Evaluation and Optimization of Leave-one-out Cross-validation for the Lasso

Ryan Burn

Abstract. I develop an algorithm to produce the piecewise quadratic that computes leave-one-out cross-validation for the lasso as a function of its hyperparameter. The algorithm can be used to find exact hyperparameters that optimize leave-one-out cross-validation either globally or locally, and its practicality is demonstrated on real-world data sets.

1. Introduction. Given observations $(\mathbf{x}^{(1)}, y^{(1)})$, ..., $(\mathbf{x}^{(n)}, y^{(n)})$, $\mathbf{x}^{(i)} \in \mathbb{R}^p$, $y^{(i)} \in \mathbb{R}$, and a shrinkage parameter $t \geq 0$, the *least absolute shrinkage and selection operator* (lasso) produces a solution to the optimization problem

$$\underset{\alpha,\boldsymbol{\beta}}{\arg\min} \left\{ \sum_{i=1}^{n} \left(y^{(i)} - \alpha - \mathbf{x}^{(i)\prime}\boldsymbol{\beta} \right)^2 \right\} \text{ such that } \sum_{j=1}^{p} |\beta_j| \leq t.$$

For t large enough, the lasso is equivalent to ordinary least squares (OLS); for smaller t, the lasso shrinks regression coefficients and can set some coefficients to zero. If t is chosen well, the lasso can both improve on the prediction accuracy of OLS and reveal more important regressors. The key to success with the lasso is estimating how out-of-sample performance is related to t and selecting a good value for the shrinkage parameter.

Arguably leave-one-out cross-validation (LO) is one of the best methods available to estimate prediction accuracy. It has lower bias than other forms of v-fold cross-validation [5, 23]; for many data sets, it is uniquely defined as a function of t; and contrary to what's often claimed, its variance may be no worse than that of v-fold cross-validation with $v < n$ [2, 3, 5]. My aim with this paper is to develop an algorithm, *leave-one-out least angle regression* (LO-LARS), to efficiently produce the piecewise quadratic function that computes LO exactly for any value of t. From there, selecting a good value of the shrinkage parameter is an easy step. We might, for example, choose the value of t that minimizes LO globally, or we might choose a smaller value of t with near optimum LO if we want to achieve greater sparsity. With the piecewise quadratic components of LO, it's straightforward to go from a strategy for selecting t to an exact value.

The lasso appears to have been first used by Fadil Santosa and William Symes in 1986 in an algorithm to deconvolve band-limited reflection seismograms [7]. In reflection seismology, the earth is probed with a seismic wavelet. As the wavelet propagates, it reflects when it passes through geologic interfaces separating layers. The resulting seismogram can be viewed as the convolution of the known source wavelet with an unknown sparse reflector series with spikes corresponding to the geologic interfaces [17, §3.3]. Using a linear approximation of the wave equation and the lasso, the convolution can be inverted to estimate the reflector series and reveal subsurface properties. Use of the lasso with its ℓ_1 restriction on coefficients makes the inversion robust to noise and forces a sparse solution. In their algorithm, Santosa and Symes left the choice of the lasso shrinkage parameter to the user, suggesting that it could be selected interactively, writing,

> The choice of the parameter to weigh the data is left to the user of the algorithm, and hence the method is suited for interactive data interpretations.

A decade after Santosa and Symes' paper, Robert Tibshirani proposed the lasso as a general-purpose tool for the estimation of linear models [18]. Tibshirani highlighted the ability of the lasso to perform both regularization and variable selection as advantages over other methods such as ridge regression or subset selection. Tibshirani additionally proposed

three different methods to select the lasso shrinkage parameter:

i) Estimate prediction error using v-fold cross-validation and select t so as to minimize prediction error across a grid of evenly space values from 0 to $\sum_{j=1}^{p} |\hat{\beta}_j^{\text{OLS}}|$ where $\hat{\boldsymbol{\beta}}^{\text{OLS}}$ represents the least squares estimate.
ii) Same as (i), but use an approximation of generalized cross-validation to estimate prediction error.
iii) Select t to minimize Stein's unbiased estimate of risk.

Benchmarking the three methods with simulated data, Tibshirani reported,

> Estimation of the lasso parameter by generalized cross-validation seems to perform the best, a trend that we find is consistent through all of our examples.

Note that generalized cross-validation is a leave-one-out cross-validation on a rotated version of the data set [9].

In an earlier paper, Prabir Burman also studied the effectiveness of different forms of cross-validation for estimating prediction error [5]. Although Burman did not explicitly look at the lasso, he developed a general theory for the asymptotic performance of cross-validation that applies to the lasso as a special case. Burman showed that if the number of regressors p is fixed and $n \to \infty$, then, under suitable conditions,

$$\mathbb{E}\left[\begin{pmatrix} v\text{-fold cross-} \\ \text{validation error} \end{pmatrix} - (\text{prediction error}) \right] = \frac{c_0}{n(v-1)} + \mathcal{O}(\frac{1}{n^2})$$

where the parameter c_0 depends on the von Mises derivative of the error estimator. It's clear from the formula that bias decreases as the number of folds increases and that cross-validation obtains its lowest asymptotic bias with n folds where the bias is $\mathcal{O}(\frac{1}{n^2})$. Burman also derived a formula for the asymptotic variance of LO, showing

$$\mathbb{V}\Big[\text{LO} - (\text{prediction error})\Big] = \mathcal{O}(\frac{1}{n}).$$

Because of the low bias of LO and the ability of the lasso to select more important variables, there's been a lot of recent interest in using the lasso with LO in high-dimensional settings where p is either greater than or close to n. Kamiar Rad, Haolin Zou, Arnab Auddy, and Arian Maleki proved that versions of Burman's bounds can be extended to certain generalized linear models and nonsmooth regularizers when the ratio p/n remains constant as $n \to \infty$ [13, 23]. Additionally using the matrix inversion lemma, Rad, Wang, and Zhou developed an efficient approximation to LO for regularized generalized linear models [12, 22].

The LO-LARS algorithm integrates the techniques used in efficient LO approximation with a version of the LARS algorithm [6, 11]. Like LARS, LO-LARS works by incrementally building lasso solution paths as a function of t, starting from $t = 0$; unlike LARS, though, it builds solution paths for every leave-one-out subset of the data set, which are used to produce leave-one-out errors. By using the matrix inversion lemma, LO-LARS is able to eliminate much of the overhead that would be incurred if the lasso subproblems were to be solved independently; and by integrating LO computation into a lasso solver, LO-LARS is able to account for activation changes in the subproblems and compute LO exactly.

2. Overview. Before getting into details, I think it will be helpful to look at how LO-LARS works on a few examples. Consider the diabetes data set from the paper by Efron, Hasti,

Johnstone, and Tibshirani that introduces the LARS algorithm [6]:

Patient	Body Mass Index BMI	Age	Sex	Blood Pressure BP	Blood Measurements TC	LDL	HDL	TCH	LTG	GLU	Progression of Diabetes
1	32.1	59	F	101	157	93.2	38	4	4.9	87	151
2	21.6	48	M	87	183	103.2	70	3	3.9	69	75
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
441	30.0	36	M	95	201	125.2	42	5	5.1	85	220
442	19.6	36	M	71	250	133.2	97	3	4.6	92	57

Table 1. *Preview of the diabetes data set from [6]. The data set measures the progression of diabetes in 442 patients over the course of a year. The 10 regressors are measured at start of the year.*

The data set quantitatively measures the progression of diabetes in 442 patients after a year. At the start of the year, 10 baseline variables are measured. Efron et al. explain the modeling objective as,

> Two hopes were evident here, that the model would produce accurate baseline predictions of response for future patients and that the form of the model would suggest which covariates were important factors in disease progression.

