Descent Methods for Tuning Parameter Refinement

Alexander Lorbert
Dept. of Electrical Engineering
Princeton University

Peter J. Ramadge
Dept. of Electrical Engineering
Princeton University

Abstract

This paper addresses multidimensional tuning parameter selection in the context of "train-validate-test" and K-fold cross validation. A coarse grid search over tuning parameter space is used to initialize a descent method which then jointly optimizes over variables and tuning parameters. We study four regularized regression methods and develop the update equations for the corresponding descent algorithms. Experiments on both simulated and real-world datasets show that the method results in significant tuning parameter refinement.

1 INTRODUCTION

We consider the problem of fitting a model from given data consisting of instances and corresponding responses (or labels). Our goal is to minimize the loss in mapping a general instance to a response. The standard approach to avoid over-fitting the data is to incorporate regularization. For example, in linear regression we are given a design matrix $\mathbf{X} \in \mathbb{R}^{n\times p}$ with corresponding response vector $\mathbf{y} \in \mathbb{R}^n$ and we seek a vector, $\boldsymbol{\beta} \in \mathbb{R}^p$, that minimizes

$$(1/n)\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda J(\boldsymbol{\beta}) \ . \tag{1}$$

The first term is the average loss and the second term regularizes $\boldsymbol{\beta}$ via the function $J(\cdot)$ and the nonnegative tuning parameter λ. In general, by choice of J and for fixed λ, minimizing (1) is a convex problem. For example, in ridge regression (Hoerl & Kennard 1970b,a), $J(\boldsymbol{\beta}) = \|\boldsymbol{\beta}\|_2^2$ and for the Lasso (Tibshirani 1996), $J(\boldsymbol{\beta}) = \|\boldsymbol{\beta}\|_1^1$. Of greater interest here is a vector of tuning parameters, $\boldsymbol{\lambda}$, associated with an array of regularization functions. For example, in Elastic Net (EN) (Zou & Hastie 2005) regularization takes the form $J(\boldsymbol{\beta}) = \lambda_1\|\boldsymbol{\beta}\|_1^1 + \lambda_2\|\boldsymbol{\beta}\|_2^2$.

Much attention has been given to the minimization of (1) for fixed λ (e.g., (Hastie et al. 2009)). However, the real problem of interest is to jointly select optimal $\boldsymbol{\lambda}$ and $\boldsymbol{\beta}$. Unfortunately, this problem is usually not convex, making it difficult to solve because of the presence of local minima. In some special cases the optimal value of the tuning parameter $\boldsymbol{\lambda}$ can be characterized in terms of an implicit equation. For example, generalized cross validation (GCV) for ridge regression provides a closed-form function to minimize, thereby producing the optimal $\boldsymbol{\lambda}$ (Golub et al. 1979). Wood (2000) extended this result for generalized ridge regression and Fu (2005) proposed a "quasi-GCV" approach for nonlinear estimators. However, these are special cases. Obtaining closed form, or implicit equations for the optimal $\boldsymbol{\lambda}$ is extremely difficult for general regularization.

A prevalent approach to selecting a suitable $\boldsymbol{\lambda}$ is to generate a set of $\boldsymbol{\beta}$'s for various values of $\boldsymbol{\lambda}$ using a training set and then pick the optimal $\boldsymbol{\beta}$ by assessing loss over a validation set (for other methods see (Hastie et al. 2009)). This is the basic idea behind the "train-validate-test" approach and its natural extension to K-fold cross validation (CV). In the simplest case, we partition the data into three subsets, denoted the training, validation, and testing sets. The training set is used to generate model parameters ($\boldsymbol{\beta}$) under fixed tuning parameters ($\boldsymbol{\lambda}$). Using these model parameters we compute the average loss over the validation set without regularization. Iterating in a coordinated manner over values of tuning parameters allows us to select an estimate for the optimal $\boldsymbol{\beta}$ and $\boldsymbol{\lambda}$. The interaction between testing and validating is analogous to a system with feedback. The training set produces candidate $\boldsymbol{\beta}$'s while the validation set (implicitly) produces candidate $\boldsymbol{\lambda}$'s. Finally, we use the testing set to produce an estimate of the expected loss. The testing set is "off limits" during the process of estimating the

Appearing in Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS) 2010, Chia Laguna Resort, Sardinia, Italy. Volume 9 of JMLR: W&CP 9. Copyright 2010 by the authors.

optimal $\boldsymbol{\beta}$ and $\boldsymbol{\lambda}$. Once the testing set is used, changes to $\boldsymbol{\beta}$ and $\boldsymbol{\lambda}$ are not allowed.

Recently, coordinate descent (CD) algorithms have been shown to yield fast estimates of the optimal $\boldsymbol{\beta}$ with $\boldsymbol{\lambda}$ fixed (Friedman et al. 2007). In a regression setting, coordinate descent has the advantage of not requiring complicated matrix inversions or eigendecompositions, making it computationally efficient when there are a large number of features. Essentially, the algorithm cyclically updates the elements of $\boldsymbol{\beta}$ until convergence (within a tolerance) is reached. As currently employed, this technique naturally resides in the training stage due to the fixed $\boldsymbol{\lambda}$.

The main objective of this paper is to explore the use of similar descent methods to refine a candidate set of tuning parameters that yield minimal loss. To this end, for several standard regression problems we design descent algorithms for tuning parameters based on both train-validate (TV) and K-fold cross validation. The approach allows the selection of multiple tuning parameters simultaneously. Although the problem is not convex, we posit that the ability to rapidly compute a local minimum can improve existing grid search methods. A coarse grid search is used to initialize the descent algorithm. When the dimension of $\boldsymbol{\lambda}$ is small, a coarse grid search is fast and can usually find a promising initial value for the descent method.

The motivation for our study stems from the selection of EN tuning parameters. When $\boldsymbol{\lambda}$ is a scalar, computing the optimal ($\boldsymbol{\beta}$, $\boldsymbol{\lambda}$) involves a line search and this does not impose a great difficulty. In contrast, when $\boldsymbol{\lambda}$ is two-dimensional the problem becomes more difficult. For EN, LARS-EN (Zou & Hastie 2005) may be used as follows: fix λ_2, find optimal λ_1 with LARS (Efron et al. 2004), iterate over λ_2. Alternatively, one can consider various ratios of λ_1/λ_2 and solve for the correct scaling. Ultimately, however, a $\boldsymbol{\lambda}$ found with these techniques may not be the global minimum or even a local one. Moreover, if the size of $\boldsymbol{\lambda}$ is greater than 2, these methods break down. For example, consider:

$$(1/n)\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \textstyle\sum_{k=1}^{p} \lambda_k \beta_k^2 \ . \tag{2}$$

To find the optimal tuning parameters for this problem by exhaustive grid search is a costly computation. Moreover, the addition of ℓ_1 penalties would add to the computational burden. However, a coarse grid search followed by our descent algorithm readily yields an acceptable set of tuning parameter values.

The remainder of the paper is organized as follows: §2 formulates the general problem of tuning parameter selection. In §3 four cases of penalized regression are analyzed for the TV technique, which naturally extends itself to K-fold cross validation, described in §3.5. §4 presents results from experiments involving a simulated dataset and 3 real-world datasets.

