{1}{2}\|\mathbf{w}_t\| + \frac{C}{2}\sum_{j\in\Omega_{trn}^{t}} (|\mathbf{x}_j'\mathbf{w}_t - y_j| - \epsilon)_+^2. \tag{13}$$

We also impose that C is positive and ϵ is nonnegative. Written out, the full BP is:

$$\begin{aligned} &\min_{\mathbf{w},C,\epsilon} \frac{1}{T}\sum_{t=1}^{T} \frac{1}{|\Omega_{val}^{t}|}\sum_{j\in\Omega_{val}^{t}} (\mathbf{x}_j'\mathbf{w}_t - y_j)^2 \\ &\text{s.t. } C > 0 \\ &\quad \epsilon \geq 0 \\ &\quad \text{and for } t = 1\ldots T\text{:} \\ &\quad \mathbf{w}_t \in \arg\min_{\mathbf{w}_t} \left\{ \frac{1}{2}\|\mathbf{w}_t\|^2 + \frac{C}{2}\sum_{j\in\Omega_{trn}^{t}} (|\mathbf{x}_j'\mathbf{w}_t - y_j| - \epsilon)_+^2 \right\}. \end{aligned} \tag{14}$$

Notice how there are multiple inner-level problems, each producing a constraint in the bilevel program.

4.3 Cross-validation for multi-group SVR

Multiple-group SVR (multiSVR) is a generalization of SVR in which each sample is assigned to a group and each group is permitted a hyperparameter C_g and ϵ_g.

The idea behind multiSVR is that different groups of samples within a dataset undergoing regression modeling should have their own value for C and ϵ. For example, experimental data produced by different laboratory technicians may possess different characteristics, different biases, or different levels of reliability. Some data groupings may be mostly inaccurate but still contribute some slight information that can improve modeling. So the purpose of multiSVR is two-fold: first, to improve the predictive capability of models by working with multiple hyperparameters and second, to identify properties in datasets based on calculated hyperparameter values. For example, we expect that a group with large C signifies good quality data while a group getting a small C is of poorer quality.

Divide the modeling set into G groups for multiSVR. As with SVR, there are T inner-level objectives $\mathcal{L}_{\text{in}}$ which correspond to the unconstrained SVR loss (9) applied to fold t:

$$\mathcal{L}_{trn}^{t} = \frac{1}{2}\|\mathbf{w}_t\|^2 + \frac{1}{2}\sum_{g=1}^{G}\left\{ C_g \sum_{j\in\Omega_{g,trn}^{t}} \left(|\mathbf{x}_j'\mathbf{w}_t - y_j| - \epsilon_g\right)_+^2 \right\} \tag{15}$$

where $\Omega^t_{g,trn}$ is the portion of set Ω^t_{trn} function whose elements belong to group g. The full BP for CV with multiSVR is BP (14) with

$$\mathbf{w}_t \in \arg\min_{\mathbf{w}_t} \left\{ \frac{1}{2}\|\mathbf{w}_t\|^2 + \frac{1}{2}\sum_{g=1}^{G} \left\{ C_g \sum_{j\in\Omega^t_{g,trn}} \left(|\mathbf{x}'_j\mathbf{w}_t - y_j| - \epsilon_g\right)_+^2 \right\} \right\} \tag{16}$$

substituted as the last constraint.

With knowledge of SVR, we will now present ImpGrad and PBP methods in general, and then apply them to SVR.

5 Implicit model selection methods

In this section, we focus on implicit methods for model selection. Implicit gradient descent methods (Chapelle et al. 2002; Do et al. 2008; Keerthi et al. 2006) optimize the validation objective, treating the weight vector $\mathbf{w}$ as a function of $\boldsymbol{\gamma}$. The gradient of the validation function is used to optimize the validation objective. We now show how this general implicit method can be extended to nonsmooth SVM.

The general implicit gradient descent method is presented as Algorithm 1. We continue to use the notation $\mathcal{L}_{\text{out}}$ and $\mathcal{L}_{\text{in}}$ here for simplicity of deriving the algorithm, then give an example of T-fold CV for SVR. The idea of the method is to iteratively refine the hyperparameters. It evaluates the implicit function and computes the gradient of the outer-level function with respect to $\boldsymbol{\gamma}$. To do this requires an inner-level problem to be trained given the fixed $\boldsymbol{\gamma}$. Utilizing this gradient information, a new candidate $\boldsymbol{\gamma}$ in the gradient direction is chosen. If the new candidate $\boldsymbol{\gamma}$ is not optimal, this process is repeated. The difficulties lie in computing the gradient of the outer-level function with respect to $\boldsymbol{\gamma}$, as it effectively requires an inversion of a matrix, and then determining the step-size since each function evaluation requires solving the inner-level problem.

The chain rule is used to construct the gradient of $\mathcal{L}_{\text{out}}(\mathbf{w}(\boldsymbol{\gamma}), \boldsymbol{\gamma})$ with respect to $\boldsymbol{\gamma}$. Note that the gradient (really a subgradient) is typically not unique unless the function is smooth. The gradient of the outer-level function can be computed at the point $\boldsymbol{\gamma}^i$ as

$$\frac{\partial}{\partial \boldsymbol{\gamma}}\left(\mathcal{L}_{\text{out}}(\mathbf{w}(\boldsymbol{\gamma}^i), \boldsymbol{\gamma}^i)\right) = \left(\frac{\partial}{\partial \mathbf{w}}\mathcal{L}_{\text{out}}(\mathbf{w}(\boldsymbol{\gamma}^i), \boldsymbol{\gamma}^i)\right)' \frac{\partial \mathbf{w}(\boldsymbol{\gamma}^i)}{\partial \boldsymbol{\gamma}} + \frac{\partial}{\partial \boldsymbol{\gamma}}\mathcal{L}_{\text{out}}(\mathbf{w}^i, \boldsymbol{\gamma}^i), \tag{17}$$

Algorithm 1 General implicit gradient descent algorithm

Choose an appropriate starting hyperparameter combination $\boldsymbol{\gamma}^1$, and exit criterion tol_1 and tol_2.
Solve the inner-level problem given the hyperparameters $\boldsymbol{\gamma}^1$. Let the solution be $\mathbf{w}^1$
Compute the function value and gradient of $\mathcal{L}_{\text{out}}$ with respect to $\boldsymbol{\gamma}$ at the point $(\boldsymbol{\gamma}^1, \mathbf{w}^1)$.
repeat
  Using the gradient information, search for a better hyperparameter combination $\boldsymbol{\gamma}^{i+1}$.
  Solve the inner-level problem given the hyperparameters $\boldsymbol{\gamma}^{i+1}$. Let the solution be $\mathbf{w}^{i+1}$
  Compute the gradient of $\mathcal{L}_{\text{out}}$ with respect to $\boldsymbol{\gamma}$ at the point $(\boldsymbol{\gamma}^{i+1}, \mathbf{w}^{i+1})$.
  Let $i = i + 1$.
until $|\mathcal{L}_{\text{out}}(\mathbf{w}(\boldsymbol{\gamma}^i), \boldsymbol{\gamma}^i) - \mathcal{L}_{\text{out}}(\mathbf{w}(\boldsymbol{\gamma}^{i-1}), \boldsymbol{\gamma}^{i-1})| < tol_1|\mathcal{L}_{\text{out}}(\mathbf{w}(\boldsymbol{\gamma}^i), \boldsymbol{\gamma}^i) + 1|$ or $\|\frac{\partial}{\partial \boldsymbol{\gamma}}\mathcal{L}_{\text{out}}(\mathbf{w}(\boldsymbol{\gamma}^i), \boldsymbol{\gamma}^i)\| < tol_2|\mathcal{L}_{\text{out}}(\mathbf{w}(\boldsymbol{\gamma}^{i+1}), \boldsymbol{\gamma}^{i+1}) + 1|$
Return: $\boldsymbol{\gamma}^i, \mathbf{w}^i$

where $\mathbf{w}^i$ is the optimal solution of the inner-level problem given $\boldsymbol{\gamma}^i$. For simplicity we assume that the $\boldsymbol{\gamma}$ does not directly impact $\mathcal{L}_{\text{out}}$, and thus the addition of the latter term is unnecessary; All results hold with its inclusion. It is typically straight forward to compute $\partial_{\mathbf{w}}\mathcal{L}_{\text{out}}(\mathbf{w}(\boldsymbol{\gamma}), \boldsymbol{\gamma})$, and therefore the difficulty lies in the computation of $\partial_{\boldsymbol{\gamma}}\mathbf{w}(\boldsymbol{\gamma})$.

The implicit nature of $\mathbf{w}(\boldsymbol{\gamma})$ is characterized by the solving the inner-level problem, that is selecting $\mathbf{w}$ given fixed $\boldsymbol{\gamma}$. Thus

$$\frac{\partial \mathbf{w}(\boldsymbol{\gamma})}{\partial \boldsymbol{\gamma}} \in \frac{\partial}{\partial \boldsymbol{\gamma}} \arg\min_{\mathbf{w}} \mathcal{L}_{\text{in}}(\mathbf{w}, \boldsymbol{\gamma}). \tag{18}$$

This is the location of a fundamental challenge with this algorithm. We assume that $\mathcal{L}_{\text{in}}$ is a differentiable function of $\mathbf{w}$. We do not assume, and regularly found false in practice, that the second derivative $\frac{\partial^2}{\partial \boldsymbol{\gamma} \partial \mathbf{w}} \mathcal{L}_{\text{in}}(\mathbf{w}(\boldsymbol{\gamma}), \boldsymbol{\gamma})$ is smooth. This second derivative, which will be fully developed in a moment, is why the right hand side of (18) is a subgradient set, and not a unique gradient. At these points $\mathcal{L}_{\text{out}}$ is *not* twice differentiable. Following the lead of Keerthi et al. (2006) who assumes the subgradient is unique, we construct one subgradient to act as the gradient in these cases.

The first step of the implicit gradient algorithm is to solve the inner-level problem given a fixed $\boldsymbol{\gamma}^i$. This is done using any appropriate linear-time SVM algorithm and returns weight vector $\mathbf{w}^i$. The next step is to calculate $\frac{\partial}{\partial \mathbf{w}} \mathcal{L}_{\text{out}}(\mathbf{w}(\boldsymbol{\gamma}), \boldsymbol{\gamma})$, which is usually quite trivial, which brings us to $\frac{\partial \mathbf{w}(\boldsymbol{\gamma})}{\partial \boldsymbol{\gamma}}$.

Given the current point $\boldsymbol{\gamma}^i$, the optimality condition of the unconstrained inner-level problem, $\frac{\partial}{\partial \mathbf{w}} \mathcal{L}_{\text{in}}(\mathbf{w}, \boldsymbol{\gamma}^i) = 0$, is written. For SVR and other similar least squares training functions, the optimality condition is a linear system[1] and we will refer to it generically as

$$H\mathbf{w} = \mathbf{h}. \tag{19}$$

Where H and $\mathbf{h}$ are both functions of $\boldsymbol{\gamma}$. To compute $\frac{\partial \mathbf{w}(\boldsymbol{\gamma})}{\partial \boldsymbol{\gamma}_k}$ we take the partial w.r.t. $\boldsymbol{\gamma}_k$ on both sides of system (19) giving:

$$\left(\frac{\partial H}{\partial \boldsymbol{\gamma}_k}\right)\mathbf{w}^i + H(\boldsymbol{\gamma}^i)\frac{\partial \mathbf{w}(\boldsymbol{\gamma})}{\partial \boldsymbol{\gamma}_k} = \frac{\partial \mathbf{h}}{\partial \boldsymbol{\gamma}_k}. \tag{20}$$

If there is ambiguity in the choice of $\partial_{\boldsymbol{\gamma}} H$ or $\partial_{\boldsymbol{\gamma}} \mathbf{h}$ because of nonsmoothness, then a specific subgradient is chosen to act as a gradient. Finally we have

$$\frac{\partial \mathbf{w}}{\partial \boldsymbol{\gamma}_k} = H^{-1}\left(\frac{\partial \mathbf{h}}{\partial \boldsymbol{\gamma}_k} - \left(\frac{\partial H}{\partial \boldsymbol{\gamma}_k}\right)\mathbf{w}^i\right). \tag{21}$$

This step requires an expensive inversion of a matrix, which even if solved as a linear system can be quite costly for large H. In the context of SVMs, the training in the dual space gives H in the number of data points. In the primal space this matrix's size is in the size of the feature space. The former grows quadratically in the number of samples in the modeling set, the latter is fixed in the number of samples but grows in the feature space. We proceed choosing the primal problem due to its scalability in the dataset size. We refer to this now as the implicit primal gradient descent (ImpGrad) algorithm.

[1] Any one-norm or two-norm squared type SVM loss functions will yield linear optimality conditions of the form of (19). But care must be taken because the choice of H, $\mathbf{h}$, and $\mathbf{w}$ may not be unique for a given γ.