LO-LARS computes the following LO function for the data set:

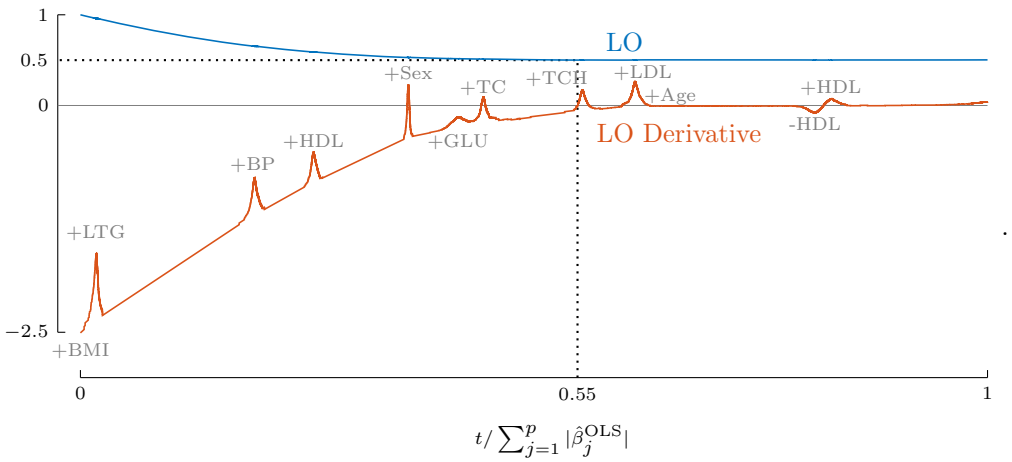

Figure 1. *Plot of LO and its derivative for the diabetes data set from [6]. Mean squared error is normalized to have a maximum value of 1 and t is normalized by the ℓ_1 norm of the OLS coefficients. The points of activation and deactivation for each of the ten regressors are shown. We can see that each spike in the LO derivative is associated with a change in activation. At its minimum,* LO ≈ 0.5.

LO is a continuous piecewise quadratic; and its derivative is, therefore, a piecewise linear function with jump discontinuities. Each point of discontinuity in the derivative corresponds to a change in activation of a leave-one-out subproblem. From the segments, it's easy to compute all the local minimums,

$t/\sum_{j=1}^{p}\lvert\hat{\beta}_j^{\mathrm{OLS}}\rvert$	LO	Active Regressors
0.360	0.52952	`BMI, LTG, BP, HDL`
0.442	0.51180	`BMI, LTG, BP, HDL, Sex, GLU`
0.548	0.50052*	`BMI, LTG, BP, HDL, Sex, GLU, TC`
0.597	0.50058	`BMI, LTG, BP, HDL, Sex, GLU, TC, TCH`
0.819	0.50090	`BMI, LTG, BP, Sex, GLU, TC, TCH, LDL, Age`
0.860	0.50182	`BMI, LTG, BP, HDL, Sex, GLU, TC, TCH, LDL, Age`
0.887	0.50178	`BMI, LTG, BP, HDL, Sex, GLU, TC, TCH, LDL, Age`

Table 2. *Local optimums of LO for the diabetes data set. Also shown are the nonzero regressors at each optimum.*

LO-LARS can also produce individual leave-one-out errors as a byproduct,

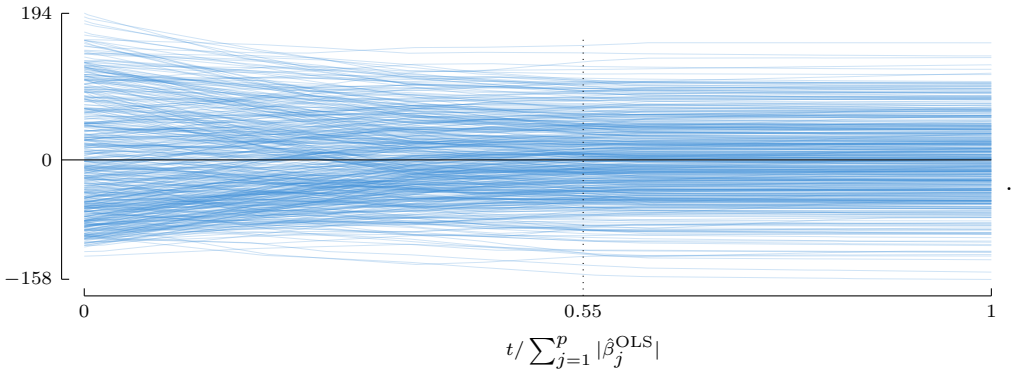

Figure 2. *Individual leave-one-out errors, actual progression minus predicted progression, as a function of the shrinkage parameter for each of the 442 patients in the diabetes data set.*

For many data sets, computing LO fully is entirely feasible. However for high-dimensional data sets where $p \gg n$, computing LO for larger values of t can become prohibitively expensive as individual solution paths won't overlap as much in their active regressors. Fortunately, it's easy to modify LO-LARS to detect local minimums as it incrementally expands the domain of LO and to bail out of the algorithm if LO increases past a given threshold of the best optimum found so far. With this modification, many higher dimensional data sets also become tractable. For example, this data set measures riboflavin levels and the expression of 4088 genes in cultures of the bacteria *Bacillus subtilis*:

	← 4088 Genes →						Log Riboflavin
Sample	AADK_at	AAPA_at	ABH_at	⋯	YYZB_at	zur_at	Production Rate
1	8.492	8.111	8.321	⋯	7.173	7.427	-6.644
2	7.639	7.240	7.793	⋯	7.187	7.545	-6.948
⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮
70	7.343	7.743	7.709	⋯	7.539	7.310	-6.764
71	6.856	8.275	7.989	⋯	7.230	6.656	-7.587

Table 3. *Riboflavin data set initially from [10] and made available by [4]. The data set measures log riboflavin production rate of the bacteria Bacillus subtilis and log transcription levels for 4088 genes.*

Riboflavin is known to regulate the transcription of certain genes, and a lasso model can help to identify which genes are influenced by riboflavin. If I instruct LO-LARS to exit early if LO exceeds 1% of the best optimum found so far, I can compute the LO function below up to the early stopping point in approximately half a second. By comparison, computing the whole LO function takes slightly over 3 seconds.