2 JOINT DESCENT ON $(\boldsymbol{\beta}, \boldsymbol{\lambda})$

Throughout the paper, a bold symbol refers to a column vector or a matrix, e.g., $\mathbf{x} = [x_1, \ldots, x_n]^T$ and $\mathbf{X} = [\mathbf{X}_1, \ldots, \mathbf{X}_p] = [X_{ij}]$ with $\mathbf{X}_j \in \mathbb{R}^n$. Let $\mathcal{X}$ and $\mathcal{Y}$ denote the instance space and feature space, respectively. We are given a data set, $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, with $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. Without loss of generality, we partition $\mathcal{D}$ into a training set, $\mathcal{U} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n_u}$, and a validation set, $\mathcal{V} = \{(\mathbf{x}_i, y_i)\}_{i=n_u+1}^{n}$, with $|\mathcal{V}| = n_v = n - n_u$.

Given a loss function, $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ we seek to minimize the empirical loss

$$L_u(\boldsymbol{\beta}) = (1/n_u) \sum_{\mathcal{U}} L(f(\mathbf{x}_i; \boldsymbol{\beta}), y_i) \tag{3}$$

where the function f belongs to a function space $\mathcal{F}$ parameterized by $\boldsymbol{\beta} \in \mathbb{R}^{p_b}$. $L_u : \mathbb{R}^{p_b} \to \mathbb{R}$ evaluates the average loss over the training set for a fixed $\boldsymbol{\beta}$. To add regularization in the training stage, bring in the regularization penalty

$$\boldsymbol{\lambda}^T \mathbf{J}(\boldsymbol{\beta}) = \textstyle\sum_{k=1}^{p_\ell} \lambda_k J_k(\boldsymbol{\beta}) \tag{4}$$

where the $J_k(\cdot)$ are given regularization functions, $\boldsymbol{\lambda} \in \mathbb{R}^{p_\ell}$ is a vector of tuning parameters and $\mathbf{J}(\boldsymbol{\beta}) = [J_1(\boldsymbol{\beta}), \ldots, J_{p_\ell}(\boldsymbol{\beta})]^T$. The unknowns in (3) and (4) are the optimal $\boldsymbol{\beta}$ and $\boldsymbol{\lambda}$.

Given $\boldsymbol{\lambda}$, one can find the optimal $\boldsymbol{\beta}$ over the training set. This defines a function $h_u : \mathbb{R}^{p_\ell} \to \mathbb{R}^{p_b}$ that maps $\boldsymbol{\lambda}$ to the optimal $\boldsymbol{\beta}$ over the training set:

$$h_u(\boldsymbol{\lambda}) = \arg\min\nolimits_{\boldsymbol{\beta}} L_u(\boldsymbol{\beta}) + \boldsymbol{\lambda}^T \mathbf{J}(\boldsymbol{\beta}). \tag{5}$$

Once $\boldsymbol{\beta}$ is obtained, generalization performance can be evaluated on the validation set. This defines a function $L_v : \mathbb{R}^{p_b} \to \mathbb{R}$:

$$L_v(\boldsymbol{\beta}) = (1/n_v) \sum_{\mathcal{V}} L(f(\mathbf{x}_i; \boldsymbol{\beta}), y_i) \tag{6}$$

Additionally, for some forms of regularization it is convenient to have a calibration function $h_c : \mathbb{R}^{p_b} \to \mathbb{R}^{p_b}$. Before evaluating the loss over the validation set, the $\boldsymbol{\beta}$ obtained is calibrated according to $h_c(\cdot)$. The function $h_c(\cdot)$ is assumed to be given a priori and in many situations we have $h_c(\boldsymbol{\beta}) = \boldsymbol{\beta}$.

In this framework, tuning parameter selection involves finding:

$$\boldsymbol{\lambda}_{opt} = \arg\min\nolimits_{\boldsymbol{\lambda}} (L_v \circ h_c \circ h_u)(\boldsymbol{\lambda}). \tag{7}$$

Let $\mathcal{Z}$ denote the domain of $\boldsymbol{\lambda}$. The conventional validation process is given by the following routine: We fix a $\boldsymbol{\lambda} \in \mathcal{Z}$ and find $\hat{\boldsymbol{\beta}} = h_u(\boldsymbol{\lambda})$. We then calibrate by setting $\boldsymbol{\beta}^* = h_c(\hat{\boldsymbol{\beta}})$. Finally, we evaluate $L_v(\boldsymbol{\beta}^*)$. This process is then repeated for a new $\boldsymbol{\lambda} \in \mathcal{Z}$ and repetition continues until a stopping criterion is met. We then set the estimated $\boldsymbol{\lambda}_{opt}$ based on the minimum $L_v(\boldsymbol{\beta}^*)$ found. In this approach, to find $\boldsymbol{\lambda}_{opt}$ requires evaluating $h_u(\boldsymbol{\lambda})$ for each candidate $\boldsymbol{\lambda}$. Primarily, this approach is followed because $h_u(\boldsymbol{\lambda})$ is generally a convex optimization problem, whereas (7) is not. Thus, we partition (7) via $h_u(\cdot)$, $h_c(\cdot)$, and $L_v(\cdot)$, to find $\boldsymbol{\lambda}_{opt}$.

Even though (7) is a non-convex minimization problem, we may ask if local minima are difficult to compute. If we can find local minima in an efficient manner, then seeking a set of local minima may yield the global minimum or a local minimum that is adequate for the accuracy desired.

In (Friedman et al. 2007) an elegant coordinate descent method is given for several popular estimators. In these schemes each coordinate β_i is updated cyclically. As in all coordinate descent schemes, after iterating the updates, the β_i may converge to the optimal solution. Arguably, the primary benefit of this method is the absence of matrix inversions and eigendecompositions in inner loops of the algorithm.

Our proposal is to use a similar cyclical coordinate descent method to optimize $\boldsymbol{\beta}$ and perform the validation process to find $\boldsymbol{\lambda}$ directly in one iteration: after updating all the β_i we update the λ_i in an attempt to tune $\boldsymbol{\lambda}$ as the optimization of $\boldsymbol{\beta}$ proceeds. The main advantage of the proposal is its potential efficiency. We can update the λ_i cyclically using coordinate descent (update($\lambda_1|\cdot$)$\rightarrow$update($\lambda_2|\cdot$)$\rightarrow \ldots$), or all at once using the gradient (update($\boldsymbol{\lambda}|\cdot$)). This general CD with validation (CD-V) is given by Algorithm 1. Rather than computing $\hat{\boldsymbol{\beta}}$ for many fixed $\boldsymbol{\lambda}$, we attempt to find a local minimum of $(L_v \circ h_c \circ h_u)(\boldsymbol{\lambda})$ using a descent method starting from a good initial guess. If this can be done efficiently, then computing a local minimum from a promising initialization may allow us to find an adequate minimum of (7), with the optimal $\boldsymbol{\beta}$ also provided. Related to gradient descent, we can implement a rudimentary forward line search by leveraging warm starts from prior, nearby $\boldsymbol{\beta}$'s. Then, using cubic interpolation from 3 points and a gradient, we can obtain the next $\boldsymbol{\lambda}$ (Dennis & Schnabel 1996). Alternatively, we need not rely on the gradient for a search direction. For $\boldsymbol{\lambda} \in \mathbb{R}^{p_\ell}$ we can use a simplex method, which also leverages warm starts (Nelder & Mead 1965).