As in Keerthi et al. (2006), a standard Broyden-Fletcher-Goldfarb-Shanno (BFGS) quasi-Newton algorithm (Bazaraa et al. 2006) with the above subgradient is used to optimize the hyperparameters. The algorithm halts when the following convergence criterion is satisfied:

* $|\mathcal{L}_{\text{out}}(\mathbf{w}(\boldsymbol{\gamma}^{i+1}), \boldsymbol{\gamma}^{i+1}) - \mathcal{L}_{\text{out}}(\mathbf{w}(\boldsymbol{\gamma}^{i}), \boldsymbol{\gamma}^{i})| < tol_1 |\mathcal{L}_{\text{out}}(\mathbf{w}(\boldsymbol{\gamma}^{i}), \boldsymbol{\gamma}^{i}) + 1|$
* $\|\frac{\partial}{\partial \boldsymbol{\gamma}}(\mathcal{L}_{\text{out}}(\mathbf{w}(\boldsymbol{\gamma}^{i+1}), \boldsymbol{\gamma}^{i+1}))\| \leq tol_2 |\mathcal{L}_{\text{out}}(\mathbf{w}(\boldsymbol{\gamma}^{i+1}), \boldsymbol{\gamma}^{i+1}) + 1|$.

While, this method is *not* guaranteed to converge to a locally optimal solution, it can work quite well in practice for large problems for both dual SVM and primal SVM. For the case when the loss function is not twice differentiable, the subgradient selected may not yield a good direction and the algorithm fails to make significant progress. Such examples of failure are seen in the results for problems where the dataset size is approximately the same as the number of features. We look to develop a more theoretically sound explicit algorithm, PBP, in Sect. 7, but first we give an example of using ImpGrad to solve the CV problem for SVR.

5.1 Using ImpGrad to solve CV for SVR

Here we present the specific example of using T-fold CV and the ImpGrad algorithm to solve SVR and highlight the issue of selecting the gradient. We will see that the subgradient is not unique. The validation function is $\mathcal{L}_{val}^t(\mathbf{w}_t) = \frac{1}{2}\sum_{j\in\Omega_{val}}(\mathbf{x}_j'\mathbf{w}_t - y_j)^2$. This gives

$$\frac{\partial}{\partial(C,\epsilon)}\mathcal{L}_{val}(\mathbf{w}) = \sum_{t=1}^{T}\left(\frac{\partial}{\partial \mathbf{w}_t}\mathcal{L}_{val}^t(\mathbf{w}_t)\right)' \frac{\partial \mathbf{w}_t}{\partial(C,\epsilon)} = \sum_{t=1}^{T}\left(\sum_{j\in\Omega_{trn}^t}\mathbf{x}_j(\mathbf{x}_j'\mathbf{w}_t - y_j)\right)' \frac{\partial \mathbf{w}_t}{\partial(C,\epsilon)}. \tag{22}$$

To calculate $\frac{\partial \mathbf{w}_t}{\partial(C,\epsilon)}$, first the primal SVM problem is solved for fixed (C^i,ϵ^i) utilizing a subgradient method. Then its solution $\mathbf{w}^i$ is used to calculate

$$\frac{\partial \mathbf{w}_t}{\partial(C,\epsilon)} = \frac{\partial}{\partial(C,\epsilon)}\arg\min_{\mathbf{w}_t}\frac{1}{2}\|\mathbf{w}_t\|_2^2 + \frac{C}{2}\sum_{j\in\Omega_{trn}^t}(|\mathbf{x}_j'\mathbf{w}_t - y_j| - \epsilon)_+^2. \tag{23}$$

The optimality condition for this training problem can be written as:

$$\mathbf{0} = \mathbf{w}_t + C\sum_{j\in\Omega_{trn}^t}\left[\mathbf{x}_j(\mathbf{x}_j'\mathbf{w}_t - y_j - \epsilon)_+ - \mathbf{x}_j(-\mathbf{x}_j'\mathbf{w}_t + y_j - \epsilon)_+\right] \tag{24}$$

and taking the partial derivative with respect to C and ϵ gives

$$\begin{aligned}\mathbf{0} = {} & \frac{\partial \mathbf{w}_t}{\partial C} + C^i\sum_{j\in\Omega_{trn}^t}\left[\delta_j^+\mathbf{x}_j\mathbf{x}_j' + \delta_j^-\mathbf{x}_j\mathbf{x}_j'\right]\frac{\partial \mathbf{w}_t}{\partial C} \\ & + \sum_{j\in\Omega_{trn}^t}\left[\mathbf{x}_j(\mathbf{x}_j'\mathbf{w}_t{}^i - y_j - \epsilon^i)_+ - \mathbf{x}_j(-\mathbf{x}_j'\mathbf{w}_t{}^i + y_j - \epsilon^i)_+\right]\end{aligned} \tag{25}$$

and

$$\mathbf{0} = \frac{\partial \mathbf{w}_t}{\partial \epsilon} + C^i\sum_{j\in\Omega_{trn}}\left[-\delta_j^+\mathbf{x}_j + \delta_j^-\mathbf{x}_j\right] + C^i\sum_{j\in\Omega_{trn}^t}\left[\delta_j^+\mathbf{x}_j\mathbf{x}_j' + \delta_j^+\mathbf{x}_j\mathbf{x}_j'\right]\partial_\epsilon\mathbf{w}, \tag{26}$$

where $\delta_j^+ = (\text{sign}(\mathbf{x}_j' \mathbf{w}_t{}^i - y_j - \epsilon^i))_+$, and likewise $\delta_j^- = (\text{sign}(-\mathbf{x}_j' \mathbf{w}_t{}^i + y_j - \epsilon^i))_+$ (recall $(\text{sign}(0))_+ \in [0, 1]$, and not necessarily unique). Written more suggestively these two equations form the linear system:

$$\begin{bmatrix} I + C^i \sum_{j \in \Omega_{trn}} (\delta_j^+ + \delta_j^-) \mathbf{x}_j \mathbf{x}_j' & \mathbf{0} \\ \mathbf{0} & I + C^i \sum_{j \in \Omega_{trn}} (\delta_j^+ + \delta_j^-) \mathbf{x}_j \mathbf{x}_j' \end{bmatrix} \begin{bmatrix} \partial_C \mathbf{w} \\ \partial_\epsilon \mathbf{w} \end{bmatrix}$$
$$= \begin{bmatrix} -\sum_{j \in \Omega_{trn}} \mathbf{x}_j \left[(\mathbf{x}_j' \mathbf{w}_t - y_j - \epsilon^i)_+ - (-\mathbf{x}_j' \mathbf{w}_t + y_j - \epsilon^i)_+ \right] \\ C^i \sum_{j \in \Omega_{trn}} \left[-\delta_j^+ \mathbf{x}_j + \delta_j^- \mathbf{x}_j \right] \end{bmatrix}. \tag{27}$$

This linear system would give a unique $\partial_{C,\epsilon} \mathbf{w}$ directly if it were not for the $\delta_j^{+/-}$ functions being non-unique for points which lie on the ϵ tube. Here $\delta_j^{+/-}$ can take on *any* value in $[0, 1]$ if $|\mathbf{x}_j' \mathbf{w}_t^i + y_j| - \epsilon^i = 0$. We follow common practice in SVM subgradients, and select the subgradient $\delta_j^{+/-} = 0$ for points on the ϵ tube. A similar assumption is made in the dual form of the problem. It assumes there is no change in the support vectors about the current set of hyperparameters. The case where the dual form is not differentiable is a result of these same points who lie on the tube.

With this gradient, a standard BFGS algorithm can search for improving hyperparameters, and thus produces a good model. But if the gradient choice is "unlucky" the algorithm may halt at a solution that is not locally optimal. An "unlucky" choice stems from selecting a subgradient that does not determine a descent direction. If no decrease is found, the BFGS algorithm will halt even if a direction of descent exists at the current point. There is no guarantee that the subgradient used yields a direction of descent. ImpGrad could be improved by searching for a subgradient that corresponds to direction of descent. This is not a straightforward task so is left for future work. In the next section we show how the search for the subgradient corresponding to greatest feasible descent direction creates a more a robust and theoretically well founded algorithm for the explicit algorithm. This search of the subdifferential also corresponds to a check of the necessary optimality condition used for termination of the algorithm. In general termination of subgradient algorithm can be challenging. Use of more robust nonsmooth optimization such as bundle methods may also improve ImpGrad (Pang and Sun 2008).

ImpGrad alternates between training a model and updating the hyperparameters. Ideally an explicit algorithm that simultaneously solves for both model weights and hyperparameters would be more efficient as there is no need to train a model to optimality when far from the optimal solution.

6 Explicit model selection methods

In this section, we focus on explicit methods for model selection. We again revert back to using the $\mathcal{L}_{\text{out}}(\mathbf{w}, \boldsymbol{\gamma})$ and $\mathcal{L}_{\text{in}}(\mathbf{w}, \boldsymbol{\gamma})$ notation for simplicity. We follow the algorithm with an example of multiSVR to highlight the finer points of implementing the algorithm.

We assume that the inner-level objective functions are differentiable and convex with respect to $\mathbf{w}$, thus the optimality condition is the partial derivative of $\mathcal{L}_{\text{in}}(\mathbf{w}, \boldsymbol{\gamma})$ with respect

to $\mathbf{w}$ is equal to zero:

$$\partial_{\mathbf{w}} \mathcal{L}_{\text{in}}(\mathbf{w}, \boldsymbol{\gamma}) = \mathbf{0}. \tag{28}$$

We substitute this for the inner-level objective in BP (3) to get BP (6). These constrain the outer-level problem. They are nonconvex and nonsmooth, but their number remains in the size of $\mathbf{w}$ and no new variables are introduced. This hopefully permits better scalability than MPEC bilevel attempts (Bennett et al. 2006; Kunapuli et al. 2008).

A penalty approach is used to bring these constraints into the outer-level objective. The squared 2-norm of the gradient vector is assessed as the penalty:

$$\|\partial_{\mathbf{w}} \mathcal{L}_{\text{in}}(\mathbf{w}, \boldsymbol{\gamma})\|_2^2. \tag{29}$$

Penalty parameter β trades off the original $\mathcal{L}_{\text{out}}$ with this penalty (29). As $\beta \to \infty$, any violation becomes arbitrarily large; minimizing

$$\mathcal{L}_{\text{out}}(\mathbf{w}) + \beta \|\partial_{\mathbf{w}} \mathcal{L}_{\text{in}}(\mathbf{w}, \boldsymbol{\gamma})\|_2^2 \tag{30}$$

ensures that

$$\partial_{\mathbf{w}} \mathcal{L}_{\text{in}}(\mathbf{w}, \boldsymbol{\gamma}) \to 0. \tag{31}$$

We summarize the transformation from a bilevel program to the nonconvex nonsmooth program:

$$\text{Original problem:} \quad \left\{ \begin{array}{l} \min_{\mathbf{w},\boldsymbol{\gamma}} \mathcal{L}_{\text{out}}(\mathbf{w}, \boldsymbol{\gamma}) \\ \text{s.t. } A\boldsymbol{\gamma} \preceq \mathbf{c} \\ \quad \mathbf{w} \in \arg\min_{\mathbf{w}} \mathcal{L}_{\text{in}}(\mathbf{w}, \boldsymbol{\gamma}) \end{array} \right\} \tag{32}$$

$$\text{Gradient equals zero:} \Rightarrow \left\{ \begin{array}{l} \min_{\mathbf{w},\boldsymbol{\gamma}} \mathcal{L}_{\text{out}}(\mathbf{w}, \boldsymbol{\gamma}) \\ \text{s.t. } A\boldsymbol{\gamma} \preceq \mathbf{c} \\ \quad 0 = \partial_{\mathbf{w}} \mathcal{L}_{\text{in}}(\mathbf{w}, \boldsymbol{\gamma}) \end{array} \right\} \tag{33}$$

$$\text{Penalized bilevel program (PBP):} \Rightarrow \left\{ \begin{array}{l} \min_{\mathbf{w},\boldsymbol{\gamma}} \mathcal{L}_{\text{out}}(\mathbf{w}, \boldsymbol{\gamma}) + \beta \|\partial_{\mathbf{w}} \mathcal{L}_{\text{in}}(\mathbf{w}, \boldsymbol{\gamma})\|_2^2 \\ \text{s.t. } A\boldsymbol{\gamma} \preceq \mathbf{c} \end{array} \right\}. \tag{34}$$

This nonsmooth approach contrasts with the smooth MPEC formulation, under which the inner-level problems have objectives and constraints that are convex, linear and smooth, such as in the constrained SVR problem (8). Lagrange multipliers are introduced for each problem and the KKT optimality conditions (first-order Lagrangian condition, primal and dual feasibility, and complementarity slackness conditions) are written down replacing the inner-level optimization programs. Performing T-fold CV with a training set of size $|\Omega_{trn}|$ results in $6T|\Omega_{trn}|$ linear constraints and $3T|\Omega_{trn}|$ equilibrium constraints. This nonconvex problem (because of the equilibrium constraints) is solved using an exact penalty method. The computational effort required to solve this MPEC problem blows up with increasing sample size.

This paper develops an algorithm to solve Problem (34). This is the topic of the next section.