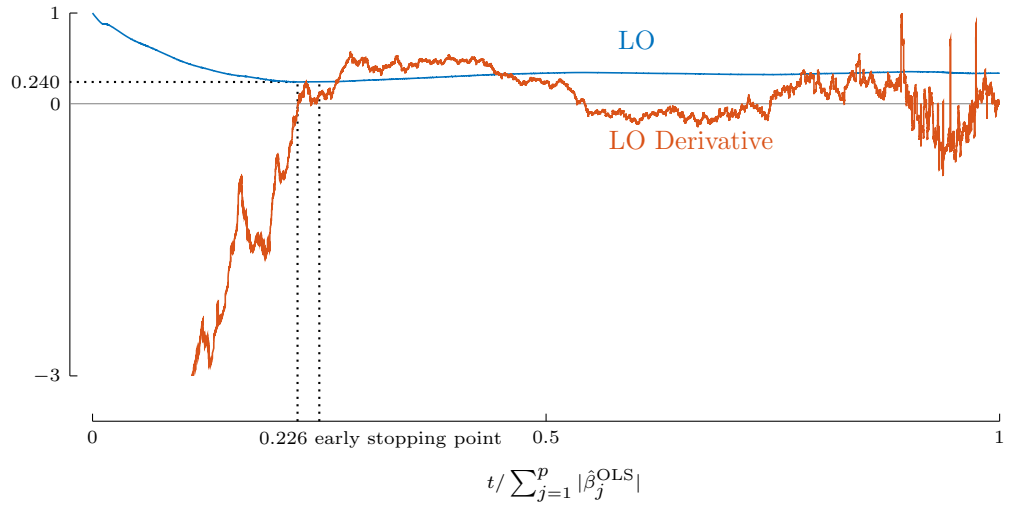

Figure 3. *Plot of LO and its derivative for the riboflavin data set from [4]. Mean squared error is normalized to have a maximum value of 1 and t is normalized by the ℓ_1 norm of the OLS coefficients. At its optimum, $t/\sum_{j=1}^{p}|\hat{\boldsymbol{\beta}}^{OLS}| \approx 0.226$, $LO \approx 0.24$, and the regressors for 20 genes are active. If we instruct LO-LARS to stop early with a 1% threshold, it still finds the global optimum and aborts at $t/\sum_{j=1}^{p}|\hat{\boldsymbol{\beta}}^{OLS}| \approx 0.25$.*

As we can see from the graph, the LO derivative gets quite choppy for larger values of t. The choppiness corresponds to divergence in the sets of active regressors for the leave-one-out solution paths, which makes LO more expense to compute.

The table below shows benchmarking results for a variety of real-world data sets of varying dimensions.

Dataset	n	p	Num Active Regressors	$t/\sum_j \lvert\beta_j^{\mathrm{OLS}}\rvert$	LO	Avg Elapse Over 5 Runs (sec)
`diabetes`	442	10	7	0.55	0.50	0.004
`California housing`	20640	8	8	0.97	0.40	0.092
`diabetes (Q)`	442	64	15	0.03	0.49	0.220
`riboflavin (E)`	71	4088	20	0.23	0.24	0.571
`SARCOS inverse`	44484	21	20	0.99	0.07	1.161
`California housing (Q)`	20640	44	39	0.33	0.35	2.680
`riboflavin`	71	4088	20	0.23	0.24	3.385
`SARCOS inverse (QE)`	44484	252	58	0.09	0.05	16.351
`SARCOS inverse (Q)`	44484	252	252	0.99	0.03	349.229

Table 4. *Benchmarks of a variety of real world data sets. Shown are the dimensions of the data set, the number of active regressors at the optimum, the shrinkage of the optimum, LO at optimum (expressed relative to LO with maximal shrinkage), and the average elapse over 5 runs.* `Q` *indicates that the original regressors were expanded with quadratic terms and* `E` *indicates that LO-LARS was stopped early with a 1% threshold. Details are given in Appendix A.*

We can clearly see that LO-LARS is fast enough to be practical for many data sets. To get a better understanding of what determines its performance and its limitations, we can look at how long it takes to fit certain simulated data sets.

Consider the linear model

$$\begin{aligned} y &= \mathbf{x}'\boldsymbol{\beta}_{\mathrm{true}} + \varepsilon, \\ \mathbf{x} &\sim N(\mathbf{0}_p, \Sigma) \quad \text{with} \quad \Sigma_{ij} = 0.8^{|i-j|}, \quad \text{and} \\ \varepsilon &\sim N(\mathbf{0}_n, \sigma^2\mathbf{I}) \quad \text{with} \quad \sigma^2 = \boldsymbol{\beta}'_{\mathrm{true}}\Sigma\boldsymbol{\beta}_{\mathrm{true}}. \end{aligned}$$

Suppose that the first k regressors of $\boldsymbol{\beta}_{\mathrm{true}}$ are drawn from a standard Laplace distribution and the remaining $p-k$ regressors are set to 0. The table below shows how long it takes me to run LO-LARS with an early exit threshold of 1% for all combinations of n, $p \in \{\,800,\ 1600\,\}$ and $k \in \{\,10,\ 50\,\}$.

n	p	k	Num Active Regressors	Avg Elapse Over 10 Runs (sec)
800	800	10	38.4	12.05
800	800	50	70.4	20.81
800	1600	10	57.3	36.90
800	1600	50	92.0	59.27
1600	800	10	51.0	52.01
1600	800	50	110.0	101.34
1600	1600	10	86.2	227.33
1600	1600	50	106.4	217.25

Table 5. *Benchmarks for fitting different simulated data sets with LO-LARS and an early exit threshold of 1%. Each entry is averaged over 10 runs. Shown are the dimensions of the simulated linear model, the number of nonzero true regressors, k, the average number of nonzero regressors of the LO optimum, and the average elapse time to run LO-LARS.*

3. Solution Paths. Given $\mathbf{y} \in \mathbb{R}^n$, $\mathbf{X} \in \mathbb{R}^{n\times p}$, and $\lambda \geq 0$, consider

$$\frac{1}{2}\left\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\right\|^2 + \lambda \sum_{j=p'+1}^{p} \left|\beta_j\right|. \tag{1}$$

If $\hat{\boldsymbol{\beta}}$ minimizes (1), then $\hat{\boldsymbol{\beta}}$ solves a lasso problem with the first p' regressors unrestricted and the ℓ^1-norm of the remaining regressors bounded above by $\sum_{j=p'+1}^{p} |\hat{\beta}_j|$. By allowing for unrestricted regressors, we can incorporate the intercept term into cross-validation estimates, and we can solve certain generalized lasso problems [20]. We'll call a function $\hat{\boldsymbol{\beta}}(\lambda) : [0,\infty) \to \mathbb{R}^p$ a *solution path* if $\hat{\boldsymbol{\beta}}(\lambda)$ is continuous and

$$\hat{\boldsymbol{\beta}}(\lambda) \in \operatorname*{arg\,min}_{\boldsymbol{\beta}} \left\{ \frac{1}{2}\left\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\right\|^2 + \lambda \sum_{j=p'+1}^{p} \left|\beta_j\right| \right\} \qquad \text{for all } \lambda \geq 0.$$

Consider, for example, the data set

$$\begin{array}{cccccc} x_1 & 0.09 & -0.88 & -1.77 & -0.10 & 1.00 \\ x_2 & 0.01 & 0.91 & -1.04 & 0.81 & 0.27 \\ y & -0.09 & -1.57 & -1.47 & -1.08 & 1.49 \end{array}.$$

The piecewise linear solution path is specified by the nodes

$$\begin{array}{cccccc} \lambda & 0 & 1.412 & 5.573 & \infty \\ x_1 & 1.321 & 0.845 & 0 & 0 \\ x_2 & -0.757 & 0 & 0 & 0 \end{array}. \tag{2}$$

Solution paths need not be unique. For instance, if we duplicate a regressor,

$$\begin{array}{cccccc} x_1 & 0.09 & -0.88 & -1.77 & -0.10 & 1.00 \\ x_2 & 0.01 & 0.91 & -1.04 & 0.81 & 0.27 \\ x_3 & 0.01 & 0.91 & -1.04 & 0.81 & 0.27 \\ y & -0.09 & -1.57 & -1.47 & -1.08 & 1.49 \end{array}\ ,$$

then clearly (2) is still a solution path, but we can easily form infinitely many additional solution paths from weighted averages of x_2 and x_3.