The β_i-update equations are derived from the KKT equations associated with $h_u(\cdot)$:

Algorithm 1 CD-V

```
1: Initialize variables
2: while All Stopping Criteria == False do
3:   for $j = 1$ to $burn_{in}$ do
4:     for $i = 1$ to $p_b$ do
5:       update($\beta_i|\mathcal{U}, \boldsymbol{\beta}, \boldsymbol{\lambda}$)
6:     end for
7:   end for
8:   for $i = 1$ to $p_\ell$ do
9:     update($\lambda_i|\mathcal{V}, \boldsymbol{\beta}, \boldsymbol{\lambda}$)
10:  end for
11:  {or update($\boldsymbol{\lambda}|\mathcal{V}, \boldsymbol{\beta}, \boldsymbol{\lambda}$)}
12: end while
```

Algorithm 2 update($\lambda_i|\mathbf{D}_{\lambda_i} L_v$)

```
1: if $\delta_i \times \mathbf{D}_{\delta_i} L_v < 0$ then
2:   $\log \lambda_i \leftarrow \log \lambda_i + \delta_i$
3: else if $\delta_i \times \mathbf{D}_{\delta_i} L_v > 0$ then
4:   $\delta_i \leftarrow -\delta_i \times \delta$
5:   $\log \lambda_i \leftarrow \log \lambda_i + \delta_i$
6: end if
```

$$\mathbf{D}_{\boldsymbol{\beta}} h_u = \mathbf{D}_{\boldsymbol{\beta}} L_u + \boldsymbol{\lambda}^T \mathbf{D}_{\boldsymbol{\beta}} \mathbf{J} = 0. \tag{8}$$

When possible, we express β_i in terms of the other elements of $\boldsymbol{\beta}$ (denoted $\boldsymbol{\beta}_{-i} \in \mathbb{R}^{p_b - 1}$) and $\boldsymbol{\lambda}$. Then, to update β_i we use the current $\boldsymbol{\beta}_{-i}$ and $\boldsymbol{\lambda}$. Initially, we set $\boldsymbol{\beta} = \mathbf{0}$, which encourages sparse solutions. For updating λ_i we use the gradient, $\nabla_{\boldsymbol{\lambda}} L_v$, evaluated from:

$$\mathbf{D}_{\boldsymbol{\lambda}} L_v = \mathbf{D}_{\boldsymbol{\beta}^*} L_v \times \mathbf{D}_{\hat{\boldsymbol{\beta}}} \boldsymbol{\beta}^* \times \mathbf{D}_{\boldsymbol{\lambda}} \hat{\boldsymbol{\beta}}. \tag{9}$$

We decrease λ_i when $\mathbf{D}_{\lambda_i} L_v$ is positive and increase it when the derivative is negative. In general, the λ_i are restricted to the nonnegative reals so we increase or decrease $\log \lambda_i$ by a variable step size. We can update λ_i based on the sign of $\mathbf{D}_{\lambda_i} L_v$ (coordinate descent) or based on the direction of the gradient $\nabla_{\boldsymbol{\lambda}} L_v$. Empirically, we have found that L_v consists of many almost-flat regions, so using the direction of the gradient, rather than the gradient itself, is sufficient. Similar to a line search, for coordinate descent we initialize $p_\ell + 1$ scalar parameters, denoted $\delta, \delta_1, \ldots, \delta_{p_\ell}$. With δ selected from the interval $[0.5, 1)$, we choose initial values for δ_i and our λ_i-update function is given by Algorithm 2. The **if** statement tests if we are heading toward a local minimum, indicated by opposite signs or a negative product. Conversely, the **else if** clause tests if we are moving away from a local minimum, in which case we scale δ_i by shrinking it and inverting its sign. For gradient descent, we normalize the gradient of L_v with respect to the log of the tuning parameters and update accordingly by standard methods.

We use standard stopping criteria. First, we consider the main loop having $iter_{max}$ iterations, typically in the 1K-10K range. We analyze consecutive values of $\boldsymbol{\beta}$ and $\boldsymbol{\lambda}$ to assess convergence. Convergence of $\boldsymbol{\lambda}$ is not as straight-forward because $\log \lambda_k$ may tend to $\pm\infty$. Therefore, we look at consecutive values of $\boldsymbol{\tau}$ where $\tau_k = \text{sgn}(\log \lambda_k) \min(C, |\log \lambda_k|)$ and C is selected to be in a range corresponding to large $|\log \lambda_k|$ (e.g., 50).

In updating $\boldsymbol{\lambda}$ we make use of $\mathbf{D}_{\boldsymbol{\lambda}}\hat{\boldsymbol{\beta}}$. This assumes that we have the actual $\hat{\boldsymbol{\beta}}$ one would attain at convergence. If we were to simply update the β_i *once* and then evaluate this derivative we would obtain a noisy estimate. Therefore, we "burn in" $\boldsymbol{\beta}$ after updating $\boldsymbol{\lambda}$. In practice, we set $burn_{in}$ to be in the tens range.

3 PENALIZED REGRESSION

We now specialize these ideas to specific forms of the regression problem. We assume that we are given data in the form of a full rank design matrix, $\mathbf{X} \in \mathbb{R}^{n \times p}$, and a response vector, $\mathbf{y} \in \mathbb{R}^n$. We partition our data as follows:

$$\mathbf{X} = \begin{pmatrix} \mathbf{X}_u \\ \mathbf{X}_v \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} \mathbf{y}_u \\ \mathbf{y}_v \end{pmatrix} \tag{10}$$

where $\mathbf{X}_u \in \mathbb{R}^{n_u \times p}$ and $\mathbf{y}_u \in \mathbb{R}^{n_u}$ correspond to training data and $\mathbf{X}_v \in \mathbb{R}^{n_v \times p}$ and $\mathbf{y}_v \in \mathbb{R}^{n_v}$ correspond to validation data. When isolating any particular subset of the data we center the sub-design matrix and corresponding sub-response vector. Our function space, $\mathcal{F}$, is the set of linear functions parameterized by $\boldsymbol{\beta} \in \mathbb{R}^p$, i.e., $p_b = p$ and $f(\mathbf{x}; \boldsymbol{\beta}) = \mathbf{x}^T\boldsymbol{\beta}$. Our loss function is $L(y_1, y_2) = (y_1 - y_2)^2$. Putting these components together yields

$$\begin{aligned} L_u(\boldsymbol{\beta}) &= (1/n_u)(\mathbf{y}_u - \mathbf{X}_u\boldsymbol{\beta})^T(\mathbf{y}_u - \mathbf{X}_u\boldsymbol{\beta}) && (11) \\ &= \boldsymbol{\beta}^T R \boldsymbol{\beta} - 2r^T\boldsymbol{\beta} + \mathbf{y}_u^T\mathbf{y}_u/n_u && (12) \\ L_v(\boldsymbol{\beta}) &= (1/n_v)(\mathbf{y}_v - \mathbf{X}_v\boldsymbol{\beta})^T(\mathbf{y}_v - \mathbf{X}_v\boldsymbol{\beta}) && (13) \\ &= \boldsymbol{\beta}^T G \boldsymbol{\beta} - 2g^T\boldsymbol{\beta} + \mathbf{y}_v^T\mathbf{y}_v/n_v && (14) \end{aligned}$$

where $\mathbf{R} = \mathbf{X}_u^T\mathbf{X}_u/n_u$, $\mathbf{r} = \mathbf{X}_u^T\mathbf{y}_u/n_u$, $\mathbf{G} = \mathbf{X}_v^T\mathbf{X}_v/n_v$, and $\mathbf{g} = \mathbf{X}_v^T\mathbf{y}_v/n_v$. It follows that

$$\begin{aligned} \mathbf{D}_{\boldsymbol{\beta}}L_u &= 2\boldsymbol{\beta}^T\mathbf{R} - 2\mathbf{r}^T && (15) \\ \mathbf{D}_{\boldsymbol{\beta}}L_v &= 2\boldsymbol{\beta}^T\mathbf{G} - 2\mathbf{g}^T. && (16) \end{aligned}$$