7 PBP approximation algorithm

Solving the PBP requires a specialized algorithm that can optimize a nonconvex, nonsmooth, penalized problem with polyhedral constraints. Our algorithm uses a standard penalty algorithm first, fixing β, then optimizes the resulting problem. The following two subsections describe these algorithms.

7.1 Penalty algorithm

First the penalty problem is addressed independently. The solution to Problem (34) tends to the same solution of Problem (32) as $\beta \to \infty$. To ensure numerical stability, β is initially selected small and then progressively increased over time. This algorithm solves Problem (34) for a fixed β, increases β and then resolves utilizing new the fixed β until desired feasibility is reached. Algorithm 2 details the penalty algorithm (Bazaraa et al. 2006).

We now develop an algorithm to solve this resulting Problem (34) with a fixed β.

7.2 Approximation algorithm

We develop a new algorithm to minimize nonconvex, nonsmooth functions with polyhedral constraints. Optimization methods for solving nonconvex, nonsmooth functions exist (Noll et al. 2008), but this problem requires the addition of polyhedral constraints. The algorithm works by minimizing a series of convex approximations of the true function. We define f to be the objective function of (34) with a fixed β and will use a locally accurate approximation function $\hat{f}(\mathbf{w}, \boldsymbol{\gamma}|\mathbf{w}^i \boldsymbol{\gamma}^i)$, about $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$, to minimize (34). A graphical representation of the relationship between f and $\hat{f}$ can be seen in Fig. 3. The approximation $\hat{f}$ must be sufficiently good as characterized by the following definition.

Definition 1 A continuous locally accurate approximation function $\hat{f}(\mathbf{w}, \boldsymbol{\gamma}|\mathbf{w}^i, \boldsymbol{\gamma}^i)$ of f about a stability center $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$ must satisfy the following assumptions:

(D1) $\hat{f}(\mathbf{w}^i, \boldsymbol{\gamma}^i|\mathbf{w}^i, \boldsymbol{\gamma}^i) = f(\mathbf{w}^i, \boldsymbol{\gamma}^i)$, that is the function values of f and $\hat{f}$ must agree at $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$,

Algorithm 2 PBP penalty algorithm for increasing β

Choose an appropriate starting penalty parameter β and exit criterion ζ.
repeat
  Solve Problem (34) with β fixed using Algorithm 3.
  Set $\beta = 2\beta$.
until $\|\partial_{\mathbf{w}} \mathcal{L}_{\text{in}}(\mathbf{w}, \boldsymbol{\gamma})\|_q^q \leq \zeta$
Return: $\mathbf{w}, \boldsymbol{\gamma}$

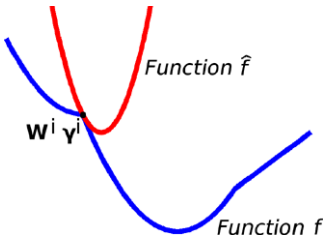

Fig. 3 Approximation function. *The upper red* line $\hat{f}$ approximates the *lower blue function* f (see (30)) at the point $(\mathbf{w}^i, \gamma^i)$. At $(\mathbf{w}^i, \gamma^i)$ both function must be equal, and the gradient of $\hat{f}$ must give the greatest feasible descent with respect to f

Algorithm 3 Proximity control approximation algorithm

Choose feasible $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$, starting τ, $0 < \rho < 1$ and exit tolerances tol and η.
repeat
  Solve (39) for $\mathbf{g}, \boldsymbol{\lambda}$.
  repeat
    Build approximation function $\hat{f}$ as in (35) about $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$.
    Solve Subproblem (37) for direction $\mathbf{d}$.
    Expand proximity control parameter $\tau = 2\tau$.
  until $f([\mathbf{w}^i, \boldsymbol{\gamma}^i] + \mathbf{d}) - f(\mathbf{w}^i, \boldsymbol{\gamma}^i) < \rho(\hat{f}([\mathbf{w}^i, \boldsymbol{\gamma}^i] + \mathbf{d}) - f(\mathbf{w}^i, \boldsymbol{\gamma}^i))$
  Update stability center $(\mathbf{w}^{i+1}, \boldsymbol{\gamma}^{i+1}) = (\mathbf{w}^i, \boldsymbol{\gamma}^i) + \mathbf{d}$, $i = i + 1$ $\tau = \frac{\tau}{\sqrt{2}}$.
until $\|\mathbf{g} + [0 \quad A_I]'\boldsymbol{\lambda}\|^2 < tol$ or $\|\mathbf{d}\| < \eta$.

(D2) $\partial \hat{f}(\mathbf{w}^i, \boldsymbol{\gamma}^i | \mathbf{w}^i, \boldsymbol{\gamma}^i) \in \partial f(\mathbf{w}^i, \boldsymbol{\gamma}^i)$, which requires the differential of $\hat{f}$ to be in differential of f at $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$,

(D3) There exists $\mathbf{g} \in \partial \hat{f}(\mathbf{w}^i, \boldsymbol{\gamma}^i | \mathbf{w}^i, \boldsymbol{\gamma}^i)$ such that $\mathbf{g}$ is the subgradient that yields direction of greatest feasible descent of f at $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$, and

(D4) For every $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$ satisfying $A\boldsymbol{\gamma}^i \leq \mathbf{c}$ and $\epsilon > 0$, there exists a $\delta > 0$ such that $f(\mathbf{w}, \boldsymbol{\gamma}) - \hat{f}(\mathbf{w}, \boldsymbol{\gamma} | \mathbf{w}^i, \boldsymbol{\gamma}^i) \leq \epsilon \|[\mathbf{w}, \boldsymbol{\gamma}] - [\mathbf{w}^i, \boldsymbol{\gamma}^i]\|$ for every $(\mathbf{w}, \boldsymbol{\gamma})$ satisfying $A\boldsymbol{\gamma} \leq \mathbf{c}$ with $\|[\mathbf{w}, \boldsymbol{\gamma}] - [\mathbf{w}^i, \boldsymbol{\gamma}^i]\| \leq \delta$.

f is a continuous Clarke subdifferentiable function that is differentiable almost everywhere. For the PBP function (34) and in the context of SVM losses, these nonsmooth points typically come from the max function. Particularly the derivative of the max function is embedded into $\partial_{\mathbf{w}} \mathcal{L}_{\text{in}}(\mathbf{w}, \boldsymbol{\gamma})$, and is differentiable almost everywhere except when there are multiple maximums.

Algorithm 3 creates a sequence of stability centers corresponding to strict decreases in the objective function. Candidate stability centers are created by minimizing $\hat{f}$ plus a proximity term, subject to any constraints ($A\boldsymbol{\gamma} \leq \mathbf{c}$) on the hyperparameters. Proximity control parameters are increased until a decrease in the actual function is sufficiently large relative to the decrease expected by $\hat{f}$.

The objective of PBP (34), is decomposed into a sum of convex functions and a sum of squared normed nonconvex nonsmooth subdifferentiable functions. At iteration i, the algorithm uses a convex approximation of f at the current iterate, $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$, based on linearizing the inner nonconvex functions $\beta \|\partial_{\mathbf{w}} \mathcal{L}_{\text{in}}(\mathbf{w}, \boldsymbol{\gamma})\|_2^2$ using their subgradients:

$$\begin{aligned} &\hat{f}(\mathbf{w}, \boldsymbol{\gamma} | \mathbf{w}^i, \boldsymbol{\gamma}^i) \\ &\quad := \mathcal{L}_{\text{out}}(\mathbf{w}, \boldsymbol{\gamma}) \\ &\qquad + \frac{\beta}{2} \|\mathcal{L}_{\text{in}}(\mathbf{w}^i, \boldsymbol{\gamma}^i) + \partial_{\mathbf{w}} \mathcal{L}_{\text{in}}(\mathbf{w}^i, \boldsymbol{\gamma}^i)'(\mathbf{w} - \mathbf{w}^i) + \partial_{\boldsymbol{\gamma}} \mathcal{L}_{\text{in}}(\mathbf{w}^i, \boldsymbol{\gamma}^i)'(\boldsymbol{\gamma} - \boldsymbol{\gamma}^i)\|^2. \end{aligned} \tag{35}$$

When $\partial_{\mathbf{w}} \mathcal{L}_{\text{in}}(\mathbf{w}, \boldsymbol{\gamma})$ is nonsmooth, then the subgradient is not unique. Furthermore it is possible that a subgradient yields a direction of assent rather than descent. Therefore the best and most robust subgradient would be the one that leads to the steepest feasible descent direction. The selection of this subgradient is discussed in detail in the next subsection.

The main convex subproblem, with the added proximity control parameter $\tau > 0$, has the form

$$\begin{aligned} &\min_{\mathbf{w},\boldsymbol{\gamma}} \hat{f}(\mathbf{w},\boldsymbol{\gamma}|\mathbf{w}^i,\boldsymbol{\gamma}^i) + \frac{\tau}{2}\|\mathbf{w}-\mathbf{w}^i\|^2 + \frac{\tau}{2}||\boldsymbol{\gamma}-\boldsymbol{\gamma}^i\|^2 \\ &\text{s.t. } A\boldsymbol{\gamma} \leq \mathbf{c}, \end{aligned} \tag{36}$$

or equivalently after the change of variables $\mathbf{d} = [\mathbf{w}-\mathbf{w}^i, \boldsymbol{\gamma}-\boldsymbol{\gamma}^i]$:

$$\begin{aligned} &\min_{\mathbf{d}} \hat{f}([\mathbf{w}^i,\boldsymbol{\gamma}^i] + \mathbf{d}|\mathbf{w}^i,\boldsymbol{\gamma}^i) + \frac{\tau}{2}\|\mathbf{d}\|^2. \\ &\text{s.t. } [0 \;\; A]\mathbf{d} \leq c - A\boldsymbol{\gamma} \end{aligned} \tag{37}$$

The approximation $\hat{f}$ is only locally valid and thus proximity control is required to trade off approximation accuracy versus step size. For a major iterate and associated approximation function, the algorithm loops on increasing values of the proximity control parameter τ. Increasing values of τ decreases $\|\mathbf{d}\|$ and thus the approximation function will better approximate the true function. When a decrease in the original function is found and the approximation is reasonably accurate then the center is updated. Formally this is represented by a parameter $\rho \in (0,1)$ which is the minimum ratio between the approximations accuracy and the seen decrease. Mathematically if

$$f(\mathbf{w}^{i+1},\boldsymbol{\gamma}^{i+1}) - f(\mathbf{w}^i,\boldsymbol{\gamma}^i) < \rho(\hat{f}(\mathbf{w}^{i+1},\boldsymbol{\gamma}^{i+1}) - f(\mathbf{w}^i,\boldsymbol{\gamma}^i)) \tag{38}$$

is satisfied then $(\mathbf{w}^i,\boldsymbol{\gamma}^i)$ and approximation function are updated and τ is reduced.

7.2.1 Optimality and subgradient selection

Two issues of the algorithm remain: selecting the best subgradient for the approximation $\hat{f}$ and detecting necessary conditions for optimality. We prove that these both reduce to solving the same optimality problem for a fixed $(\mathbf{w}^i,\boldsymbol{\gamma}^i)$:

$$\begin{aligned} &\min_{\mathbf{g},\boldsymbol{\lambda}} \|\mathbf{g} + [0 \;\; A_I]'\boldsymbol{\lambda}\|^2 \\ &\text{s.t. } \boldsymbol{\lambda} \geq 0 \\ &\qquad \mathbf{g} \in \partial f(\mathbf{w}^i,\boldsymbol{\gamma}^i) \end{aligned} \tag{39}$$

where $I = \{i | A_i\boldsymbol{\gamma} = c_i\}$ is the index set of active constraints and A_I is the corresponding submatrix.

Theorem 1 (Necessary Optimality Condition of PBP) *Let* $\mathbf{w}^\star,\boldsymbol{\gamma}^\star$ *be a local minimum of problem* (34) *with fixed* β *and let* $\mathbf{g}^\star,\boldsymbol{\lambda}^\star$ *solve problem* (39)*, then* $\mathbf{g}^\star + [0 \;\; A_I]'\boldsymbol{\lambda}^\star = 0$.

Proof Specializing a corollary (Clarke 1990, p. 52) to the normal cone of polyhedral constraints yields the necessary optimality condition: $0 \in \{\partial f(\mathbf{w}^\star,\boldsymbol{\gamma}^\star) + A_I'\boldsymbol{\lambda} | \boldsymbol{\lambda} \geq 0\}$. Thus the lower bound of problem (39) is feasible and thus optimal. □

Furthermore this same problem yields the direction of maximum decrease.