We can use convex optimization theory to derive some basic properties of solution paths. Define $\mathcal{L}(\boldsymbol{\beta}) := \frac{1}{2}\left\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\right\|^2$, $r(\boldsymbol{\beta}) := \sum_{j=p'+1}^{p} |\beta_j|$, $f(\boldsymbol{\beta};\lambda) := \mathcal{L}(\boldsymbol{\beta}) + \lambda\, r(\boldsymbol{\beta})$, and

$$\texttt{active}(\boldsymbol{\beta}) := \begin{cases} \text{The ordered vector that corresponds to the set} \\ \qquad \{\,1,\ldots,p'\,\} \cup \{\,j \text{ such that } \beta_j \neq 0\,\}\,. \end{cases}$$

The following theorem is adopted from Ryan Tibshirani [19].

Theorem P. *If $\lambda \geq 0$, then*

1. *There exists a $\hat{\boldsymbol{\beta}}$ such that $\hat{\boldsymbol{\beta}}$ minimizes $f(\,\cdot\,;\lambda)$ and*
$$\operatorname{rank} \mathbf{X}_{\texttt{active}(\hat{\boldsymbol{\beta}})} = \#\texttt{active}(\hat{\boldsymbol{\beta}})$$
where # provides the length of a vector.
2. *If $\hat{\boldsymbol{\beta}}_1$ and $\hat{\boldsymbol{\beta}}_2$ both minimize $f(\,\cdot\,;\lambda)$, then $\mathcal{L}(\hat{\boldsymbol{\beta}}_1) = \mathcal{L}(\hat{\boldsymbol{\beta}}_2)$ and $r(\hat{\boldsymbol{\beta}}_1) = r(\hat{\boldsymbol{\beta}}_2)$ if $\lambda > 0$.*
3. *The vector $\hat{\boldsymbol{\beta}}$ minimizes $f(\,\cdot\,;\lambda)$ if and only if $\mathbf{X}'\left(\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}\right) = \lambda\,\boldsymbol{\gamma}$ for some $\boldsymbol{\gamma}$ where*
$$\gamma_j \in \begin{cases} \{\,0\,\}, & \text{if } j \leq p'; \\ \{\,\operatorname{sign}\beta_j\,\}, & \text{if } j > p' \text{ and } \beta_j \neq 0; \\ [-1,\,1]\,, & \text{otherwise.} \end{cases}$$

Proof of P1. If $\lambda = 0$, let $\mathcal{A}$ denote a set of regressors such that $\mathbf{X}_{\mathcal{A}}$ has maximal rank. Set $\hat{\boldsymbol{\beta}}$ so that $\hat{\boldsymbol{\beta}}_{-\mathcal{A}} = \mathbf{0}$ and $\hat{\boldsymbol{\beta}}_{\mathcal{A}} = \left(\mathbf{X}'_{\mathcal{A}}\mathbf{X}_{\mathcal{A}}\right)^{-1}\mathbf{X}'_{\mathcal{A}}\mathbf{y}$. Then $\hat{\boldsymbol{\beta}}$ minimizes $f(\,\cdot\,;\lambda)$ and the nonzero entries identify regressors of full rank.

The next part of the proof follows Rosset [16]. Suppose $\lambda > 0$. Assume first that the unrestricted regressors are of full rank. Clearly, $f(\boldsymbol{\beta};\lambda) \to \infty$ as $\|\boldsymbol{\beta}\| \to \infty$. Choose R so that $f(\boldsymbol{\beta};\lambda) > f(\mathbf{0};\lambda)$ if $\|\boldsymbol{\beta}\| > R$. Since

$$B_R(\mathbf{0}) = \left\{\,\boldsymbol{\beta} \text{ such that } \|\boldsymbol{\beta}\| \leq R\,\right\}$$

is compact and f is continuous, f attains a minimum, $\hat{\boldsymbol{\beta}}$, in $B_R(\mathbf{0})$. Suppose that

$$\operatorname{rank}\mathbf{X}_{\mathcal{A}} < \#(\mathcal{A}) \qquad \text{for } \mathcal{A} = \texttt{active}(\hat{\boldsymbol{\beta}}).$$

Then we can pick $\mathbf{u} \neq \mathbf{0}$ such that $\mathbf{u}_{-\mathcal{A}} = \mathbf{0}$ and $\mathbf{X}_{\mathcal{A}}\mathbf{u}_{\mathcal{A}} = \mathbf{0}$. Put $g(t) = f(\hat{\boldsymbol{\beta}} + t\mathbf{u};\lambda)$. Then g is differentiable in a neighborhood of 0 and $g'(0) = 0$; otherwise we would have $f(\hat{\boldsymbol{\beta}} + \delta\mathbf{u};\lambda) < f(\hat{\boldsymbol{\beta}};\lambda)$ for some δ. Put

$$\hat{t} = \arg\min_t |t| \ \text{ such that } \left(\hat{\boldsymbol{\beta}} + t\mathbf{u}\right)_j = 0 \text{ for some } j \in \mathcal{A} \text{ and } j > p'$$

and form a new minimum with fewer active regressors by setting $\hat{\boldsymbol{\beta}} \leftarrow \hat{\boldsymbol{\beta}} + \hat{t}\mathbf{u}$. If the nonzero coefficients of $\hat{\boldsymbol{\beta}}$ still do not identify regressors of full rank, repeat the process until the full rank condition is satisfied.

Finally, assume the first p' regressors are not of full rank. Let $\mathcal{F}$ be a subset of $\{1,\ldots,p'\}$ identifying regressors of full rank such that $\operatorname{rank}\mathbf{X}_{\mathcal{F}}$ is maximized. Put $\mathcal{F} \leftarrow \mathcal{F} \cup \{p'+1,\ldots,p\}$. Set $\hat{\boldsymbol{\beta}}_{-\mathcal{F}} = \mathbf{0}$ and set $\hat{\boldsymbol{\beta}}_{\mathcal{F}}$ to be a full rank minimizer of the lasso problem for $(\mathbf{X}_{\mathcal{F}}, \mathbf{y})$. ∎

Proof of P2. Suppose that $\hat{\boldsymbol{\beta}}_1$ and $\hat{\boldsymbol{\beta}}_2$ are two minimums of f_λ and $\mathcal{L}(\hat{\boldsymbol{\beta}}_1) \neq \mathcal{L}(\hat{\boldsymbol{\beta}}_2)$. Put

$$g(t) = \mathcal{L}(\hat{\boldsymbol{\beta}}_1 + t(\hat{\boldsymbol{\beta}}_2 - \hat{\boldsymbol{\beta}}_1)).$$

Then

$$g(t) = g(0) + g'(0)t + \frac{1}{2}g''(0)t^2 \quad \text{and} \quad g''(0) > 0.$$

Now,

$$\begin{aligned}\frac{1}{2}\left(g(0) + g(1)\right) &= g(0) + \frac{1}{2}g'(0) + \frac{1}{4}g''(0) \\ &> g(0) + \frac{1}{2}g'(0) + \frac{1}{8}g''(0) = g(\frac{1}{2}).\end{aligned}$$

Thus,

$$\begin{aligned}f(\frac{1}{2}\hat{\boldsymbol{\beta}}_1 + \frac{1}{2}\hat{\boldsymbol{\beta}}_2; \lambda) &= \mathcal{L}(\frac{1}{2}\hat{\boldsymbol{\beta}}_1 + \frac{1}{2}\hat{\boldsymbol{\beta}}_2) + \lambda\, r(\frac{1}{2}\hat{\boldsymbol{\beta}}_1 + \frac{1}{2}\hat{\boldsymbol{\beta}}_2) \\ &= g(\frac{1}{2}) + \lambda\, r(\frac{1}{2}\hat{\boldsymbol{\beta}}_1 + \frac{1}{2}\hat{\boldsymbol{\beta}}_2) \\ &< \frac{1}{2}\mathcal{L}(\hat{\boldsymbol{\beta}}_1) + \frac{1}{2}\mathcal{L}(\hat{\boldsymbol{\beta}}_2) + \lambda\, r(\frac{1}{2}\hat{\boldsymbol{\beta}}_1 + \frac{1}{2}\hat{\boldsymbol{\beta}}_2).\end{aligned}$$