3.1 RIDGE

In ridge regression $p_\ell = 1$, $J_1(\boldsymbol{\beta}) = \boldsymbol{\beta}^T\boldsymbol{\beta}$, and $\boldsymbol{\beta}^* = h_c(\boldsymbol{\beta}) = \boldsymbol{\beta}$. Therefore, $\mathbf{D}_{\boldsymbol{\beta}}\mathbf{J}(\boldsymbol{\beta}) = 2\lambda_1\boldsymbol{\beta}^T$ and $\mathbf{D}_{\boldsymbol{\beta}}\boldsymbol{\beta}^* = \mathbf{I}$. For updating β_i we use (8), which yields

$$\begin{aligned} 0 &= 2\boldsymbol{\beta}^T\mathbf{R} - 2\mathbf{r}^T + 2\lambda_1\boldsymbol{\beta}^T && (17) \\ \boldsymbol{\beta} &= (\mathbf{R} + \lambda_1\mathbf{I})^{-1}\mathbf{r} && (18) \\ \beta_i &\leftarrow (R_{ii}\beta_i - \mathbf{R}_i^T\boldsymbol{\beta} + r_i)/(R_{ii} + \lambda_1)\ . && (19) \end{aligned}$$

Algorithm 3 Ridge CD-V

```
 1: Initialize variables
 2: Compute SVD $n_u^{-1/2}\mathbf{X} = \mathbf{U\Sigma V}^T$.
 3: while All Stopping Criteria == False do
 4:   for $j = 1$ to $burn_{in}$ do
 5:     for $i = 1$ to $p$ do
 6:       $\beta_i \leftarrow (R_{ii}\beta_i - \mathbf{R}_i^T\boldsymbol{\beta} + r_i)/(R_{ii} + \lambda_1)$
 7:     end for
 8:   end for
 9:   $\mathbf{D}_{\lambda_1}L_v \leftarrow -2(\boldsymbol{\beta}^T\mathbf{G} - \mathbf{g}^T)\mathbf{V}(\boldsymbol{\Sigma}^2 + \lambda_1\mathbf{I})^{-2}\mathbf{V}^T\mathbf{r}$
10:   update($\lambda_1 | \mathbf{D}_{\lambda_1}L_v$)
11: end while
```

Although β_i appears on the right hand side of the update, this is for ease of representation. The numerator contains a "$+R_{ii}\beta_i$" and a "$-R_{ii}\beta_i$". Assuming convergence yields the fixed point equations

$$\begin{aligned} \boldsymbol{\beta} &= (\boldsymbol{\Lambda} + \lambda_1\mathbf{I})^{-1}(\boldsymbol{\Lambda\beta} - \mathbf{R}\boldsymbol{\beta} + \mathbf{r}) && (20) \\ &= (\boldsymbol{\Lambda} + \lambda_1\mathbf{I})^{-1}(\mathbf{M}\boldsymbol{\beta} + \mathbf{r}) \end{aligned}$$

where $\boldsymbol{\Lambda} = \text{diag}\{R_{11}, \ldots, R_{pp}\}$ and $\mathbf{M} = \boldsymbol{\Lambda} - \mathbf{R}$. We can use (18) or (20) to find $\mathbf{D}_{\lambda_1}\boldsymbol{\beta}$. Here, we use (18) and obtain

$$\mathbf{D}_{\lambda_1}\boldsymbol{\beta} = -(\mathbf{R} + \lambda_1\mathbf{I})^{-2}\mathbf{r} = -(\mathbf{R} + \lambda_1\mathbf{I})^{-1}\beta \tag{21}$$

The ridge case has the nice property of allowing us to compute $\mathbf{D}_{\boldsymbol{\lambda}}\boldsymbol{\beta}$ directly from the KKT equation. Let $n_u^{-1/2}\mathbf{X}$ have SVD $\mathbf{U\Sigma V}^T$. Then $\mathbf{R} = \mathbf{V\Sigma}^2\mathbf{V}^T$ and

$$\mathbf{D}_{\lambda_1}\boldsymbol{\beta} = -\mathbf{V}(\boldsymbol{\Sigma}^2 + \lambda_1\mathbf{I})^{-2}\mathbf{V}^T\mathbf{r}. \tag{22}$$

This requires one SVD but saves considerable computation time. Putting the above together, the CD-V method for ridge regression is given in Algorithm 3.

3.2 DIAGONAL TIKHONOV

We now look at a more interesting problem. Along the lines of ridge regression, we set $p_\ell = p$, $J_k(\boldsymbol{\beta}) = \beta_k^2$, and $h_c(\boldsymbol{\beta}) = \boldsymbol{\beta}$. It follows that $\boldsymbol{\lambda}^T\mathbf{J}(\boldsymbol{\beta}) = \boldsymbol{\beta}^T\boldsymbol{\Gamma\beta}$ where $\boldsymbol{\Gamma} = \text{diag}(\boldsymbol{\lambda})$. The regularization term, $\boldsymbol{\beta}^T\boldsymbol{\Gamma\beta}$, is a special case of Tikhonov regression where $\boldsymbol{\Gamma}$ is diagonal. Whereas ridge assigns a global penalty over β_i, i.e., $\lambda_1\|\boldsymbol{\beta}\|_2^2$, Diagonal Tikhonov Regression (DTR) regularizes each β_i individually. Consequently, the issue of sparsity comes into play. A λ_k of approximately zero implies that β_k is best determined by OLS, whereas a very large λ_k would result in $\beta_k = 0$.

Using the conventional approach for finding $\boldsymbol{\lambda}_{opt}$ is simply not practical. Without prior knowledge, if we restrict λ_k to be one of m values then we would need to search over a set of m^p elements. Even for small values of m and p, computation time would be enormous.