Theorem 2 (Greatest Feasible Descent Direction) *Let* $\mathbf{g}^\star,\boldsymbol{\lambda}^\star$ *be a non-degenerate solution of problem* (39)*. If the current point is not optimal, that is* $\mathbf{g}^\star + [\mathbf{0} \; A_I]'\boldsymbol{\lambda}^\star \neq 0$*, then* $\mathbf{g}^\star$ *and*

$$\mathbf{d}^\star = \frac{\mathbf{g}^\star + [\mathbf{0} \; A_I]'\boldsymbol{\lambda}^\star}{\|\mathbf{g}^\star + [\mathbf{0} \; A_I]'\boldsymbol{\lambda}^\star\|} \tag{40}$$

solves the optimal feasible descent direction problem (Clarke 1990, p. 27):

$$\begin{aligned} &\min_{\mathbf{d}}\{\max_{\mathbf{g}\in\partial f}\mathbf{g}'\mathbf{d}\} \\ &\text{s.t. } [0\ \ A_I]\mathbf{d}\le 0 \\ &\quad \|\mathbf{d}\|=1. \end{aligned} \tag{41}$$

The algorithm can be shown to converge to a point satisfying the necessary optimality conditions under mild assumptions.

Theorem 3 (Convergence of PBP) *Let f have bounded level sets. Then $f(\mathbf{w}^i)$ is a strictly decreasing sequence that converges to $f(\mathbf{w}^\star)$. Furthermore, if $\tau^i \le \bar{\tau},\ \forall i$, then PBP converges in a finite number of iterations and $\mathbf{w}^\star$ satisfies the necessary optimality conditions of Theorem 1.*

In our experiments, the algorithms always converges finitely with finite τ at a locally optimal solution. The only case where there may be an infinite sequence of stability centers is if τ^i is unbounded, that is, as $i\to\infty$, $\tau^i\to\infty$. For the infinite case the solution may or may not satisfy the optimality conditions. The later case is best described by the presence of a generalized nonsmooth, nonconvex saddle point. Given convexity or smoothness locally, then the necessary conditions will always be satisfied. This nonconvex nonsmooth occurrence we believe is rare and was not detected in practice throughout our experiments as the algorithm exited in a finite number of iterations. The imprecision of computer arithmetic may help the algorithm bypass these points in practice.

The proof of Theorems 2 and 3 can be found in the Appendices A and B. We now move on to describing the SVR loss functions and provide examples of critical steps for using PBP for SVR.

7.3 SVR using the PBP algorithm

Here we show how PBP solves T-fold multiSVR. The multiSVR training and validation functions give the following bilevel program:

$$\begin{aligned} &\min_{w,C,\epsilon}\frac{1}{T}\sum_{t=1}^{T}\frac{1}{|\Omega^t_{val}|}\sum_{j\in\Omega^t_{val}}\left(\mathbf{x}'_j\mathbf{w}_t-y_j\right)^2 \\ &\qquad\qquad\qquad\qquad\qquad\qquad\textbf{(outer-level)} \\ &\text{s.t. } C_{LB}\le C\le C_{UB} \\ &\quad \epsilon_{LB}\le\epsilon\le\epsilon_{UB} \\ &\quad \text{and for } t=1,\ldots,T \\ &\quad \mathbf{w}_t\in\left\{\arg\min_{\mathbf{w}_t}\frac{1}{2}\|\mathbf{w}_t\|^2+\frac{1}{2}\sum_{g=1}^{G}\frac{C_g}{|\Omega^t_{g,val}|}\sum_{j\in\Omega^t_{g,val}}\left(|\mathbf{x}'_j\mathbf{w}_t-y_j|-\epsilon_g\right)^2_+\right\}. \\ &\qquad\qquad\qquad\qquad\qquad\qquad\textbf{(inner-level)} \end{aligned} \tag{42}$$

The optimality condition to be penalized for each inner-level problem, $t=1\ldots T$, is:

$$0\in\left\{\mathbf{w}_t+\sum_{g=1}^{G}\frac{C_g}{|\Omega^t_{g,trn}|}\sum_{j\in\Omega^t_{g,trn}}\mathbf{x}'_jQ^t_{g,j}\right\} \tag{43}$$

where $Q^t_{g,j} = ((\mathbf{x}'_j\mathbf{w}_t - y_j - \epsilon_g)_+ - (-\mathbf{x}'_j\mathbf{w}_t + y_j - \epsilon_g)_+)$. Using this optimality condition gives a nonsmooth nonconvex NLP with bound constraints:

$$\begin{aligned} f(\mathbf{w}, C, \epsilon) = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{|\Omega^t_{val}|}\sum_{j\in\Omega^t_{val}}\left(\mathbf{x}'_j\mathbf{w}_t - y_j\right)^2 \\ + \sum_{t=1}^{T}\beta^t\left\|\mathbf{w}_t + \sum_{g=1}^{G}\frac{C_g}{|\Omega^t_{g,trn}|}\sum_{j\in\Omega^t_{g,trn}}\mathbf{x}'_j Q^t_{g,j}\right\|_2^2 \\ \text{s.t. } C_{LB} \le C \le C_{UB} \\ \epsilon_{LB} \le \epsilon \le \epsilon_{UB} \end{aligned} \tag{44}$$

Similarly to before, the nonsmooth nonconvex portions of the penalty terms are approximated with their first order Taylor series approximations. Thus assuming a stability center $(\mathbf{w}^i, C^i, \epsilon^i)$, choosing a subgradient $P^{t,i}_{g,j} \in \{\partial_{\mathbf{w}_t,\epsilon}((\mathbf{x}'_j\mathbf{w}^i_t - y_j - \epsilon_g)_+ - (-\mathbf{x}'_j\mathbf{w}^i_t + y_j - \epsilon_g)_+)\}$, and letting $Q^{t,i}_{g,j} = ((\mathbf{x}'_j\mathbf{w}^i_t - y_j - \epsilon^i_g)_+ - (-\mathbf{x}'_j\mathbf{w}^i_t + y_j - \epsilon^i_g)_+)$, the approximation problem with proximity control is to minimize with respect to $(\mathbf{w}, C, \epsilon)$ the following:

$$\begin{aligned} &\hat{f}(\mathbf{w}, C, \epsilon|\mathbf{w}^i, C^i, \epsilon^i) \\ &= \frac{1}{T}\sum_{t=1}^{T}\frac{1}{|\Omega^t_{val}|}\sum_{j\in\Omega^t_{val}}\left(\mathbf{x}'_j\mathbf{w}_t - y_j\right)^2 + \cdots \\ &\quad + \sum_{t=1}^{T}\beta^t\left\|\mathbf{w}_t + \sum_{g=1}^{G}\frac{1}{|\Omega^t_{g,trn}|}\sum_{j\in\Omega^t_{g,val}} C_g\mathbf{x}'_j Q^{t,i}_{g,j} + C^i_g\mathbf{x}'_j\left(P^{t,i}_{g,j}\left([\mathbf{w}_t, \epsilon] - [\mathbf{w}^i_t, \epsilon^i]\right)\right)\right\|_2^2 \\ &\quad + \tau\left\|[\mathbf{w}, C, \epsilon] - \left[\mathbf{w}^i, C^i, \epsilon^i\right]\right\|_2^2 \\ &\text{s.t. } C_{LB} \le C \le C_{UB} \\ &\quad \epsilon_{LB} \le \epsilon \le \epsilon_{UB}. \end{aligned} \tag{45}$$

This new formulation is convex and smooth for a fixed stability center and choice of P, but it has freedom in choosing which subgradient to use in the Taylor series approximation. In order to achieve a robust algorithm, the optimization program (39) must be solved to select the proper value of $P^{t,i}_{g,j}$.

Fortunately, the subgradient of $\max(\alpha, 0) = (\alpha)_+$ can be written as a convex function of the parameter p. In this case $\partial\max(\alpha, 0) = p$, where $p = 1$ if $\alpha > 0$, $p = 0$ if $\alpha < 0$ and $p \in [0, 1]$ if $\alpha = 0$. Note that value of p is fixed except at the 'kinks' where the plus function is at equality. Using this fact, we can replace problem (39) with

$$\begin{aligned} &\min_{p,\lambda} \|\mathbf{g}(p) + [0 \quad A_I^T]\lambda\| \\ &\text{s.t. } \lambda \ge 0,\ p \in P \end{aligned} \tag{46}$$

where $\mathbf{g}(p) \in \partial f(\mathbf{w}^i, \boldsymbol{\gamma}^i)$ for all $p \in P$.

The details for multiSVR are as follows.

$$\frac{\partial}{\partial \mathbf{w}_t} f(\mathbf{w}^i, C^i, \epsilon^i) = \frac{1}{T}\sum_{g=1}^{G} \frac{1}{|\Omega^t_{val}|} \sum_{j\in\Omega^t_{val}} \mathbf{x}_j \left(\mathbf{x}'_j \mathbf{w}^i_t - y_j\right) + \beta^t \left(1 + \sum_{g=1}^{G} \frac{C^i_g}{|\Omega^t_{g,trn}|} \sum_{j\in\Omega^t_{g,trn}} \mathbf{x}'_j \hat{P}^{t,i}_{g,j}\right)' Z_t \tag{47}$$

where $Z_t = \mathbf{w}_t + \sum_{g=1}^{G} \frac{C^i_g}{|\Omega^t_{g,j}|} \sum_{j\in\Omega^t_{g,trn}} \mathbf{x}'_j Q^{t,i}_{g,j}$, $Q^{t,i}_{g,j} = ((\mathbf{x}'_j \mathbf{w}^i_t - y_j - \epsilon_g)_+ - (-\mathbf{x}'_j \mathbf{w}^i_t + y_j - \epsilon_g)_+)$ and $\hat{P}^{t,i}_{g,j} \in \{\partial_{\mathbf{w}_t}((\mathbf{x}'_j \mathbf{w}^i_t - y_j - \epsilon_g)_+ - (-\mathbf{x}'_j \mathbf{w}^i_t + y_j - \epsilon_g)_+)\}$

$$\frac{\partial}{\partial C_g} f(\mathbf{w}^i_t, C^i, \epsilon^i) = \sum_{t=1}^{T} \beta^t \left(\frac{1}{|\Omega^t_{g,trn}|} \sum_{j\in\Omega^t_{g,trn}} \mathbf{x}'_j Q^{t,i}_{g,j}\right)' Z_t$$

$$\frac{\partial}{\partial \epsilon_g} f(\mathbf{w}^i_t, C^i, \epsilon^i) = \sum_{t=1}^{T} \beta^t \left(\frac{C^i_g}{|\Omega^t_{g,val}|} \sum_{j\in\Omega^t_{g,trn}} \bar{P}^{t,i}_{g,j}\right)' Z_t$$

and

$$\bar{P}^{t,i}_{g,j} \in \left\{\partial_\epsilon \left((\mathbf{x}'_j \mathbf{w}^i_t - y_j - \epsilon_g)_+ - (-\mathbf{x}'_j \mathbf{w}^i_t + y_j - \epsilon_g)_+\right)\right\}.$$

The nonsmoothness of f takes place at the small number of points where $|\mathbf{x}'_j \mathbf{w}^i_t - y_j| = \epsilon_g$ and thus $\hat{P}^{j,i}_{t,g}$ and $\bar{P}^{j,i}_{t,g}$ are not unique. Let $\hat{j}$ in group $\hat{g}$ be an example of such a point. Using our domain knowledge of SVMs, we reduce this subdifferential by constraining each component of the loss function to use a consistent subgradient. Thus $\hat{P}^{t,i}_{\hat{g},\hat{j}} \in \{\mathbf{x}_{\hat{j}} p\}$ and $\bar{P}^{t,i}_{\hat{g},\hat{j}} \in \{-p\}$ where the interval p is defined to be within is:

$$\begin{aligned} p &\in [-1, 1] \quad \text{if } \mathbf{x}'_{\hat{j}} \mathbf{w}^i_t - y_{\hat{j}} = \epsilon_{\hat{g}} = 0 \\ p &\in [0, 1] \quad \text{if } \mathbf{x}'_{\hat{j}} \mathbf{w}^i_t - y_{\hat{j}} = \epsilon_{\hat{g}} > 0 \\ p &\in [-1, 0] \quad \text{if } \mathbf{x}'_{\hat{j}} \mathbf{w}^i_t - y_{\hat{j}} = -\epsilon_{\hat{g}} < 0. \end{aligned}$$

This defines $\mathbf{g}(p) \in [\partial_{\mathbf{w}_t} f(\mathbf{w}^i_t, C^i, \epsilon^i), \partial_C f(\mathbf{w}^i_t, C^i, \epsilon^i), \partial_\epsilon f(\mathbf{w}^i_t, C^i, \epsilon^i)]$ and the bounds for p to be used in Problem (46). The subgradient chosen is the one of steepest descent with respect to the feasible directions and should be used in the approximation function $\hat{f}$, (45).

With this information, we now conduct experiments to compare the presented model selections algorithms.

8 Experiment A: Small QSAR datasets

The first experiment constructs quantitative structure activity relationship (QSAR) models to predict the bioactivity of small molecules. We reproduced the experimental design of Bennett et al. (2008) for a direct comparison with the MPEC explicit approaches. The four QSAR datasets considered are aquasol, blood/brain barrier, cancer, and cholecystokinin.

Each dataset was divided into 20 modeling/testing splits but the size of the splits varied for each dataset, according to Table 1.