By the triangle inequality, $r(\boldsymbol{\beta}_1 + \boldsymbol{\beta}_2) \le r(\boldsymbol{\beta}_1) + r(\boldsymbol{\beta}_2)$; hence,

$$\begin{aligned}f(\frac{1}{2}\hat{\boldsymbol{\beta}}_1 + \frac{1}{2}\hat{\boldsymbol{\beta}}_2; \lambda) &< \frac{1}{2}\left(\mathcal{L}(\hat{\boldsymbol{\beta}}_1) + \lambda\, r(\hat{\boldsymbol{\beta}}_1)\right) + \frac{1}{2}\left(\mathcal{L}(\hat{\boldsymbol{\beta}}_2) + \lambda\, r(\hat{\boldsymbol{\beta}}_2)\right) \\ &= \frac{1}{2}f(\hat{\boldsymbol{\beta}}_1; \lambda) + \frac{1}{2}f(\hat{\boldsymbol{\beta}}_2; \lambda).\end{aligned}$$

But since $\hat{\boldsymbol{\beta}}_1$ and $\hat{\boldsymbol{\beta}}_2$ are both minimums, $f(\hat{\boldsymbol{\beta}}_1; \lambda) = f(\hat{\boldsymbol{\beta}}_2; \lambda)$; and $\frac{1}{2}\left(\hat{\boldsymbol{\beta}}_1 + \hat{\boldsymbol{\beta}}_2\right)$ must therefore be a minimum smaller than the others—a contradiction. We conclude that $\mathcal{L}(\hat{\boldsymbol{\beta}}_1) = \mathcal{L}(\hat{\boldsymbol{\beta}}_2)$ and likewise $\lambda\, r(\hat{\boldsymbol{\beta}}_1) = \lambda\, r(\hat{\boldsymbol{\beta}}_2)$. ∎

To prove P3, we make use of subgradients [15, §23]. If $f : \mathbb{R}^n \to \mathbb{R}$ is a convex function, then a subgradient of f at a point $\mathbf{x}_0$ is a vector $\mathbf{u} \in \mathbb{R}^n$ such that

$$f(\mathbf{x}) - f(\mathbf{x}_0) \ge \mathbf{u}'(\mathbf{x} - \mathbf{x}_0) \quad \text{for all} \quad \mathbf{x} \in \mathbb{R}^n.$$

The set of all subgradients of f at $\mathbf{x}_0$, denoted by $\partial f(\mathbf{x}_0)$, is called a subdifferential. Clearly, $\mathbf{x}_0$ is a minimum of f if and only if $\mathbf{0} \in \partial f(\mathbf{x}_0)$.

Proof of P3. We have $|x| \ge cx$ for all x if and only if $c \in [-1, 1]$. Thus, the subdifferential of r at $\boldsymbol{\beta}$ is given by

$$\boldsymbol{\gamma} \in \partial r(\boldsymbol{\beta}) \quad \text{if and only } \gamma_j \in \begin{cases} \{\, 0 \,\}, & \text{if } j \le p'; \\ \{\, \operatorname{sign}\beta_j \,\}, & \text{if } j > p' \text{ and } \beta_j \ne 0; \\ [-1,\, 1]\,, & \text{otherwise;} \end{cases} \qquad \text{for all } j.$$

The subdifferential of $\mathcal{L}$ is given by its gradient,

$$\partial\mathcal{L}(\boldsymbol{\beta}) = \{\, -\mathbf{X}'\left(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\right) \}\,.$$

So, $\mathbf{0} \in \partial f(\hat{\boldsymbol{\beta}}; \lambda)$ is equivalent to

$$\mathbf{X}'\left(\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}\right) = \lambda\,\boldsymbol{\gamma} \quad \text{for some } \boldsymbol{\gamma} \in \partial r(\hat{\boldsymbol{\beta}}),$$

which establishes the result. ∎

4. LARS Algorithm. In this section, we develop a lasso solver built on the primitive operations used for QR factorizations. When there is only a single solution path, the solver will produce results identical to LARS; when there are multiple solution paths, it differs from LARS in that it will produce a solution path with the smallest active set instead of the largest active set [19, §5.1].

Define

$$\texttt{signs}(\boldsymbol{\beta}) := \begin{cases} \text{Set } \mathcal{A} \leftarrow \texttt{active}(\boldsymbol{\beta}) \text{ and return the vector } \mathbf{s} \text{ such that} \\ \qquad s_j = \begin{cases} 0, & \text{if } j \le p'; \\ \operatorname{sign} \beta_{\mathcal{A}_j}, & \text{otherwise.} \end{cases} \end{cases}$$

$\texttt{complete}(\mathcal{A}, \boldsymbol{\beta}) :=$ the vector $\mathbf{u}$ such that $\mathbf{u}_{-\mathcal{A}} = \mathbf{0}$ and $\mathbf{u}_{\mathcal{A}} = \boldsymbol{\beta}$.

Suppose $\hat{\boldsymbol{\beta}}_0$ minimizes $f(\,\cdot\,; \lambda_0)$ and $\operatorname{rank} \mathbf{X}_{\mathcal{A}} = \#\mathcal{A}$ where $\mathcal{A} = \texttt{active}(\hat{\boldsymbol{\beta}}_0)$. Put $\mathbf{s} \leftarrow \texttt{signs}(\hat{\boldsymbol{\beta}}_0)$,

$$\boldsymbol{\gamma}(\lambda) \leftarrow (\mathbf{X}'_{\mathcal{A}}\mathbf{X}_{\mathcal{A}})^{-1} \left[\mathbf{X}'_{\mathcal{A}}\mathbf{y} - \lambda \mathbf{s}\right], \quad \text{and} \quad \xi(\lambda) \leftarrow \mathbf{X}'_{-\mathcal{A}}\big(\mathbf{y} - \mathbf{X}_{\mathcal{A}}\boldsymbol{\gamma}(\lambda)\big).$$

If $\min_j |\gamma(\lambda)_j| \ge 0$ and $\max_j |\xi(\lambda)_j| \le \lambda$ for all $\lambda \in [\lambda', \lambda_0]$, then Theorem P tells us that

$$\texttt{complete}(\mathcal{A}, \boldsymbol{\gamma}(\lambda)) \in \operatorname*{arg\,min}_{\boldsymbol{\beta}} f(\boldsymbol{\beta}; \lambda) \qquad \text{for all } \lambda \in [\lambda', \lambda_0].$$

The function `step` takes a piece of a solution path, enlarges its domain as much as possible, and identifies the constraint that is eventually violated.

$$\texttt{step}(\boldsymbol{\gamma}, \xi, \mathbf{s}, \lambda_0) := \begin{cases} \text{Decrease } \lambda' \text{ starting from } \lambda_0 \text{ until one of the following} \\ \text{conditions is true:} \\ \quad \lambda' = 0, \\ \quad\quad \text{return } 0, \texttt{NULL}; \\ \quad \boldsymbol{\gamma}(\lambda')_j = 0 \text{ and } \operatorname{sign} \boldsymbol{\gamma}(0)_j \ne \operatorname{sign} s_j \text{ for some } j > p', \\ \quad\quad \text{return } \lambda', (\texttt{DEACTIVATE}, j); \\ \quad |\xi(\lambda')_j| = \lambda' \text{ and } \frac{d}{d\lambda} |\xi(\lambda)_j| \text{ at } \lambda' < 1, \\ \quad\quad \text{return } \lambda', (\texttt{ACTIVATE}, j). \end{cases}$$

With `step`, we can sketch out an algorithm to solve for $\hat{\boldsymbol{\beta}}(\lambda)$. Start from $\lambda_0 = \infty$ and repeatedly step backwards, solving for segments $\boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_N$, until the entire interval $[0, \infty)$ is covered. Before proceeding, though, we should address an issue of efficiency: computing $\boldsymbol{\gamma}$ and ξ naively at each step by evaluating and solving expressions with $\mathbf{X}'_{\mathcal{A}}\mathbf{X}_{\mathcal{A}}$ would be expensive. For a solver to be practical, we need to do better.