Algorithm 4 DTR CD-V

```
 1: Initialize variables
 2: Set $\mathbf{W} = (\mathbf{R} + \operatorname{diag} \boldsymbol{\lambda}_0)^{-1}$
 3: while All Stopping Criteria == False do
 4:    for $j = 1$ to $burn_{in}$ do
 5:       for $i = 1$ to $p$ do
 6:          $\beta_i \leftarrow (R_{ii}\beta - \mathbf{R}_i^T\boldsymbol{\beta} + r_i)/(R_{ii} + \lambda_i)$
 7:       end for
 8:    end for
 9:    for $k = 1$ to $p$ do
10:       $\mathbf{D}_{\lambda_k} L_v \leftarrow -2(\boldsymbol{\beta}^T\mathbf{G} - \mathbf{g}^T)(\beta_k \mathbf{W}_k)$
11:       update$(\lambda_k | \mathbf{D}_{\lambda_k} L_v)$
12:       $\gamma_k = \lambda_k - \lambda_k^{prev}$
13:       $\mathbf{W} \leftarrow \mathbf{W} - \frac{\gamma_k}{1+\gamma_k W_{kk}} \mathbf{W}_k \mathbf{W}_k^T$
14:    end for
15: end while
```

This case is covered by Wood (2000) using Newton's method for GCV. We posit that our joint coordinated search method is well suited for this problem.

We proceed as before: the update equations for β_i are given by

$$\beta_i \leftarrow (R_{ii}\beta - \mathbf{R}_i^T\boldsymbol{\beta} + r_i)/(R_{ii} + \lambda_i). \tag{23}$$

Similarly, we obtain

$$\mathbf{D}_{\lambda_k}\boldsymbol{\beta} = -\beta_k(\mathbf{R} + \operatorname{diag}(\boldsymbol{\lambda}))^{-1}\mathbf{e}_k \tag{24}$$

where $\mathbf{e}_k$ is the k-th standard basis vector. Starting with an initial positive $\boldsymbol{\lambda}_0$, we compute $\mathbf{W} = (\mathbf{R} + \operatorname{diag}(\boldsymbol{\lambda}_0))^{-1}$ (the inverse exists). From here all future updates can be done without inversion via the update

$$\mathbf{W} \leftarrow (\mathbf{W} + \gamma_k e_k e_k^T)^{-1} \tag{25}$$

$$(\mathbf{W} + \gamma_k e_k e_k^T)^{-1} = \mathbf{W} - \frac{\gamma_k}{1 + \gamma_k W_{kk}} \mathbf{W}_k \mathbf{W}_k^T \tag{26}$$

where γ_k is the change in λ_k. The above is derived from the matrix inversion lemma. Here, we see that cycling through $\boldsymbol{\lambda}$ and performing one-at-a-time updates facilitates the expression (25). The CD-V for DTR is given by Algorithm 4.

3.3 LASSO

The Lasso regression problem has $p_\ell = 1$ with $J_1(\boldsymbol{\beta}) = \|\boldsymbol{\beta}\|_1^1$ and $h_c(\boldsymbol{\beta}) = \boldsymbol{\beta}$. Let $\langle \mathbf{x} \rangle$ denote the vector-valued hinge function of the vector $\mathbf{x}$, i.e., $\langle \mathbf{x} \rangle_i = x_i \mathbb{1}\{x_i > 0\}$. A sub-gradient of $\langle \mathbf{x} \rangle$ is given by the vector-valued function, $[\![\mathbf{x}]\!]$, with $[\![\mathbf{x}]\!]_i = \mathbb{1}\{x_i > 0\}$. Using the hinge function, we define the soft-thresholding function as

$$S(\mathbf{x}, \mathbf{b}) = \langle \mathbf{x} - \mathbf{b} \rangle - \langle -\mathbf{x} - \mathbf{b} \rangle \tag{27}$$

Algorithm 5 Lasso CD-V

```
 1: Initialize variables
 2: while All Stopping Criteria == False do
 3:    for $j = 1$ to $burn_{in}$ do
 4:       for $i = 1$ to $p$ do
 5:          $\beta_i \leftarrow S(R_{ii}\beta_i - \mathbf{R}_i^T\boldsymbol{\beta} + r_i, (\lambda_1/2)\mathbf{1})/R_{ii}$
 6:       end for
 7:    end for
 8:    $\mathbf{a}_{1/2} \leftarrow [\![\pm\mathbf{M}\boldsymbol{\beta} \pm \mathbf{r} - (\lambda_1/2)\mathbf{1}]\!]$
 9:    $\mathbf{A} \leftarrow \operatorname{diag}(\mathbf{a}_1 + \mathbf{a}_2)$
10:    $\mathbf{D}_{\lambda_1}\boldsymbol{\beta} \leftarrow \frac{1}{2}((\mathbf{I} - \mathbf{A})\boldsymbol{\Lambda} + \mathbf{A}\mathbf{R})^\dagger(\mathbf{a}_2 - \mathbf{a}_1)$
11:    $\mathbf{D}_{\lambda_1} L_v \leftarrow 2(\boldsymbol{\beta}^T\mathbf{G} - \mathbf{g}^T)\mathbf{D}_{\lambda_1}\boldsymbol{\beta}$
12:    update$(\lambda_1 | \mathbf{D}_{\lambda_1} L_v)$
13: end while
```

for vector $\mathbf{x}$ and nonnegative vector $\mathbf{b}$. The KKT and update equations for Lasso are given by

$$\mathbf{D}_{\beta_i} L_u = 2\boldsymbol{\beta}^T\mathbf{R}_i - 2\mathbf{r}_i + \lambda_1 \operatorname{sgn} \beta_i = 0 \tag{28}$$

$$\beta_i \leftarrow S(R_{ii}\beta_i - \mathbf{R}_i^T\boldsymbol{\beta} + r_i, (\lambda_1/2)\mathbf{1})/R_{ii} \ . \tag{29}$$

At convergence, (29) can be written as

$$\begin{aligned} \boldsymbol{\beta} &= \boldsymbol{\Lambda}^{-1} S\left(\mathbf{M}\boldsymbol{\beta} + \mathbf{r}, (\lambda_1/2)\mathbf{1}\right) \\ &= \boldsymbol{\Lambda}^{-1}\langle \mathbf{M}\boldsymbol{\beta} + \mathbf{r} - (\lambda_1/2)\mathbf{1} \rangle \\ &\quad - \boldsymbol{\Lambda}^{-1}\langle -\mathbf{M}\boldsymbol{\beta} - \mathbf{r} - (\lambda_1/2)\mathbf{1} \rangle \end{aligned} \tag{30}$$

From (30) we obtain

$$((\mathbf{I} - \mathbf{A})\boldsymbol{\Lambda} + \mathbf{A}\mathbf{R})\mathbf{D}_{\lambda_1}\boldsymbol{\beta} = (\mathbf{a}_2 - \mathbf{a}_1)/2 \tag{31}$$

$$\mathbf{a}_{1/2} = [\![\pm\mathbf{M}\boldsymbol{\beta} \pm \mathbf{r} - (\lambda_1/2)\mathbf{1}]\!] \tag{32}$$

$$\mathbf{A} = \operatorname{diag}(\mathbf{a}_1 + \mathbf{a}_2) \tag{33}$$

The matrix $\mathbf{A}$ is an idempotent diagonal matrix with 0/1 entries along the main diagonal. Therefore, $((\mathbf{I} - \mathbf{A})\boldsymbol{\Lambda} + \mathbf{A}\mathbf{R})$ has rank at least $\min(n_u, p)$ and solving (31) for the derivative will require evaluating the Moore-Penrose pseudoinverse (symbolized by "†"). This agrees with Lasso being a convex problem, but not strictly convex. If $\mathbf{R}$ is full rank, i.e., $n \geq p$, then we can develop inverse updates similar to the one for DTR. However, in the general case, efficient updates are not easily derived (see (Meyer 1973)). The CD-V for Lasso is given by Algorithm 5.