For each dataset SVR with 5-fold CV was performed by grid search, three variations of the MPEC algorithm (FILTER, SLAMS, EZ-SLAMS), ImpGrad and PBP. Grid search refers to a discrete search over hyperparameters C and ϵ. FILTER (SQP) is a general purpose sequential programming algorithm applied to a NLP version of the MPEC. SLAMS and EZ-SLAMS are two variants of successive linearization algorithms for bilevel programming. These four algorithms optimize the least absolute deviation as the validation error and the ϵ-insensitive loss with 2-norm regularization for the inner-level problems. Full details of these can be found in Bennett et al. (2008). ImpGrad and PBP both constrain $C \in [10^{-3}, 10^{3}]$ and $\epsilon \in [0, \text{std}(y)]$.

All computations except FILTER were performed on a 2.33 GHz Intel Core Duo laptop running Windows XP and `MATLAB 7.6` (http://www.mathworks.com/). ImpGrad and PBP algorithms used Adrian Wills' `qpas` solver to solve the QP subproblems (QPC project, http://www.sigpromu.org/quadprog/). A linear-time subgradient algorithm (Bergeron et al. 2011a) solves SVR and multiSVR optimization problems for grid search and ImpGrad. MATLAB's routine `fmincon` uses a BFGS routine to solve the hyperparameter-update subproblem for ImpGrad.

Figures 4 and 5 report the generalization errors and computational time for the six different CV algorithms. The average generalization mean squared errors obtained on the 20 splits are reported as well as the wall clock computation times. Note that FILTER is executed on the NEOS optimization server (http://www-neos.mcs.anl.gov/). FILTER is the best of the MPEC approaches, but it is very slow on any computer.

PBP is the best method in terms of generalization error for 3 of the datasets. For these 3, PBP performs significantly better than ImpGrad. ImpGrad converges prematurely at a less desirable solution. For these small problems proper choice among the possible subgradient can be critical for implicit gradient algorithms since the subgradient may not be a direction

Table 1 Small QSAR datasets. The number of modeling, testing, and features present in each QHTS dataset

	Aquasol	Blood/Brain Barrier	Cancer	Cholecystokinin
Modeling samples	100	60	40	60
Testing samples	97	2	6	6
Features	25	25	25	25

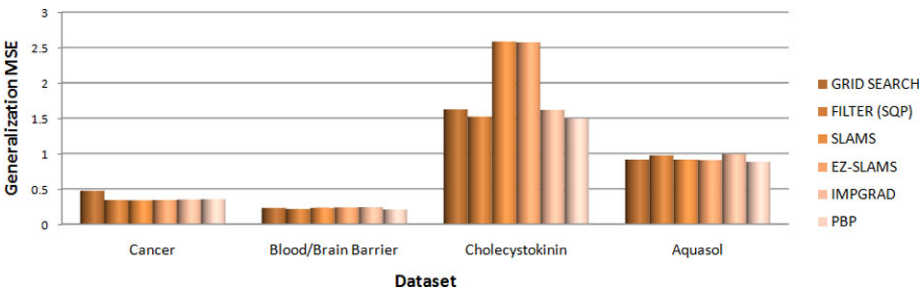

Fig. 4 Generalization mean squared errors for small QSAR datasets averaged across 20 splits

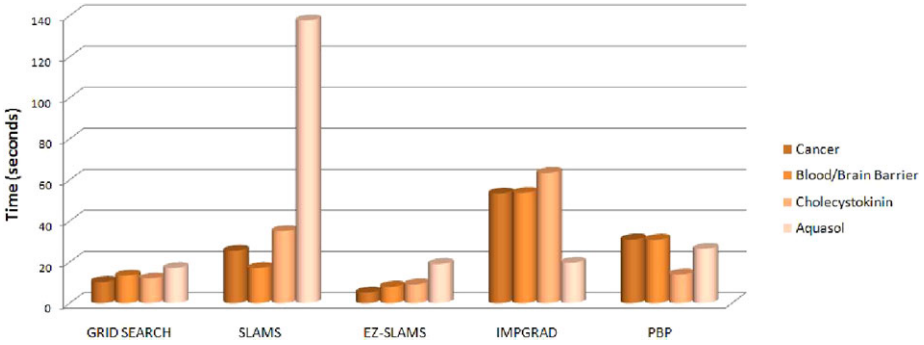

Fig. 5 Computational time for small QSAR datasets. Results are averaged across 20 splits

of decrease. PBP selects the subgradient corresponding with direction of greatest feasible descent; this is crucial for both theoretic convergence and better generalization in practice. For the fourth dataset, cancer, we note that all methods except grid search are effectively tied in terms of generalization error. Comparing execution times is inconclusive for these small datasets, as any algorithm (except perhaps FILTER) generate results quickly. As discussed by Bennett et al. (2008), EZ-SLAMS is always faster than SLAMS due to its relaxed stopping criteria, without sacrificing much error.

9 Experiment B: Large QSAR datasets

We ran experiments on two large QSAR datasets. This section describes the task, experimental design, and modeling.

9.1 Predicting drug candidate effectiveness

The task is to predict the half-maximal activity concentration (pAC_{50}), which is the negative log-concentration at which 50% of the inhibition of activation of a specified enzyme by a small drug-like molecule is achieved. Molecules that are effective at a low concentration (having high pAC_{50}) make the first cut in drug design. This is called *hit identification*. Hits undergo further testing to ensure that they are nontoxic, water soluble, metabolize well, etc. The ability to shift laboratory workload onto modeling using computers is an important step towards accelerating the speed at which new medicines are developed.

In this experiment, we have domain knowledge about the quality of each of the responses. The pAC_{50} is determined from the quantitative high-throughput screening (qHTS) experiment in which molecules are assayed at several concentrations. The classic approach fits the Hill equation to these data points using a least squares criterion. We have proposed an alternate method that exploits domain knowledge to calculate better estimates of the pAC_{50} (Bergeron et al. 2011a). We use the pAC_{50}'s from the latter technique for modeling. The "quality" groups for multiSVR modeling are defined based on both fitting techniques.

Two assays are considered in this paper: pyruvate kinase and tau-fibril. Pyruvate kinase is an enzyme involved in glucosis. Hemolytic anemia may be caused by a deficiency in pyruvate kinase; it is also a frequent target in cancer therapies. Tau proteins contribute to structural and regulatory cellular functions, and particularly on axonal transport. A deficit in tau proteins is related to neurodegenerative diseases including Alzheimer's Disease. Both

Table 2 Large QHTS datasets. The number of modeling, testing, and features in the pyruvate kinase and tau-fibril datasets

	Pyruvate kinase	Tau-fibril
Modeling samples	2000	10000
Testing samples	1383	790
Features	851	851

enzymes are described on PubChem (http://pubchem.ncbi.nlm.nih.gov/) under assay identification numbers (AID) 361 and 1463, respectively. Descriptive statistics for both datasets are found in Table 2.

For each dataset, features were calculated from molecular structures for each molecule. These consisted of MOE (implemented within the Molecular Operating Environment software, Chemical Computing Group, http://www.chemcomp.com/) and RECON (Breneman and Rhem 1997) descriptors that were generated at the Rensselaer Exploratory Center for Cheminformatics Research (RECCR, http://reccr.chem.rpi.edu/). Features were median centered, absolute deviation scaled, and thresholded to range $[-5, 5]$. Principal components analysis eliminated linearly dependent features and reduced the dimensionality to 100 components.

9.2 Experimental design

For multiSVR modeling, each dataset was split into 5 evenly-sized groups according to sample quality, where quality is defined by the relative difference between the pAC_{50} determined using the least squares and domain knowledge methods. If they both produce the same pAC_{50}, then a sample is reliable. On the other hand, if the pAC_{50}'s are not close, then the sample is uncertain. The rationale behind this is to assign samples of differing quality to different groups, and allow multiSVR to build regression models with the flexibility of different hyperparameter values for each group. In particular, we expect better quality samples (group 1, smallest relative difference) to be weighted higher in the multiSVR objective (through tradeoff C_g) and/or have lower values of the tube parameter ϵ_g, than the uncertain ones (group 5, greatest relative difference). Hence, the multiSVR problem involves 10 hyperparameters.

We consider modeling sets of increasing size to assess the impact of sample size on generalization error and execution time. We consider the following algorithms: grid search, ImpGrad and PBP algorithms. The explicit MPEC methods (SLAMS, EZ-Slams or FILTER) do not solve this problem within reasonable time.

Each algorithm performed 5-fold CV to choose C and ϵ, then a final model was trained utilizing these hyperparameters and the entire modeling set Ω. To ensure an accurate comparison, each algorithm divided the training and validation sets identically. This gives ten holdout generalization estimates after training a final model utilizing the entire modeling set. These estimates and computational time are averaged and the mean and standard deviations are reported.

All experiments were conducted on an Intel 2.4 GHz quad-core computer running Ubuntu Linux. The algorithms were programmed on `MATLAB 7.9` (http://www.mathworks.com/) and Adrian Wills' `qpas` solver was used to solve the QP subproblem (QPC project, http://www.sigpromu.org/quadprog/). A linear-time subgradient algorithm (Bergeron et al. 2011b) solves SVR and multiSVR optimization problems for grid search and ImpGrad. MATLAB's routine `fmincon` uses a BFGS routine to solve the hyperparameter-update subproblem within ImpGrad.

The remainder of this subsection contains algorithm-specific experimental design issues. Grid search is conducted for single group SVR over $C \in \{10^k, k = -4, \ldots, 3\}$ and $\epsilon \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$. These choices result in $8 \cdot 6 = 48$ grid points. Optimality of the training problem is achieved when a subgradient is found with norm less than 0.001.

For multiSVR, the number of grid points quickly blows up. In our experimental design, there are 8 choices for each C_g and 6 choices for each ϵ_g, for a total of $8^5 6^5 \approx 254$ million points to consider across a 10-dimensional grid, an unmanageable task, which explains why multi-hyperparameter modeling is not commonplace. We therefore restrict ourselves to a much smaller grid distribution of $C \in \{0.1, 10\}$ and $\epsilon \in \{0, 1\}$ that still involves 1024 points in a 10-dimensional grid. While this grid may seem insufficient, it will require solving 21 times more SVM problems than for machine learning optimization problems than for the finer grid with SVR and 256 more problems than using the coarse grid with SVR.

For ImpGrad, the starting hyperparameter values were $C = 1$ and $\epsilon = 0$. Hyperparameters were constrained to remain within the extremes of the grid search. Termination tolerances are $tol_1 = tol_2 = 10^{-3}$.

For PBP, the set of polyhedral constraints restricts the hyperparameters to be within the extremes of the grid search for a fair comparison. For better numerical properties, first the model selection process was solved with $\epsilon = 0$ fixed, and then once C is optimal, both hyperparameters are optimized together. Termination tolerances are $tol = \zeta = \eta = 10^{-3}$.

9.3 Results

Figure 6 and Table 3 presents results for grid search, ImpGrad and PBP on the pyruvate kinase dataset. We make the following observations:

- For each reported choice of model (SVR or multiSVR) and modeling set size, PBP finds the smallest generalization error with low variance.
- As the sample size increases, ImpGrad's generalization error approaches that of PBP with a negligible difference at 2000 points and beyond. However, ImpGrad is much more computationally efficient than PBP for large sample sizes. Thus for multiSVR, ImpGrad is preferable for large problem sizes.
- Grid search returned the worst generalization errors. This was expected as it is limited to a discrete sampling of hyperparameter combinations; the other algorithms are not so constrained.
- Grid search with a finer mesh is expected to return a lower generalization error than when using a coarse mesh, but we see in Table 3 that this is not always true. As expected, the coarse grid always executes faster than the fine one.