Let $\mathbf{Q}_{\mathcal{A}}$, $\mathbf{R}_{\mathcal{A}}$ denote a QR factorization for $\mathbf{X}_{\mathcal{A}}$,

$$\mathbf{X}_{\mathcal{A}} = \mathbf{Q}_{\mathcal{A}} \begin{pmatrix} \mathbf{R}_{\mathcal{A}} \\ \mathbf{0} \end{pmatrix}.$$

Then

$$\begin{aligned} \boldsymbol{\gamma}(\lambda) &= (\mathbf{X}'_{\mathcal{A}}\mathbf{X}_{\mathcal{A}})^{-1} [\mathbf{X}'_{\mathcal{A}}\mathbf{y} - \lambda \mathbf{s}] \\ &= (\mathbf{R}'_{\mathcal{A}}\mathbf{R}_{\mathcal{A}})^{-1} [(\,\mathbf{R}'_{\mathcal{A}} \quad \mathbf{0}\,)\, \mathbf{Q}'_{\mathcal{A}}\mathbf{y} - \lambda \mathbf{s}] \\ &= (\,\mathbf{R}_{\mathcal{A}}^{-1} \quad \mathbf{0}\,)\, \mathbf{Q}'_{\mathcal{A}}\mathbf{y} - \lambda \mathbf{R}_{\mathcal{A}}^{-1}\mathbf{R}_{\mathcal{A}}^{-1\ \prime}\mathbf{s} \end{aligned}$$

and

$$\begin{aligned} \xi(\lambda) &= \mathbf{X}'_{-\mathcal{A}} [\mathbf{y} - \mathbf{X}_{\mathcal{A}}\boldsymbol{\gamma}(\lambda)] \\ &= \mathbf{X}'_{-\mathcal{A}} \left[\mathbf{y} - \mathbf{Q}_{\mathcal{A}} \begin{pmatrix} \mathbf{R}_{\mathcal{A}} \\ \mathbf{0} \end{pmatrix} \left((\,\mathbf{R}_{\mathcal{A}}^{-1} \quad \mathbf{0}\,)\, \mathbf{Q}'_{\mathcal{A}}\mathbf{y} - \lambda \mathbf{R}_{\mathcal{A}}^{-1}\mathbf{R}_{\mathcal{A}}^{-1\ \prime}\mathbf{s} \right) \right] \\ &= (\mathbf{Q}'_{\mathcal{A}}\mathbf{X}_{-\mathcal{A}})' \left[\begin{pmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \mathbf{Q}'_{\mathcal{A}}\mathbf{y} + \lambda \begin{pmatrix} \mathbf{R}_{\mathcal{A}}^{-1\ \prime} \\ \mathbf{0} \end{pmatrix} \mathbf{s} \right]. \end{aligned}$$

Thus, with $\mathbf{R}_{\mathcal{A}}$, $\mathbf{Q}'_{\mathcal{A}}\mathbf{X}_{-\mathcal{A}}$, and $\mathbf{Q}'_{\mathcal{A}}\mathbf{y}$, the parametric linear equations for $\boldsymbol{\gamma}(\lambda)$ and $\xi(\lambda)$ can be computed big-O of

$$\big(\text{num active regressors}\big)^2 + \big(\text{num data}\big) \times \big(\text{num inactive regressors}\big).$$

Let $\mathbf{F}_{\mathcal{A}} \in \mathbb{R}^{n\times(p+1)}$ denote what we'll call a *factor matrix* for $\mathcal{A}$,

$$\mathbf{F}_{\mathcal{A}} = \begin{pmatrix} \mathbf{R}_{\mathcal{A}} & \mathbf{Q}_{\mathcal{A}}'\mathbf{X}_{-\mathcal{A}} & \mathbf{Q}_{\mathcal{A}}'\mathbf{y} \\ \mathbf{0} & & \end{pmatrix}.$$

If we can update $\mathbf{F}_{\mathcal{A}}$ as $\mathcal{A}$ changes, we can use it to efficiently compute $\boldsymbol{\gamma}$ and ξ. We need to consider two cases: addition of a new regressor into $\mathcal{A}$ and removal of a regressor from $\mathcal{A}$. Fortunately, both can be handled by algorithms that mirror QR factorization by Householder reflectors and Givens rotations, respectively [8, ch. 5].

Algorithm A (*Activate a regressor*). Given a factor matrix, $\mathbf{F}$, with r active regressors, update $\mathbf{F}$ to activate the j-th inactive regressor.

A1. [Swap columns.] Swap the $(r+1)$-th column of $\mathbf{F}$ with the $(r+j)$-th column.

A2. [Update $(r+1)$-th Column.] Let $\mathbf{H}$ denote a Householder reflector such that

$$\mathbf{H}\,(\,F_{1,r+1} \quad \cdots \quad F_{r,r+1} \quad * \quad \cdots \quad *\,)' \\ = (\,F_{1,r+1} \quad \cdots \quad F_{r,r+1} \quad * \quad 0 \quad \cdots \quad 0\,)'.$$

Set $\mathbf{F} \leftarrow \mathbf{HF}$. ∎

Algorithm A can be implemented using the LAPACK routines `LARFG` and `ORMQR`, and it has computational complexity big-O of

$$(\text{num data}) + (\text{num data} - \text{num active regressors}) \times (\text{num inactive regressors}).$$

Algorithm D (*Deactivate a regressor*). Given a factor matrix, $\mathbf{F}$, with r active regressors, update $\mathbf{F}$ to deactivate the j-th active regressor.

D1. [Permute Columns.] Pick $\mathbf{P}$ to be a permutation matrix so that $\mathbf{PF}$ performs a left cyclical rotation of columns j, $j+1$, …, r of $\mathbf{F}$:

$$(\mathbf{PF})_j = \mathbf{F}_{j+1}, \quad \ldots, \quad (\mathbf{PF})_{r-1} = \mathbf{F}_r, \quad \text{and} \quad (\mathbf{PF})_r = \mathbf{F}_j.$$

Set $\mathbf{F} \leftarrow \mathbf{PF}$.

D2. [Annihilate subdiagonal.] For $j \le k < r$, let $\mathbf{G}_k$ denote a Givens rotation such that

$$\mathbf{G}_k\,(\,F_{1k} \quad \cdots \quad F_{k-1,k} \quad * \quad * \quad 0 \quad \cdots \quad 0\,)' \\ = (\,F_{1k} \quad \cdots \quad F_{k-1,k} \quad * \quad 0 \quad 0 \quad \cdots \quad 0\,)'.$$

Set $\mathbf{F} \leftarrow \mathbf{G}_{r-1}\mathbf{G}_{r-2}\cdots\mathbf{G}_j\mathbf{F}$. ∎

Similarly, Algorithm D can be implemented with the BLAS routine `ROT` and has computational cost big-O of

$$(\text{num active regressors})^2 + (\text{num active regressors}) \times (\text{num inactive regressors}).$$

With Algorithm A and Algorithm D, we are now in a position to put together an efficient solver for $\hat{\boldsymbol{\beta}}(\lambda)$.