3.4 ELASTIC NET

The Elastic Net (EN) problem formulation has $p_\ell = 2$, $J_1(\boldsymbol{\beta}) = \|\boldsymbol{\beta}\|_1^1$, $J_2(\boldsymbol{\beta}) = \|\boldsymbol{\beta}\|_2^2$, and $h_c(\boldsymbol{\beta}) = (\mathbf{I} + \lambda_2\boldsymbol{\Lambda}^{-1})\boldsymbol{\beta}$. In this example, we need to account for calibration, i.e., we must incorporate $\mathbf{D}_{\hat{\boldsymbol{\beta}}}\boldsymbol{\beta}^* = (\mathbf{I} + \lambda_2\boldsymbol{\Lambda}^{-1})$. For positive λ_2 the problem is strictly convex in all cases, so there is no concern about pseudoinverses of rank-deficient matrices. The KKT and

Algorithm 6 Elastic Net CD-V

```
1: Initialize variables
2: while All Stopping Criteria == False do
3:     for $j = 1$ to $burn_{in}$ do
4:         for $i = 1$ to $p$ do
5:             $\beta_i \leftarrow S(R_{ii}\beta_i - \mathbf{R}_i^T\boldsymbol{\beta} + r_i, \frac{1}{2}\lambda_1\mathbf{1})/(R_{ii} + \lambda_2)$
6:         end for
7:     end for
8:     $\mathbf{a}_{1/2} \leftarrow [\![\pm\mathbf{M}\boldsymbol{\beta} \pm \mathbf{r} - \frac{1}{2}\lambda_1\mathbf{1}]\!]$
9:     $\mathbf{A} \leftarrow \mathrm{diag}(\mathbf{a}_1 + \mathbf{a}_2)$
10:    $\mathbf{W} \leftarrow (\boldsymbol{\Lambda} + \lambda_2\mathbf{I} - \mathbf{AM})^{-1}$
11:    $\boldsymbol{\beta}^* = (\mathbf{I} + \lambda_2\boldsymbol{\Lambda}^{-1})\boldsymbol{\beta}$
12:    $\mathbf{D}_{\lambda_1}L_v \leftarrow (\boldsymbol{\beta}^{*T}\mathbf{G} - \mathbf{g}^T)(\mathbf{I} + \lambda_2\boldsymbol{\Lambda}^{-1})\mathbf{W}(\mathbf{a}_2 - \mathbf{a}_1)$
13:    $\mathbf{D}_{\lambda_2}L_v \leftarrow -2(\boldsymbol{\beta}^{*T}\mathbf{G} - \mathbf{g}^T)(\mathbf{I} + \lambda_2\boldsymbol{\Lambda}^{-1})\mathbf{W}\boldsymbol{\beta}$
14:    update$(\boldsymbol{\lambda}|\mathbf{D}_{\boldsymbol{\lambda}}L_v)$
15: end while
```

update equations for EN are given by

$$\mathbf{D}_{\beta_i}L_u = 2\boldsymbol{\beta}^T\mathbf{R}_i - 2\mathbf{r}_i + \lambda_1 \operatorname{sgn}\beta_i + 2\lambda_2\beta_i = 0 \tag{34}$$

$$\beta_i \leftarrow S(R_{ii}\beta_i - \mathbf{R}_i^T\boldsymbol{\beta} + r_i, (\lambda_1/2)\mathbf{1})/(R_{ii} + \lambda_2) \ . \tag{35}$$

At convergence, (35) can be written as

$$\begin{aligned}\boldsymbol{\beta} &= (\boldsymbol{\Lambda} + \lambda_2\mathbf{I})^{-1}S\left(\mathbf{M}\boldsymbol{\beta} + \mathbf{r}, (\lambda_1/2)\mathbf{1}\right) \\ &= (\boldsymbol{\Lambda} + \lambda_2\mathbf{I})^{-1}\langle\mathbf{M}\boldsymbol{\beta} + \mathbf{r} - (\lambda_1/2)\mathbf{1}\rangle \\ &\quad - (\boldsymbol{\Lambda} + \lambda_2\mathbf{I})^{-1}\langle -\mathbf{M}\boldsymbol{\beta} - \mathbf{r} - (\lambda_1/2)\mathbf{1}\rangle\end{aligned} \tag{36}$$

Using (36) we evaluate two derivatives:

$$\mathbf{D}_{\lambda_1}\boldsymbol{\beta} = (1/2)(\boldsymbol{\Lambda} + \lambda_2\mathbf{I} - \mathbf{AM})^{-1}(\mathbf{a}_2 - \mathbf{a}_1) \tag{37}$$

$$\mathbf{D}_{\lambda_2}\boldsymbol{\beta} = -(\boldsymbol{\Lambda} + \lambda_2\mathbf{I} - \mathbf{AM})^{-1}\boldsymbol{\beta} \tag{38}$$

where $\mathbf{a}_1$, $\mathbf{a}_2$, and $\mathbf{A}$ are defined as in the case of Lasso. For sparse $\boldsymbol{\beta}$ and non-zero λ_2, the two derivatives obtained can be computed efficiently with the matrix inversion lemma. This is accomplished by setting $\mathbf{W} = (\boldsymbol{\Lambda} + \lambda_2\mathbf{I})^{-1}$ and updating $\mathbf{W} \leftarrow \mathbf{W} + \frac{1}{1-\mathbf{M}_k^T\mathbf{W}_k}(\mathbf{W}_k\mathbf{M}_k^T)\mathbf{W}$ for all k with $A_{kk} = 1$. The CD-V for EN is given by Algorithm 6.

3.5 *K*-FOLD CROSS VALIDATION

We briefly outline what an analogous algorithm would look like for K-fold cross validation (CV). Without loss of generality, we partition our design matrix into K sub-matrices $\mathbf{X}_{[1]}, \ldots, \mathbf{X}_{[K]}$ where $\mathbf{X}_{[i]} \in \mathbb{R}^{n_i \times p}$ and $n_1 + \ldots + n_K = n$. We partition the response vector in the same way to generate $\mathbf{y}_{[1]}, \ldots, \mathbf{y}_{[K]}$. Let $\mathbf{X}_{[-i]} \in \mathbb{R}^{(n-n_i)\times p}$ denote the design matrix with $\mathbf{X}_{[i]}$ removed (the same notation applies for the response vector as well). In CV we have

$$\hat{\boldsymbol{\beta}}_i = \arg\min_{\boldsymbol{\beta}} \boldsymbol{\beta}^T\mathbf{R}_{[i]}\boldsymbol{\beta} - 2\mathbf{r}_{[i]}^T\boldsymbol{\beta} + \boldsymbol{\lambda}^T\mathbf{J}(\boldsymbol{\beta}) \tag{39}$$