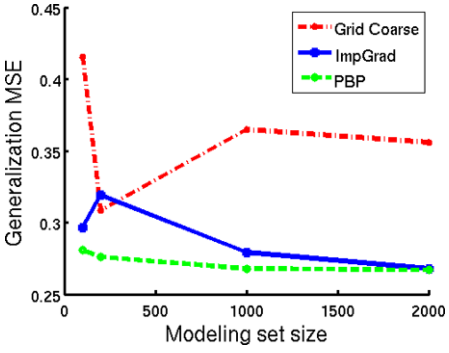

Fig. 6 Generalization results for the pyruvate kinase dataset with 5-groups and using multiSVR. The plot shows that PBP performs best overall. Coarse grid performs poorly. ImpGrad has some trouble with small datasets but then achieves results comparable to PBP on the larger datasets. See Table 3 for full results

Table 3 Results for pyruvate kinase dataset. For each model and modeling set size, the result with the lowest generalization MSE is highlighted. Results are presented as a mean and standard deviation across 10 splits

Method	SVR		multiSVR	
	Generalization MSE	Time in seconds	Generalization MSE	Time in seconds
100 pts				
GRID FINE	0.363 ± 0.161	23.1 ± 1.7	–	–
GRID COARSE	0.532 ± 0.466	11.8 ± 1.3	0.415 ± 0.129	3557.2 ± 264.9
IMPGRAD	0.308 ± 0.040	96.6 ± 50.4	0.297 ± 0.026	167.1 ± 80.8
PBP	$\mathbf{0.274 \pm 0.013}$	25.1 ± 48.2	$\mathbf{0.281 \pm 0.015}$	37.0 ± 62.6
200 pts				
GRID FINE	0.483 ± 0.457	23.9 ± 2.2	–	–
GRID COARSE	0.521 ± 0.395	11.9 ± 1.0	0.309 ± 0.036	3944.1 ± 163.4
IMPGRAD	0.562 ± 0.108	58.3 ± 36.6	0.319 ± 0.037	549.5 ± 36.3
PBP	$\mathbf{0.269 \pm 0.012}$	12.3 ± 7.2	$\mathbf{0.276 \pm 0.011}$	25.1 ± 27.8
1000 pts				
GRID FINE	0.552 ± 0.303	116.7 ± 7.2	–	–
GRID COARSE	0.445 ± 0.298	54.4 ± 4.4	0.365 ± 0.190	2797.2 ± 57.6
IMPGRAD	0.285 ± 0.011	57.9 ± 29.0	0.279 ± 0.012	77.4 ± 9.8
PBP	$\mathbf{0.264 \pm 0.011}$	93.5 ± 57.7	$\mathbf{0.268 \pm 0.012}$	146.1 ± 121.3
2000 pts				
GRID FINE	0.616 ± 0.277	602.3 ± 187.6	–	–
GRID COARSE	0.663 ± 0.397	161.5 ± 21.9	0.356 ± 0.172	2619.7 ± 32.3
IMPGRAD	0.270 ± 0.012	55.2 ± 3.5	0.268 ± 0.012	69.2 ± 16.0
PBP	$\mathbf{0.264 \pm 0.011}$	161.9	$\mathbf{0.267 \pm 0.009}$	218.5 ± 115.2

– As the validation and generalization metrics are the same for SVR and multiSVR, it does not appear from Table 3 that there is any predictive gains in building models with several C's and ϵ's for this dataset and task.

Figure 7 and Table 4 repeat Figure 6 and Table 3 on tau-fibril. On large problems for several choices of model (SVR or multiSVR) and modeling set size, ImpGrad finds high quality solutions much quicker than PBP. For small problems, ImpGrad finds worse solutions with higher variance than PBP. PBP achieves good generalization for all training-set sizes but more efforts are needed it to optimize it's computation time on larger datasets. We also note that multiSVR models do not significantly improve generalization over SVR models but there is no evidence of over-fitting with use of more hyperparameters. Grid remains the poorest algorithm; we analyze execution time for the grid searches with a modeling set consisting of 100 points. The coarse grid for multiSVR is approximately 400 times slower than coarse grid with SVR and 35 times slower than fine grid with SVR. These numbers are of the same order of magnitude as 256 and 21, respectively, that were estimated in the previous subsection.

We can inspect the values for C and ϵ for PBP trained from both models. The story is the same for both datasets. We found $C = C_g \approx 10^{-4}, g = 1, \ldots, G$, meaning that the groups were not weighed differently in the multiSVR problem. Each C_g took on approximately the

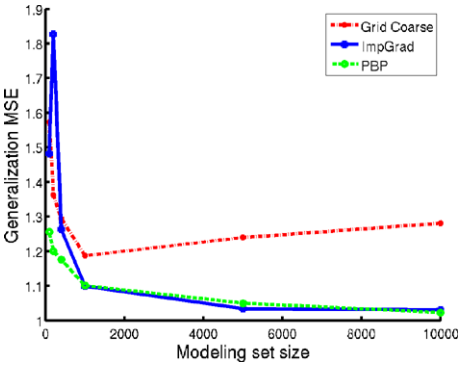

Fig. 7 Partial results for tau-fibril dataset on 5-group multiSVR. The plot shows that the coarse grid does poorly, while ImpGrad works erratically for small dataset sizes and then improves, and PBP does the best consistently. Notice that the PBP and ImpGrad results converges for the large datasets. See Table 4 for full results on pyruvate kinase

Table 4 Results for tau-fibril dataset. For each model and modeling set size, the result with the lowest generalization MSE is highlighted. Results are presented as a mean and standard deviation across 10 splits

Method	SVR		multiSVR	
	Generalization MSE	Time in seconds	Generalization MSE	Time in seconds
100 pts				
GRID FINE	1.282 ± 0.109	181.6 ± 17.6	–	–
GRID COARSE	1.474 ± 0.204	15.1	1.573 ± 0.232	6316.8 ± 260.3
IMPGRAD	1.667 ± 0.197	53.2 ± 16.4	1.482 ± 0.259	426.0 ± 415.4
PBP	$\mathbf{1.227 \pm 0.051}$	122.4 ± 44.8	$\mathbf{1.256 \pm 0.073}$	220.9 ± 240.1
200 pts				
GRID FINE	1.481 ± 0.291	170.8 ± 8.8	–	–
GRID COARSE	1.544 ± 0.266	15.1	1.362 ± 0.235	7084.1 ± 276.5
IMPGRAD	1.936 ± 0.561	75.6 ± 51.9	1.826 ± 0.486	792.8 ± 421.0
PBP	$\mathbf{1.204 \pm 0.073}$	98.4 ± 55.5	$\mathbf{1.200 \pm 0.051}$	169.7 ± 73.1
1000 pts				
GRID FINE	1.311 ± 0.144	363.6 ± 16.4	–	–
GRID COARSE	1.434 ± 0.229	66.3 ± 8.8	1.187 ± 0.160	4326.9 ± 162.0
IMPGRAD	1.121 ± 0.049	44.2 ± 4.0	$\mathbf{1.099 \pm 0.051}$	83.2 ± 19.0
PBP	$\mathbf{1.116 \pm 0.047}$	119.3 ± 26.1	1.101 ± 0.055	326.5 ± 99.7
5000 pts				
GRID FINE	1.317 ± 0.072	5259.7 ± 114.0	–	–
GRID COARSE	1.344 ± 0.141	906.0 ± 77.1	1.240 ± 0.123	4755.4 ± 21.2
IMPGRAD	$\mathbf{1.035 \pm 0.053}$	72.4 ± 11.2	$\mathbf{1.034 \pm 0.054}$	90.0 ± 13.7
PBP	1.244 ± 0.042	66.9 ± 33.6	1.050 ± 0.088	1893.2 ± 1123.2
10000 pts				
GRID COARSE	1.372 ± 0.142	4935.5 ± 508.1	1.281 ± 0.099	6794.4 ± 255.1
IMPGRAD	$\mathbf{1.030 \pm 0.054}$	126.7 ± 12.1	1.030 ± 0.051	109.5 ± 24.1
PBP	1.231 ± 0.037	129.7 ± 67.0	$\mathbf{1.023 \pm 0.054}$	4394.8 ± 1204.2

value of the single C in SVR. We also found $\epsilon = \epsilon_1 = \epsilon_2 = \epsilon_3 = 0$, $\epsilon_4 \approx 0.1$ and $\epsilon \approx 0.5$. This suggests that PBP is identifying that groups 4 and especially 5 contain samples whose pAC_{50} are increasingly uncertain, and assigning higher values for ϵ to these. Hence, we

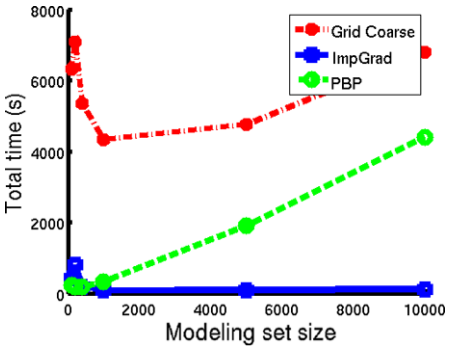

Fig. 8 Scalability in sample size for Grid, ImpGrad and PBP with 10 hyperparameters for the tau-fibril dataset. Note Grid Coarse is only selecting over two choices per hyperparameters; a full grid search is impractical

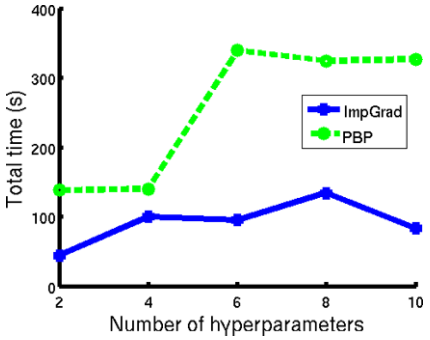

Fig. 9 Scalability in hyperparameter size for multiSVM. The hyperparameter scalability for ImpGrad and PBP with two to ten hyperparameters (one to five groups) for the 1000 point tau-fibril dataset

have successfully used a multiple-hyperparameter model in an explicative setting, that is, to detect the relative quality of data segments in a dataset.

We seek to develop algorithms that scale linearly in sample size. Figure 8 assesses the empirical scalability of the algorithms presented in this paper. Coarse grid search uses more computational time than the other algorithms at 100 modeling points, and then its computational time begins to grow as expected after 1000 points. A full fine grid is impractical and would be off the chart. ImpGrad and PBP scale modesty. ImpGrad has high variability for the smaller datasets, while PBP is more consistent, but grows at a higher rate. For larger datasets ImpGrad was the fastest.

The ability of an algorithm to scale in the number of hyperparameters is important. Figure 9 addresses this issue for ImpGrad and PBP using the tau-fibril 1000 modeling point datasets. The plot shows that as the number of hyperparameters (groups) are increased from two hyperparameters (one group) to 10 hyperparameters (five groups), the computation time is quite reasonable. Further study into this relationship is necessary.

Coarse grid search was reasonably fast; faster than both ImpGrad and PBP. In terms of generalization though, coarse grid search performed the worst. Implicit and PBP algorithms performed better, with PBP being faster on the smaller datasets and ImpGrad being faster on the larger datasets. Generalization was slightly better for PBP.

The ability to train large models is important, and here we see that grid search scales linearly in the number of training parameters as expected because it uses a linear time algorithm. ImpGrad's computational time actually decreases given a larger set of modeling samples. PBP's computation costs grow roughly linearly in sample size.

10 Conclusions

This paper makes several significant contributions to hyperparameter selection for SVM-type methods. First, it shows how the bilevel programming approach to hyperparameter optimization can produce two types of methods: implicit and explicit algorithms. While implicit methods are the most broadly used in machine learning, the paper established that explicit methods that optimize both the SVM-type model hyperparameters and parameters are very competitive and offer advantages over explicit methods. Second, we developed two scalable methods for selecting hyperparameters of SVM-type regression, ImpGrad and PBP, which allow novel models to be developed containing both many points and many hyperparameters. We have seen that model selection is an important part of machine learning; suboptimal models lead to poor generalization ability. Modern model selection algorithms are able to find good models in a reasonable amount of time.

The computational results clearly show that primal implicit and explicit methods for multiple hyperparameter model selection of SVM-type models can efficiently perform directly in the primal space for problems with many hyperparameters and data points. It is clear from the results that modeling with many hyperparameters is beyond the capabilities of grid search and alternative approaches should become the norm in machine learning.

ImpGrad finds solutions with good generalization very quickly for large datasets, but illustrates more erratic behavior on all of the small datasets. For small datasets, we believe the practice of picking one of the many possible subgradients for use in the BFGS leads to poor performance. The difficultly comes from points that lie exactly on the ϵ tube. For large datasets, these bounding points do not dominate the objective, so that the leeway in the choice of subgradient is less critical. This should serve as a cautionary tale that simply picking any subgradient may not always work. Doing a little extra work to search for a good subgradient may yield much better performance. New nonsmooth BFGS methods could be exploited (Skajaa 2010). These enhancements are left for future work. The method shows excellent scalability but a more formal analysis of the convergence and scalability is needed. ImpGrad shares the benefits of all implicit methods, namely that the problem naturally decomposes the problem into subproblem for each fold that can be solved by any appropriate existing SVM algorithm. It does this without generating or using the kernel matrix or dual variables. Future work includes looking at ways to finding the best combination of ImpGrad's speed on the larger datasets and PBP's robustness on the smaller datasets.

PBP uses a well-founded subgradient method with proven convergence properties and yields a robust explicit algorithm that performed well on problems of all sizes. While it appears to be roughly linear in the training time required per modeling set size, it does work significantly harder than ImpGrad on the 5000 and 10000 sized tau-fibril datasets, for roughly the same generalization. PBP works by solving a series of constrained least squares problems that approximate the original problem. The subgradient search insures that this approximation is locally good. Currently a general purpose QP solver is used to solve these subproblems. A special-purpose primal solver for constrained least squares problems could greatly accelerate PBP much as special-purpose SVM solvers do for implicit methods. The advantage of the explicit approach is that you do not need a different core algorithm for different losses, just one powerful simpler solver. The potential of this approach is great and future work is needed to determine the best approach.

Like all machine learning algorithms, PBP and ImpGrad have algorithm parameters that must be defined such as exit criteria, starting points, and proximity parameters. These new parameters beg the question: Is it necessary to cross-validate over these new algorithm parameters? We believe that these algorithm parameters can be set by policy just as SVM

training algorithm parameters are set today in publicly available packages. The model hyperparameters have a direct and significant impact on the quality of the final model. We can rely on regularization parameters along with model selection algorithms to improve model robustness rather than algorithmic strategies such as early stopping.