Algorithm S (*Compute a full rank solution path*). Suppose $\mathbf{X} \in \mathbb{R}^{n\times p}$, $\mathbf{y} \in \mathbb{R}^n$, and $\operatorname{rank}\mathbf{X}_{\{1,\ldots,p'\}} = p'$. Produce a piecewise linear function $\hat{\boldsymbol{\beta}}(\lambda)$ such that for any $\lambda \ge 0$,

$$\hat{\boldsymbol{\beta}}(\lambda) \in \arg\min_{\boldsymbol{\beta}} \left\{ \frac{1}{2}\left\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\right\|^2 + \lambda \sum_{j=p'+1}^{p} |\beta_j| \right\}$$

and the nonzero entries of $\hat{\boldsymbol{\beta}}(\lambda)$ identify regressors of full rank.

S1. [Initialize.] Set $\mathcal{A} \leftarrow (1,\ldots,p')$, $\mathbf{s} \leftarrow \mathbf{0}_{p'}$, $\lambda_1 \leftarrow \infty$, and $k \leftarrow 1$. Initialize $\mathbf{F}$ to a factor matrix for $\mathcal{A}$ using Algorithm A and put

$$\boldsymbol{\beta}_1 \leftarrow \texttt{complete}\left(\mathcal{A},\ \mathbf{R}_{\mathcal{A}}^{-1}\mathbf{Q}_{\mathcal{A}}'\mathbf{y}\right).$$

S2. [Loop.] Until $\lambda_k = 0$, do Step S3 and set $k \leftarrow k+1$. When done, go to Step S4.

S3. [Step solution.] Extract $\mathbf{R}_{\mathcal{A}}$, $\mathbf{Q}'_{\mathcal{A}}\mathbf{X}_{-\mathcal{A}}$, and $\mathbf{Q}'_{\mathcal{A}}\mathbf{y}$ from the factor matrix $\mathbf{F}$. Set

$$\boldsymbol{\gamma}(\lambda) \leftarrow \begin{pmatrix} \mathbf{R}_{\mathcal{A}}^{-1} & \mathbf{0} \end{pmatrix} \mathbf{Q}'_{\mathcal{A}}\mathbf{y} - \lambda \mathbf{R}_{\mathcal{A}}^{-1}\mathbf{R}_{\mathcal{A}}^{-1\,\prime}\mathbf{s},$$
$$\xi(\lambda) \leftarrow (\mathbf{Q}'_{\mathcal{A}}\mathbf{X}_{-\mathcal{A}})' \left[\begin{pmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \mathbf{Q}'_{\mathcal{A}}\mathbf{y} + \lambda \begin{pmatrix} \mathbf{R}_{\mathcal{A}}^{-1\,\prime} \\ \mathbf{0} \end{pmatrix} \mathbf{s} \right],$$
$$\lambda_{k+1},\ \texttt{ACT} \leftarrow \texttt{step}(\boldsymbol{\gamma}, \xi, \mathbf{s}, \lambda_k), \quad \text{and} \quad \boldsymbol{\beta}_{k+1} \leftarrow \texttt{complete}(\mathcal{A}, \boldsymbol{\gamma}(\lambda_{k+1})).$$

If $\texttt{ACT} = (\texttt{ACTIVATE}, j)$, add the regressor for j to $\mathcal{A}$, append $\operatorname{sign}\xi(\lambda_{k+1})_j$ to $\mathbf{s}$, and update $\mathbf{F}$ using Algorithm A.

If $\texttt{ACT} = (\texttt{DEACTIVATE}, j)$, remove the j-th entry from $\mathcal{A}$ and $\mathbf{s}$. Then update $\mathbf{F}$ using Algorithm D.

Return to Step S2.

S4. [Assemble solution.] Return $\hat{\boldsymbol{\beta}}(\lambda)$, the piecewise linear function that goes through the nodes $(\lambda_k, \boldsymbol{\beta}_k), \ldots, (\lambda_2, \boldsymbol{\beta}_2)$ and equals $\boldsymbol{\beta}_1$ for $\lambda > \lambda_2$.

5. LO-LARS Algorithm. In this section, we adapt Algorithm S to compute leave-one-out error paths and the LO function.

Given a solution path, $\hat{\boldsymbol{\beta}}(\lambda)$, define $T_{\hat{\boldsymbol{\beta}}}(\lambda) := \sum_{j=p'+1}^{p} |\hat{\boldsymbol{\beta}}(\lambda)_j|$. $T_{\hat{\boldsymbol{\beta}}}$ computes the partial ℓ_1 norm of a solution. Since $\hat{\boldsymbol{\beta}}$ is a piecewise parametric linear function of λ where the sign of a restricted regressor is constant on any segment, $T_{\hat{\boldsymbol{\beta}}}$ is also a piecewise linear function of λ with the same segments as $\hat{\boldsymbol{\beta}}$.

Suppose $B = (\hat{\boldsymbol{\beta}}^{(-1)}(\lambda), \ldots, \hat{\boldsymbol{\beta}}^{(-n)}(\lambda))$ is an array of solution paths such that $\hat{\boldsymbol{\beta}}^{(-i)}(\lambda)$ minimizes

$$f(\boldsymbol{\beta}; \lambda) - \frac{1}{2}\left(y^{(i)} - \mathbf{x}^{(i)\prime}\boldsymbol{\beta}\right)^2.$$

Here are two reasonable ways we might define LO for B:

$$\mathrm{LO}^{(1)}(\lambda; B) = \sum_{i=1}^{n} \left(y_i - \hat{\boldsymbol{\beta}}^{(-i)}(\lambda)\right)^2 \quad \text{or} \quad \mathrm{LO}^{(2)}(t; B) = \sum_{i=1}^{n} \left(y_i - \hat{\boldsymbol{\beta}}^{(-i)}(T^{-1}_{\hat{\boldsymbol{\beta}}^{(-i)}}(t))\right)^2.$$

$\mathrm{LO}^{(1)}$ keeps the penalty multiplier λ constant across leave-one-out subproblems, and $\mathrm{LO}^{(2)}$ keeps the ℓ_1 restriction constant across leave-one-out subproblems.

We can sketch out a high level algorithm to compute $\mathrm{LO}^{(1)}$ and $\mathrm{LO}^{(2)}$.

Algorithm L (*Compute LO*). Given $\mathbf{X} \in \mathbb{R}^{n\times p}$ and $\mathbf{y} \in \mathbb{R}^n$, compute $\mathrm{LO}^{(1)}$ and $\mathrm{LO}^{(2)}$ for some collection of leave-one-out solution paths B.