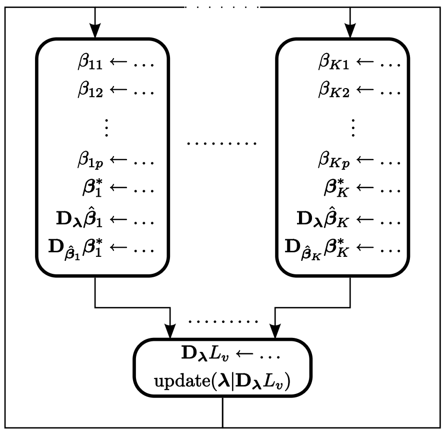
$$\boldsymbol{\beta}_i^* = h_{c[i]}(\hat{\boldsymbol{\beta}}_i) \tag{40}$$

$$L_v(\boldsymbol{\lambda}) = \textstyle\sum_{i=1}^K(\boldsymbol{\beta}^{*T}\mathbf{G}_{[i]}\boldsymbol{\beta}^* - 2\mathbf{g}_{[i]}^T\boldsymbol{\beta}^*) \tag{41}$$

where

$$\mathbf{R}_{[i]} = \frac{1}{n-n_i}\mathbf{X}_{[-i]}^T\mathbf{X}_{[-i]} \qquad \mathbf{G}_{[i]} = \frac{1}{n}\mathbf{X}_{[i]}^T\mathbf{X}_{[i]} \tag{42}$$

$$\mathbf{r}_{[i]} = \frac{1}{n-n_i}\mathbf{X}_{[-i]}^T\mathbf{y}_{[-i]} \qquad \mathbf{g}_{[i]} = \frac{1}{n}\mathbf{X}_{[i]}^T\mathbf{y}_{[i]} \ . \tag{43}$$



$\beta_{11} \leftarrow \ldots$
$\beta_{12} \leftarrow \ldots$
$\beta_{1p} \leftarrow \ldots$
$\boldsymbol{\beta}_1^* \leftarrow \ldots$
$\mathbf{D}_{\boldsymbol{\lambda}}\hat{\boldsymbol{\beta}}_1 \leftarrow \ldots$
$\mathbf{D}_{\hat{\boldsymbol{\beta}}_1}\boldsymbol{\beta}_1^* \leftarrow \ldots$
$\beta_{K1} \leftarrow \ldots$
$\beta_{K2} \leftarrow \ldots$
$\beta_{Kp} \leftarrow \ldots$
$\boldsymbol{\beta}_K^* \leftarrow \ldots$
$\mathbf{D}_{\boldsymbol{\lambda}}\hat{\boldsymbol{\beta}}_K \leftarrow \ldots$
$\mathbf{D}_{\hat{\boldsymbol{\beta}}_K}\boldsymbol{\beta}_K^* \leftarrow \ldots$
$\mathbf{D}_{\boldsymbol{\lambda}}L_v \leftarrow \ldots$
update$(\boldsymbol{\lambda}|\mathbf{D}_{\boldsymbol{\lambda}}L_v)$

Figure 1: Flow Diagram of CD-CV.

The CD-CV algorithm runs K CD-V processes in parallel for cyclical updates of each $\boldsymbol{\beta}_i$ (Fig. 1). After K calibrations, this parallel implementation would yield K $\boldsymbol{\beta}_i^*$'s, which are inserted into

$$\mathbf{D}_{\boldsymbol{\lambda}}L_v = 2\sum_{i=1}^K(\boldsymbol{\beta}_i^{*T}\mathbf{G}_{[i]} - \mathbf{g}_{[i]}^T)(\mathbf{D}_{\hat{\boldsymbol{\beta}}_i}\boldsymbol{\beta}_i^*)(\mathbf{D}_{\boldsymbol{\lambda}}\hat{\boldsymbol{\beta}}_i) \ , \tag{44}$$

which is used for updating $\boldsymbol{\lambda}$.

4 EXPERIMENTS

We now investigate the application of the specified algorithms, starting from the result of a coarse grid search, to several example datasets. The coarse search is implemented by evaluating the estimated loss at every point on a predefined grid. For EN, we employ a raster scan search over the λ_1-λ_2 grid as depicted in Fig. 2. Starting from the top-right point, i.e., the point with the greatest values for λ_1 and λ_2, we solve for the corresponding $\hat{\boldsymbol{\beta}}$, denoted $\hat{\boldsymbol{\beta}}^{(1)}$. We start from this point because the CD rate of convergence is generally faster for larger tuning parameters as there is more shrinkage. Then, keeping λ_2 fixed, as done in LARS-EN, we advance λ_1 in the appropriate direction and use $\hat{\boldsymbol{\beta}}^{(1)}$ for the next initialization in finding

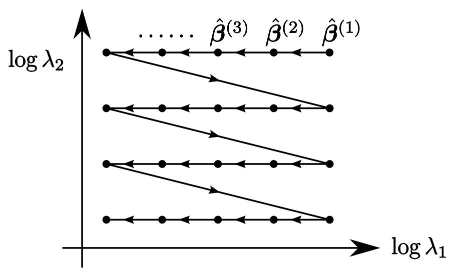

$\ldots\ldots$ $\hat{\boldsymbol\beta}^{(3)}$ $\hat{\boldsymbol\beta}^{(2)}$ $\hat{\boldsymbol\beta}^{(1)}$
$\log\lambda_2$
$\log\lambda_1$

Figure 2: Grid Traversal with a Raster Scan.

$\hat{\boldsymbol\beta}^{(2)}$. This process repeats itself until the whole grid has been covered. As described in (Friedman et al. 2007), this path-wise approach results in fewer main-loop iterations because neighboring (λ_1, λ_2) pairs will generally yield $\hat{\boldsymbol\beta}$'s that are near one another. Certainly, different search paths are possible, e.g., zig-zag or spiral. When $\boldsymbol\lambda$ is a scalar we employ the same technique described, but in one dimension. For computational reasons, when p_ℓ is large the coarse search only considers points on the line $\lambda_1 = \lambda_2 = \ldots = \lambda_{p_\ell}$.

For both validation and K-fold CV, we analyzed 4 datasets. The first data set, denoted $\mathcal{D}^{n,p}_{sim}$, was simulated. Each row of the design matrix, $\mathbf{X} \in \mathbb{R}^{n\times p}$, is drawn from $\mathcal{N}(\mathbf{0}, \boldsymbol\Sigma)$ where $\Sigma_{ij} = \rho^{|i-j|}$ ($\rho = 0.8$). The actual weight vector, denoted $\boldsymbol\beta_a$, was set to $[2, 1, 4, -4, 3, 6, 0, \ldots, 0]^T$ and the response vector was computed according to $\mathbf{y} = \mathbf{X}\boldsymbol\beta_a + \boldsymbol\epsilon$ where $\epsilon_i \sim \mathcal{N}(0, 8)$ iid. To assess performance in the situation where $n < p$ we chose $n = 50$ and $p = 500$. The second dataset is the prostate dataset of (Stamey et al. 1989) with $\mathbf{X} \in \mathbb{R}^{97\times 8}$; the third dataset is the [white] wine dataset of (Cortez et al. 2009) with $\mathbf{X} \in \mathbb{R}^{4898\times 11}$, and the fourth dataset is the voting dataset of (Schlimmer 1987) with $\mathbf{X} \in \mathbb{R}^{435\times 16}$. For TV and K-fold CV we analyzed 100 random permutations of each dataset. All experiments were conducted in Matlab® on an Intel® Core™ i7-920 processor running 64-bit Linux®.