The contribution of the PBP algorithm goes beyond machine learning; it is also applicable to general bilevel programs. The nonsmooth implicit and explicit bilevel-programming approaches examined here are entirely novel in the mathematical programming arena. The problems attempted here are massive in comparison to problems reported in bilevel programming papers. The techniques developed here can be applied to the many other applications of bilevel programming in economics, management, planning, transportation, and design.

A limitation of this paper is that ImpGrad and PBP assume that the inner-level objective functions are at least once differentiable. Also, the results presented are limited to support vector regression (SVR). In principal, both the ImpGrad and PBP can be extended to subdifferential inner-level objectives involving subdifferentiable loss and 1-norm regularization. Once again the chief difficulty will come from selecting which subgradient to use to guarantee descent of the outer-level gradient. However, the success of dual implicit methods for large problems indicates that such subgradients are readily computable in practice. We leave this to future work.

The ability to efficiently optimize models with many hyperparameters allows us to explore new questions. What types of models with many hyperparameters will yield better performance on existing learning tasks and perhaps enable new modeling tasks? Do we need to regularize the hyperparameters as in Bayesian methods (Seeger 1999)? What are the best combinations of outer validation and inner training loss and regularization functions?

Acknowledgements This work was funded by grants by the Office of Naval Research (number N00014-06-1-0014) and the National Institutes of Health (number R01LM009731). The authors thank Prof. Curt M. Breneman and Mr. Michael Krein at the Rensselaer Exploratory Center for Cheminformatics Research (RECCR, http://reccr.chem.rpi.edu/) for generating the QSAR features for the pyruvate kinase and tau-fibril datasets.

Appendix A

Here we first present the proof of Theorem 2 (Greatest Feasible Descent Direction) and follow this with details of the convergence of the PBP algorithm.

A.1 Proof of Theorem 2

Proof Note, that $\{\mathbf{d} \mid [0 \;\; A_I]\mathbf{d} \leq 0\}$ is the set of feasible directions $\mathbf{d}$ at the current iterate $\mathbf{w}, \boldsymbol{\gamma}$. For simplicity of the proof, let $A = [\mathbf{0} \;\; A_I]$.

We begin with problem (39) and show it is equivalent to problem (41). We represent the convex polyhedral set $\mathbf{g} \in \partial f$ as a polyhedral set denoted as $J\mathbf{g} \leq \mathbf{j}$. The two-norm in the objective replaces the two-norm squared, which is used above only for computational efficiency purposes; for this problem structure, a solution to the two-norm squared problem is also a solution to the two-norm un-squared problem. Thus problem (39) becomes:

$$\begin{aligned} &\min_{\mathbf{g},\boldsymbol{\lambda}} \|\mathbf{g} + A'\boldsymbol{\lambda}\| \\ &\text{s.t.} \quad \boldsymbol{\lambda} \geq 0 \\ &\qquad J\mathbf{g} \leq j. \end{aligned} \tag{A.1}$$

Its equivalent Wolfe dual (Mangasarian 1994), with dual variables $\mathbf{u}$ and $\mathbf{v}$ is:

$$\begin{aligned} \max_{\mathbf{g},\boldsymbol{\lambda},\mathbf{u},\mathbf{v}} \ & \|\mathbf{g}+A'\boldsymbol{\lambda}\|+\mathbf{u}'(J\mathbf{g}-\mathbf{j})-\mathbf{v}'\boldsymbol{\lambda} \\ \text{s.t.} \ & \frac{\mathbf{g}+A'\boldsymbol{\lambda}}{\|\mathbf{g}+A'\boldsymbol{\lambda}\|}+J'\mathbf{u}=0 \\ & \frac{A(\mathbf{g}+A'\boldsymbol{\lambda})}{\|\mathbf{g}+A'\boldsymbol{\lambda}\|}-\mathbf{v}=0 \\ & \mathbf{u}\geq 0 \\ & \mathbf{v}\geq 0. \end{aligned} \tag{A.2}$$

Eliminating the variable $\mathbf{v}$ yields a simplified problem:

$$\begin{aligned} \max_{\mathbf{g},\boldsymbol{\lambda},\mathbf{u}} \ & \|\mathbf{g}+A'\boldsymbol{\lambda}\|+\mathbf{u}'(J\mathbf{g}-\mathbf{j})-\left(\frac{A(\mathbf{g}+A'\boldsymbol{\lambda})}{\|\mathbf{g}+A'\boldsymbol{\lambda}\|}\right)'\boldsymbol{\lambda} \\ \text{s.t.} \ & \frac{\mathbf{g}+A'\boldsymbol{\lambda}}{\|\mathbf{g}+A'\boldsymbol{\lambda}\|}+J'\mathbf{u}=0 \\ & \frac{A(\mathbf{g}+A'\boldsymbol{\lambda})}{\|\mathbf{g}+A'\boldsymbol{\lambda}\|}\geq 0 \\ & \mathbf{u}\geq 0 \end{aligned} \tag{A.3}$$

and utilizing the first constraint to simplify the objective and second constraint yields:

$$\begin{aligned} \max_{\mathbf{g},\boldsymbol{\lambda},\mathbf{u}} \ & \|\mathbf{g}+A'\boldsymbol{\lambda}\|+\mathbf{u}'(J\mathbf{g}-\mathbf{j})+\mathbf{u}'JA'\boldsymbol{\lambda} \\ \text{s.t.} \ & \frac{\mathbf{g}+A'\boldsymbol{\lambda}}{\|\mathbf{g}+A'\boldsymbol{\lambda}\|}+J'\mathbf{u}=0 \\ & AJ'\mathbf{u}\geq 0 \\ & \mathbf{u}\geq 0. \end{aligned} \tag{A.4}$$

Rewriting the objective produces

$$\begin{aligned} \max_{\mathbf{g},\boldsymbol{\lambda},\mathbf{u}} \ & \|\mathbf{g}+A'\boldsymbol{\lambda}\|+\mathbf{u}'J(\mathbf{g}+A'\boldsymbol{\lambda})-\mathbf{u}'\mathbf{j} \\ \text{s.t.} \ & \frac{\mathbf{g}+A'\boldsymbol{\lambda}}{\|\mathbf{g}+A'\boldsymbol{\lambda}\|}+J'\mathbf{u}=0 \\ & AJ'\mathbf{u}\geq 0 \\ & \mathbf{u}\geq 0 \end{aligned} \tag{A.5}$$

Now we claim that $\|\mathbf{g}+A'\boldsymbol{\lambda}\|+\mathbf{u}'J(\mathbf{g}+A'\boldsymbol{\lambda})$ in the objective is equal to zero due to the first constraint. Multiplying the first constraint by $\mathbf{u}'J\|\mathbf{g}+A'\boldsymbol{\lambda}\|$ on the left gives: $\mathbf{u}'JJ'\mathbf{u}\|\mathbf{g}+A'\boldsymbol{\lambda}\|+\mathbf{u}'J(\mathbf{g}+A'\boldsymbol{\lambda})=0$. Now taking the norm of the first constraint we know that $\mathbf{u}'JJ'\mathbf{u}=1$. Therefore the problem reduces to:

$$\begin{aligned} \min_{\mathbf{g},\boldsymbol{\lambda},\mathbf{u}} \ & \mathbf{u}'\mathbf{j} \\ \text{s.t.} \ & \frac{\mathbf{g}+A'\boldsymbol{\lambda}}{\|\mathbf{g}+A'\boldsymbol{\lambda}\|}+J'\mathbf{u}=0 \\ & AJ'\mathbf{u}\geq 0 \\ & \mathbf{u}\geq 0 \end{aligned} \tag{A.6}$$

and noting that the depreciated dependence on $\mathbf{g}$ and $\boldsymbol{\lambda}$ involves only the norm of $J'\mathbf{u} = 1$ provides:

$$\begin{aligned} &\min_{\mathbf{u}} \mathbf{u}'\mathbf{j} \\ &\text{s.t. } \|J'\mathbf{u}\| = 1 \\ &\quad AJ'\mathbf{u} \geq 0 \\ &\quad \mathbf{u} \geq 0 \end{aligned} \tag{A.7}$$

From here we let $J'\mathbf{u} = \mathbf{d}$ giving:

$$\begin{aligned} &\min_{\mathbf{d},\mathbf{u}} \mathbf{u}'\mathbf{j} \\ &\text{s.t. } \|\mathbf{d}\| = 1 \\ &\quad A\mathbf{d} \geq 0 \\ &\quad J'\mathbf{u} = \mathbf{d} \\ &\quad \mathbf{u} \geq 0 \end{aligned} \tag{A.8}$$

and now splitting up the minimization problems:

$$\begin{aligned} &\min_{\mathbf{d}} \min_{\mathbf{u}} \mathbf{u}'\mathbf{j} \\ &\text{s.t. } \|\mathbf{d}\| = 1 \\ &\quad A\mathbf{d} \geq 0 \\ &\quad J'\mathbf{u} = \mathbf{d} \\ &\quad \mathbf{u} \geq 0 \end{aligned} \tag{A.9}$$

which is exactly problem (41) where the inner maximization is replaced by its dual minimization problem. □

This completes the proof of Theorem 2 (Greatest Feasible Descent Direction). We now detail the convergence of the algorithm.

Appendix B: Convergence of PBP

To prove the convergence of the PBP algorithm, we start with an analysis of the stability center updates, followed by an analysis of the overall convergence of the algorithm. For simplicity, we refer to $\hat{f}(\mathbf{w}, \boldsymbol{\gamma}|\mathbf{w}^i, \boldsymbol{\gamma}^i)$ simply as $\hat{f}(\mathbf{w}, \boldsymbol{\gamma})$ and assume it is centered around $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$. Thus stability centers will be referred to at $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$ and candidate points of the inner loop will be referred to as $(\mathbf{w}, \boldsymbol{\gamma})$.

B.1 Strict decrease in stability centers

We begin showing the convergence of the inner loop of PBP by eliminating the special case of the inner loop returning the stability center.

Lemma 1 *Given a stability center* $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$, *suppose the solution to Problem* (36) (*or equivalently Problem* (37)) *is itself, namely* $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$, *then* $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$ *is an optimal solution to Problem* (34) *with fixed* β.

Proof By the optimality condition of Problem (36), $0 \in \partial \hat{f}(\mathbf{w}^i, \boldsymbol{\gamma}^i) + A'\hat{\rho}$, where $\hat{\rho} \geq 0$, $(A\boldsymbol{\gamma}^i - \mathbf{c}) \geq 0$ and $\hat{\rho}'(A\boldsymbol{\gamma}^i - \mathbf{c}) = 0$. The optimality condition of (34) is similar $0 \in \partial(f(\mathbf{w}^i, \boldsymbol{\gamma}^i)) + A'\rho$, where $\rho \geq 0$, $(A\boldsymbol{\gamma}^i - \mathbf{c}) \geq 0$, and $\rho'(A\boldsymbol{\gamma}^i - \mathbf{c}) = 0$. Clearly setting $\rho = \hat{\rho}$ and noting by construction of $\hat{f}$ (requirement *(D2)*) $\partial(\hat{f}(\mathbf{w}^i, \boldsymbol{\gamma}^i)) \in \partial(f(\mathbf{w}^i, \boldsymbol{\gamma}^i))$, therefore $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$ must be optimal for (34) as well. □

This means that if $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$ is optimal, then the inner loop would never be entered and the algorithm would exit. The next lemma and theorem show that the inner loop will converge in a finite number of iterations to a point giving a strict decrease in the original function.

Lemma 2 *Assume that* $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$ *is not locally optimal for* f *and* $(\mathbf{w}, \boldsymbol{\gamma})$ *is sufficiently close to* $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$, *then* $f(\mathbf{w}, \boldsymbol{\gamma}) - \hat{f}(\mathbf{w}^i, \boldsymbol{\gamma}^i) < 0$.

Proof We begin by adding and subtracting $\hat{f}(\mathbf{w})$ giving:

$$f(\mathbf{w}, \boldsymbol{\gamma}) - \hat{f}(\mathbf{w}^i, \boldsymbol{\gamma}^i) = f(\mathbf{w}, \boldsymbol{\gamma}) - \hat{f}(\mathbf{w}, \boldsymbol{\gamma}) + \hat{f}(\mathbf{w}, \boldsymbol{\gamma}) - \hat{f}(\mathbf{w}^i, \boldsymbol{\gamma}^i). \tag{B.1}$$

By construction $\hat{f}(\mathbf{w}, \boldsymbol{\gamma}) = f(\mathbf{w}^i, \boldsymbol{\gamma}^i) + \mathbf{g}'([\mathbf{w}, \boldsymbol{\gamma}] - [\mathbf{w}^i, \boldsymbol{\gamma}^i])$, thus we get:

$$f(\mathbf{w}, \boldsymbol{\gamma}) - \hat{f}(\mathbf{w}^i, \boldsymbol{\gamma}^i) = f(\mathbf{w}, \boldsymbol{\gamma}) - \hat{f}(\mathbf{w}, \boldsymbol{\gamma}) + \mathbf{g}'([\mathbf{w}, \boldsymbol{\gamma}] - [\mathbf{w}^i, \boldsymbol{\gamma}^i]). \tag{B.2}$$

From requirement (D4), we know that there exists an $\Delta > 0$ such that $f(\mathbf{w}, \boldsymbol{\gamma}) - \hat{f}(\mathbf{w}, \boldsymbol{\gamma}) \leq \epsilon \|[\mathbf{w}, \boldsymbol{\gamma}] - [\mathbf{w}^i, \boldsymbol{\gamma}^i]\|$ for all $\epsilon > 0$ and $\|[\mathbf{w}, \boldsymbol{\gamma}] - [\mathbf{w}^i, \boldsymbol{\gamma}^i]\| \leq \Delta$. We assume that $(\mathbf{w}, \boldsymbol{\gamma})$ is sufficiently close to $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$ so that $\|[\mathbf{w}, \boldsymbol{\gamma}] - [\mathbf{w}^i, \boldsymbol{\gamma}^i]\| \leq \Delta$ and set $\epsilon = \frac{\mathbf{g}'([\mathbf{w}, \boldsymbol{\gamma}] - [\mathbf{w}^i, \boldsymbol{\gamma}^i])}{\|[\mathbf{w}, \boldsymbol{\gamma}] - [\mathbf{w}^i, \boldsymbol{\gamma}^i]\|}$.