We want to study if the proposed descent methods decrease the minimum estimated expected loss. However, this minimum is usually located in an almost-flat region, in which case a small decrease in loss can require a significant change in $\boldsymbol\lambda$. Therefore, it will be useful to look at how $\boldsymbol\lambda$ changes in a log-scale distance:

$$d(\boldsymbol\lambda_1, \boldsymbol\lambda_2) = 10\left(\textstyle\sum_{i=1}^{p_\ell}(\log_{10}\lambda_{1i} - \log_{10}\lambda_{2i})^2\right)^{1/2}. \tag{45}$$

We can think of this as distance in orders-of-magnitude. As an illustration, $d(0.1, 0.2) = d(1, 2) \approx 3$ and $d(0.2, 0.3) = d(2, 3) \approx 1.76$.

For our coarse grid search, our predefined grid points are $\{10^{-6}, 10^{-5}, \ldots, 10^{3}\}^{p_\ell}$, resulting in a search over 10^{p_ℓ} points. In the case of DTR, we only use the diagonal elements of this set because a coarse search over 10^{p_ℓ} values would only be feasible for small p_ℓ. For the [white] wine dataset we used the methodology outlined in the previous section. We used gradient descent with cubic interpolation for the prostate and voting data. Lastly, a Nelder-Mead approach was taken for the simulated data. Tables 1, 2, and 3 contain 4 columns: the first column is the regularization used, the second column records the number of tuning parameters, the third column is the percentage of refined $\boldsymbol\lambda$'s that yielded a smaller estimated loss (i.e., the coarse search did not trap the descent search in a misleading local minimum), and the fourth column lists the average log-scale-distance between the coarse grid $\boldsymbol\lambda$ and the refined $\boldsymbol\lambda$ when a decrease in loss was encountered. For example, in Table 2 we would say that for EN, in 90% of the trials the descent algorithm yielded a better estimated loss and the refined $\boldsymbol\lambda$ was a log-scale-distance of 4.3 away from the coarse grid search $\boldsymbol\lambda$. In the DTR cases, the mean distance obtained was quite large and we arbitrarily limited this at 20 (greater than 2 decades away).

In general, we note significant refinement for all four types of regression. As expected, DTR showed the most movement due to the unavoidable, highly-coarse grid search. This also exhibits the scalability of our method with respect to p_ℓ. The computed log-scale distances indicate significant refinement from the coarse grid search. For finer grid searches, suitable for small p_ℓ, we would expect to see smaller changes.

Since one of our datasets was simulated, we can go a step further and generate test data to assess whether our results generalize well. Following 2:1 TV, we generated 200 new examples and found that the $\boldsymbol\beta$'s produced by the refined $\boldsymbol\lambda$'s generalized better for all cases, albeit minuscule for ridge. This is exhibited in Table 4 along with computation times (to the nearest second).

5 CONCLUSION

We have proposed a hybrid method of tuning parameter selection involving a coarse search followed by a fast descent algorithm. Motivated by the need to tune parameters in the highly successful Elastic Net, we combined both $\boldsymbol\beta$ and $\boldsymbol\lambda$ updates within the same training loop to minimize estimated expected loss. Whereas $\boldsymbol\beta$ updates were always performed via coordinate descent, the $\boldsymbol\lambda$ updates were designed to accommodate the particular choice of regularization. Our algorithm design did not make use of Hessian matrices for improving rates of convergence. This would introduce more computational cost but may be beneficial for certain difficult regularization functions.

Table 1: Results for 2:1 TV on the Wine Dataset

Method	$\mathbf{p}_\ell$	**Smaller loss encountered**	**Mean log-scale distance**
ridge	1	100%	3.5
Lasso	1	99%	2.5
EN	2	98%	3.4
DTR	11	100%	> 20

Table 2: Results for 5-fold CV on the Prostate Dataset

Method	$\mathbf{p}_\ell$	**Smaller loss encountered**	**Mean log-scale distance**
ridge	1	100%	3.1
Lasso	1	100%	3.2
EN	2	90%	4.3
DTR	8	100%	> 20

Table 3: Results for 10-fold CV on the Voting Dataset

Method	$\mathbf{p}_\ell$	**Smaller loss encountered**	**Mean log-scale distance**
ridge	1	100%	1.8
Lasso	1	100%	2.2
EN	2	99%	2.6
DTR	16	100%	> 20

Table 4: Results for 2:1 TV on Simulated Data

Method	$\mathbf{p}_\ell$	**Comp. Time (CPU time)**	**MSE Improvement**
ridge	1	00:06	0.06%
Lasso	1	00:22	5.05%
EN	2	02:11	8.38%
DTR	500	14:57	26.29%

As demands for performance increase and new regularization methods are introduced, selecting good values for multiple regularization parameters will become increasingly important. Intrinsically, these parameters assign relative weights to different defining characteristics of the data. Therefore, it is reasonable to expect that performance relies heavily on both the choice of regularization and corresponding parameter selection. We have shown that our method scales well with the number of tuning parameters. This is highlighted by the successful tuning of DTR where there are many unknown tuning parameters. Of course, the method is attempting to solve a complex, non-convex problem and hence has inherent limitations. The presence of local minima hinder maximal refinement. Incorporation of effective and computationally efficient stochastic methods may help in this regard.

References

Boyd, S., & Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, *47*(4), 547–553.

Dennis, J., & Schnabel, R. (1996). *Numerical methods for unconstrained optimization and nonlinear equations*. Society for Industrial Mathematics.

Efron, B., Hastie, T., Johnstone, I., & Tibshirani, R. (2004). Least angle regression. *Ann. Statist.*, *32*(2), 407–451.

Friedman, J., Hastie, T., Höfling, H., & Tibshirani, R. (2007). Pathwise coordinate optimization. *Ann. Appl. Stat.*, *1*(2), 302–332.

Fu, W. (2005). Nonlinear GCV and quasi-GCV for shrinkage models. *J. Statist. Plann. Inference*, *131*(2), 333–347.

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

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc.

Hoerl, A. E., & Kennard, R. W. (1970a). Ridge regression: Applications to nonorthogonal problems. *Technometrics*, *12*(1), 69–82.

Hoerl, A. E., & Kennard, R. W. (1970b). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, *12*(1), 55–67.

Magnus, J., & Neudecker, H. (1988). Matrix Differential Calculus with Applications in Econometrics and Statistics.

Meyer, J., Carl D. (1973). Generalized inversion of modified matrices. *SIAM J. Appl. Math.*, *24*(3), 315–323.

Nelder, J., & Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, *7*(4), 308–313.

Schlimmer, J. (1987). *Concept acquisition through representational adjustment*. Ph.D. thesis.

Stamey, T., Kabalin, J., McNeal, J., Johnstone, I., Freiha, F., Redwine, E., & Yang, N. (1989). Prostate specific antigen in the diagnosis and treatment of adenocarcinoma of the prostate II radical prostatectomy treated patients. *J. Urology*, *16*, 1076–1083.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *J. Roy. Statist. Soc. Ser. B*, *58*, 267–288.

Wood, S. (2000). Modelling and smoothing parameter estimation with multiple quadratic penalties. *J. Roy. Statist. Soc. Ser. B*, *62*(2), 413–428.

Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. *J. Roy. Statist. Soc. Ser. B*, *67*, 301–320.