Now because $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$ is not optimal, and $\mathbf{g}$ was selected as the gradient yielding the direction of greatest descent, $\mathbf{g}'([\mathbf{w}, \boldsymbol{\gamma}] - [\mathbf{w}^i, \boldsymbol{\gamma}^i]) < 0$ and this gives:

$$f(\mathbf{w}, \boldsymbol{\gamma}) - \hat{f}(\mathbf{w}^i, \boldsymbol{\gamma}^i) < 0 \tag{B.3}$$

as desired. □

Theorem 4 (Strict descent after inner loop) *The inner loop of PBP will exit after a finite number of iterations. That is, a solution* $(\mathbf{w}, \boldsymbol{\gamma})$ *will be found such that* $f(\mathbf{w}, \boldsymbol{\gamma}) - f(\mathbf{w}^i, \boldsymbol{\gamma}^i) < \rho(\hat{f}(\mathbf{w}, \boldsymbol{\gamma}) - f(\mathbf{w}^i, \boldsymbol{\gamma}^i))$ *in a finite number of iterations.*

Proof

$$\begin{aligned} f(\mathbf{w}, \boldsymbol{\gamma}) - f(\mathbf{w}^i, \boldsymbol{\gamma}^i) &= f(\mathbf{w}, \boldsymbol{\gamma}) - (1-\rho)\hat{f}(\mathbf{w}^i, \boldsymbol{\gamma}^i) + (1-\rho)\hat{f}(\mathbf{w}^i, \boldsymbol{\gamma}^i) - f(\mathbf{w}^i, \boldsymbol{\gamma}^i) \\ &= f(\mathbf{w}, \boldsymbol{\gamma}) - \hat{f}(\mathbf{w}^i, \boldsymbol{\gamma}^i) + \rho(\hat{f}(\mathbf{w}^i, \boldsymbol{\gamma}^i) - f(\mathbf{w}^i, \boldsymbol{\gamma}^i)) \end{aligned} \tag{B.4}$$

which by Lemma 2 and assuming that $\|[\mathbf{w}, \boldsymbol{\gamma}] - [\mathbf{w}^i, \boldsymbol{\gamma}^i]\|$ is sufficiently small gives our result that $f(\mathbf{w}, \boldsymbol{\gamma}) - f(\mathbf{w}^i, \boldsymbol{\gamma}^i) < \rho(\hat{f}(\mathbf{w}, \boldsymbol{\gamma}) - f(\mathbf{w}^i, \boldsymbol{\gamma}^i))$. $\|[\mathbf{w}, \boldsymbol{\gamma}] - [\mathbf{w}^i, \boldsymbol{\gamma}^i]\|$ is guaranteed to become sufficiently small in a finite number of iterations as at each null step the proximity parameter is doubled, restricting $\|\mathbf{d}\|$. □

B.2 Convergence theorem

With knowledge of the strict decreases in stability centers, we proceed to convergence of the entire algorithm.

Theorem 5 (Finite Termination of PBP) *Let f have bounded level sets. Then $f(\mathbf{w}^i, \boldsymbol{\gamma}^i)$ is a strictly decreasing sequence that converges to $f(\mathbf{w}^\star, \boldsymbol{\gamma}^\star)$. Furthermore if $\tau^i \leq \bar{\tau},\ \forall i$, then PBP converges in a finite number of iterations and $(\mathbf{w}^\star, \boldsymbol{\gamma}^\star)$ satisfies the necessary optimality conditions of Theorem 1.*

Proof By the above lemma, the sequence $f(\mathbf{w}^i, \boldsymbol{\gamma}^i)$ is monotonically decreasing and bounded below thus it must converge to some $f(\mathbf{w}^\star, \boldsymbol{\gamma}^\star)$.

If a feasible stability center $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$ is found such that $\|g + A\boldsymbol{\lambda}\|^2 \leq tol(g \in \partial f(\mathbf{w}^i, \boldsymbol{\gamma}^i), \boldsymbol{\lambda} \geq 0)$, then a local optimal solution has been found. Therefore we now assume that the outer loop generates an infinite sequence of strictly decreasing stability centers, and thus $g^T([\mathbf{w}, \boldsymbol{\gamma}] - [\mathbf{w}^i, \boldsymbol{\gamma}^i]) \geq tol|[\mathbf{w}, \boldsymbol{\gamma}] - [\mathbf{w}^i, \boldsymbol{\gamma}^i]|$.

Under the assumption that f has bounded level sets on the feasible region, if follows directly that a subsequence of stability centers $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$ converge to an accumulation point.

The decrease seen in f given the sequence of stability centers $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$, under the said assumption can be characterized as:

$$\begin{aligned} f(\mathbf{w}^i, \boldsymbol{\gamma}^i) - f(\mathbf{w}^{i+1}, \boldsymbol{\gamma}^{i+1}) &> \rho(f(\mathbf{w}^i, \boldsymbol{\gamma}^i) - \hat{f}(\mathbf{w}^{i+1}, \boldsymbol{\gamma}^{i+1})) \\ &= \rho(f(\mathbf{w}^i, \boldsymbol{\gamma}^i) - (f(\mathbf{w}^i, \boldsymbol{\gamma}^i) + \mathbf{g}'([\mathbf{w}^{i+1}, \boldsymbol{\gamma}^{i+1}] - [\mathbf{w}^i, \boldsymbol{\gamma}^i]))) \\ &= \rho\mathbf{g}'([\mathbf{w}^i, \boldsymbol{\gamma}^i] - [\mathbf{w}^{i+1}, \boldsymbol{\gamma}^{i+1}]). \end{aligned} \tag{B.5}$$

and if τ^i is bounded for all i and $1 > \rho > 0$, then we claim there exists and $\eta > 0$ such that

$$\rho\mathbf{g}'([\mathbf{w}^i, \boldsymbol{\gamma}^i] - [\mathbf{w}^{i+1}, \boldsymbol{\gamma}^{i+1}]) > \eta > 0. \tag{B.6}$$

To show this we exploit the structure of (36) and assumptions that τ^i is bounded above and $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$ is not optimal. Under the assumption that $(\mathbf{w}^i, \boldsymbol{\gamma}^i)$ is not optimal, then $\mathbf{g}'([\mathbf{w}^{i+1}, \boldsymbol{\gamma}^{i+1}] - [\mathbf{w}^i, \boldsymbol{\gamma}^i]) < tol\|[\mathbf{w}^{i+1}, \boldsymbol{\gamma}^{i+1}] - [\mathbf{w}^i, \boldsymbol{\gamma}^{i+1}]\|$. Finally this gives that

$$\rho\mathbf{g}'([\mathbf{w}^i, \boldsymbol{\gamma}^i] - [\mathbf{w}^{i+1}, \boldsymbol{\gamma}^{i+1}]) > \rho\ tol\ \|([\mathbf{w}^i, \boldsymbol{\gamma}^i] - [\mathbf{w}^{i+1}, \boldsymbol{\gamma}^{i+1}])\| > \eta > 0. \tag{B.7}$$

This ensures that each stability center update yields a significant (at least η) decrease in the objective f. This coupled with the assumption that f has bounded level sets and τ^i is bounded for all i guarantees that in a finite number of steps the algorithm will find a critical point. □

References

Bazaraa, M. S., Sherali, H. D. & Shetty, C. M. (2006). *Nonlinear programming theory and algorithms*. New York: Wiley.

Bennett, K. P., Hu, J., Ji, X., Kunapuli, G., & Pang, J. S. (2006). Model selection via bilevel optimization. In *International joint conference on neural networks* (pp. 1922–1929).

Bennett, K. P., Kunapuli, G., Hu, J., & Pang, J. S. (2008). Bilevel optimization and machine learning. In J. Zurada et al. (Eds.), *Lecture notes in computer science: Vol. 5050. Computational intelligence: research frontiers: IEEE WCCI 2008*, Hong Kong, China, 1–6 June, 2008: Plenary/invited Lectures (pp. 25–47). New York: Springer.

Bergeron, C., Moore, G., Krein, M., Breneman, C. M., & Bennett, K. P. (2011a). Exploiting domain knowledge for improved quantitative high-throughput screening curve fitting. Journal of Chemical Information and Modeling

Bergeron, C., Moore, G., Zaretzki, J., Breneman, C. M. & Bennett, K. P. (2011b). Fast bundle algorithm for multiple instance learning. IEEE Transactions on Pattern Analysis and Machine Intelligence

Bracken, J., & McGill, J. (1973). Mathematical programs with optimization problems in the constraints. *Operations Research*, *21*, 37–44.

Breneman, C., & Rhem, M. (1997). A QSPR analysis of hplc column capacity factors for a set of high-energy materials using electronic van der Waals surface property descriptors computed by the transferable atom equivalent method. *Journal of Computational Chemistry*, *18*(2), 182–197.

Chapelle, O., Vapnik, V., Bousquet, O., & Mukherjee, S. (2002). Choosing multiple parameters for support vector machines. *Machine Learning*, *46*(1–3), 131–159.

Clarke, F. H. (1990). *Optimization and nonsmooth analysis*. New York: Wiley.

Colson, B., Marcotte, P., & Savard, G. (2007). An overview of bilevel programming. *Annals of Operation Research*, *153*, 235–256.

Dempe, S. (2003). Annotated bibliography on bilevel programming and mathematical programs with equilibrium constraints. *Optimization*, *52*, 333–359.

Do, C., Foo, C. S., & Ng, A. (2008). Efficient multiple hyperparameter learning for log-linear models. In *Advances in neural information processing systems 20* (pp. 377–384).

Eitrich, T., & Lang, B. (2006). Efficient optimization of support vector machine learning parameters for unbalanced datasets. *Journal of Computational and Applied Mathematics*, *196*(2), 425–436.

Golub, G. H., Heath, M., & Wahba, G. (1979). Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, *21*(2), 215–223.

Joachims, T. (2006). Training linear SVMs in linear time. In *Knowledge discovery and data mining* (pp. 217–226).

Keerthi, S., Sindhwani, V., & Chapelle, O. (2006). An efficient method for gradient-based adaptation of hyperparameters in svm models. In *Neural information processing systems 18* (pp. 673–680).

Kunapuli, G., Bennett, K. P., & Pang, J. S. (2008). Bilevel model selection for support vector machines. In P. Pardalos & P. Hansen (Eds.), *AMS: Vol. 45. Data mining and mathematical programming* (pp. 129–158).

Mangasarian, O. L. (1994). *Nonlinear programming*. New York: McGraw-Hill.

Momma, M., & Bennett, K. P. (2002). A pattern search method for model selection of support vector regression. In *Proceedings of the second SIAM international conference on data mining*.

Moore, G., Bergeron, C., & Bennett, K. P. (2009). Nonconvex bilevel programming for hyperparameter selection. In *Proceedings of the 2009 IEEE international conference on data mining workshops* (pp. 374–381).

Noll, D., Prot, O., & Rondepierre, A. (2008). A proximity control algorithm to minimize nonsmooth and nonconvex functions. *Pacific Journal of Optimization*, *4*, 571–604.

Pang, J. S., & Sun, D. (2008). First-order sensitivity of linearly constrained strongly monotone composite variational inequalities. Technical report.

Ruszczyński, A. (2006). *Nonlinear optimization*. Princeton: Princeton University Press.

Seeger, M. (1999). Bayesian model selection for support vector machines, Gaussian processes and other kernel classifiers. In *Advances in neural information processing systems* (pp. 603–609).

Skajaa, A. (2010). Limited memory bfgs for nonsmooth optimization. *Master's thesis*, Courant Institute of Mathematical Science, New York University.

Suykens, J., Brabanter, J., & Gestel, T. V. (2002). *Least squares support vector machines*. Singapore: World Scientific.

Teo, C. H., Le, Q. V., Smola, A., & Vishwanathan, S. V. N. (2007). A scalable modular convex solver for regularized risk minimization. In *Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 727–736).