L1. [Produce error paths.] For each i, compute the segments of a piecewise linear leave-one-out error function,

$$\left(\lambda_1^{(i)}, t_1^{(i)}, e_1^{(i)}\right), \ldots, \left(\lambda_{m_i}^{(i)}, t_{m_i}^{(i)}, e_{m_i}^{(i)}\right), \qquad 1 \le i \le n$$

where $\infty = \lambda_1^{(i)} > \cdots > \lambda_{m_i}^{(i)} = 0$ and for some leave-one-out solution path $\hat{\boldsymbol{\beta}}^{(-i)}$,

$$\hat{\boldsymbol{\beta}}^{(-i)}(\lambda) \text{ is linear on } [\lambda_k^{(i)}, \lambda_{k+1}^{(i)}], \quad t_k^{(i)} = \sum_{j=p'+1}^{p} |\hat{\boldsymbol{\beta}}^{(-i)}(\lambda_k^{(i)})_j|, \quad \text{and}$$
$$e_k^{(i)} = y_i - \mathbf{x}_i' \hat{\boldsymbol{\beta}}^{(-i)}(\lambda_k^{(i)}).$$

L2. [Assemble LO.] Let $e^{(i)}(\lambda)$ denote the piecewise linear function that connects the points $\{(\lambda_k^{(i)}, e_k^{(i)})\}$ and let B denote the collection of solution paths from Step L1. Then

$$\mathrm{LO}^{(1)}(\lambda; B) = \sum_{i=1}^{n} e^{(i)}(\lambda)^2.$$

Similarly, we can connect the piecewise linear points $\{(t_k^{(i)}, e_k^{(i)})\}$ to compute $\mathrm{LO}^{(2)}$. ∎

To make Step L1 tractable, we need some of the computation of individual error paths to be shared.

Suppose

$$\begin{aligned}\boldsymbol{\gamma}^{(-i)}(\lambda) &= \left(\mathbf{X}_{\mathcal{A}}^{(-i)\prime}\mathbf{X}_{\mathcal{A}}^{(-i)}\right)^{-1}\left[\mathbf{X}_{\mathcal{A}}^{(-i)\prime}\mathbf{y}^{(-i)} - \lambda\mathbf{s}\right] \\ &= \left(\mathbf{X}_{\mathcal{A}}'\mathbf{X}_{\mathcal{A}} - \mathbf{x}_{\mathcal{A}}^{(i)}\mathbf{x}_{\mathcal{A}}^{(i)\prime}\right)^{-1}\left[\mathbf{X}_{\mathcal{A}}'\mathbf{y} - y^{(i)}\mathbf{x}_{\mathcal{A}}^{(i)} - \lambda\mathbf{s}\right]\end{aligned}$$

corresponds to a segment of a solution for the i-th leave-one-out lasso subproblem. Applying the matrix inversion lemma,

$$(\mathbf{A} + \mathbf{u}\mathbf{v}')^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}'\mathbf{A}^{-1}}{1 + \mathbf{v}'\mathbf{A}^{-1}\mathbf{u}},$$

gives us

$$\begin{aligned}\left(\mathbf{X}_{\mathcal{A}}'\mathbf{X}_{\mathcal{A}} - \mathbf{x}_{\mathcal{A}}^{(i)}\mathbf{x}_{\mathcal{A}}^{(i)\prime}\right)^{-1} &= (\mathbf{X}_{\mathcal{A}}'\mathbf{X}_{\mathcal{A}})^{-1} + \frac{(\mathbf{X}_{\mathcal{A}}'\mathbf{X}_{\mathcal{A}})^{-1}\mathbf{x}_{\mathcal{A}}^{(i)}\mathbf{x}_{\mathcal{A}}^{(i)\prime}(\mathbf{X}_{\mathcal{A}}'\mathbf{X}_{\mathcal{A}})^{-1}}{1 - \mathbf{x}_{\mathcal{A}}^{(i)\prime}(\mathbf{X}_{\mathcal{A}}'\mathbf{X}_{\mathcal{A}})^{-1}\mathbf{x}_{\mathcal{A}}^{(i)}} \\ &= (\mathbf{X}_{\mathcal{A}}'\mathbf{X}_{\mathcal{A}})^{-1} + \frac{\mathbf{w}_{\mathcal{A}}^{(i)}\mathbf{w}_{\mathcal{A}}^{(i)\prime}}{1 - h_{\mathcal{A}}^{(i)}}\end{aligned}$$

with $\mathbf{w}_{\mathcal{A}}^{(i)} = (\mathbf{X}_{\mathcal{A}}'\mathbf{X}_{\mathcal{A}})^{-1}\mathbf{x}_{\mathcal{A}}^{(i)}$ and $h_{\mathcal{A}}^{(i)} = \mathbf{x}_{\mathcal{A}}^{(i)\prime}(\mathbf{X}_{\mathcal{A}}'\mathbf{X}_{\mathcal{A}})^{-1}\mathbf{x}_{\mathcal{A}}^{(i)}$. Thus,

$$\boldsymbol{\gamma}^{(-i)}(\lambda) = \boldsymbol{\gamma}^{(-i)}(0) + \lambda\,\dot{\boldsymbol{\gamma}}^{(-i)}(0)$$

where

$$\begin{aligned}\boldsymbol{\gamma}^{(-i)}(0) &= \left[(\mathbf{X}_{\mathcal{A}}'\mathbf{X}_{\mathcal{A}})^{-1} + \frac{\mathbf{w}_{\mathcal{A}}^{(i)}\mathbf{w}_{\mathcal{A}}^{(i)\prime}}{1 - h_{\mathcal{A}}^{(i)}}\right]\left(\mathbf{X}_{\mathcal{A}}'\mathbf{y} - y^{(i)}\mathbf{x}_{\mathcal{A}}^{(i)}\right) \\ &= \boldsymbol{\gamma}(0) + \frac{\mathbf{w}_{\mathcal{A}}^{(i)}\mathbf{w}_{\mathcal{A}}^{(i)\prime}}{1 - h_{\mathcal{A}}^{(i)}}\mathbf{X}_{\mathcal{A}}'\mathbf{y} - (\mathbf{X}_{\mathcal{A}}'\mathbf{X})^{-1}\,y^{(i)}\mathbf{x}_{\mathcal{A}}^{(i)} - \frac{\mathbf{w}_{\mathcal{A}}^{(i)}\mathbf{w}_{\mathcal{A}}^{(i)\prime}}{1 - h_{\mathcal{A}}^{(i)}}y^{(i)}\mathbf{x}_{\mathcal{A}}^{(i)} \\ &= \boldsymbol{\gamma}(0) + \frac{\mathbf{x}_{\mathcal{A}}^{(i)\prime}\boldsymbol{\gamma}(0)}{1 - h_{\mathcal{A}}^{(i)}}\mathbf{w}_{\mathcal{A}}^{(i)} - y^{(i)}\mathbf{w}_{\mathcal{A}}^{(i)} - \frac{h_{\mathcal{A}}^{(i)}y^{(i)}}{1 - h_{\mathcal{A}}^{(i)}}\mathbf{w}_{\mathcal{A}}^{(i)} \\ &= \boldsymbol{\gamma}(0) - \frac{y^{(i)} - \mathbf{x}_{\mathcal{A}}^{(i)\prime}\boldsymbol{\gamma}(0)}{1 - h_{\mathcal{A}}^{(i)}}\mathbf{w}_{\mathcal{A}}^{(i)}\end{aligned}$$

and

$$\dot{\boldsymbol{\gamma}}^{(-i)}(0) = -\left[(\mathbf{X}_{\mathcal{A}}'\mathbf{X}_{\mathcal{A}})^{-1} + \frac{\mathbf{w}_{\mathcal{A}}^{(i)}\mathbf{w}_{\mathcal{A}}^{(i)\prime}}{1 - h_{\mathcal{A}}^{(i)}}\right]\mathbf{s} = \dot{\boldsymbol{\gamma}}(0) - \frac{\mathbf{w}_{\mathcal{A}}^{(i)\prime}\mathbf{s}}{1 - h_{\mathcal{A}}^{(i)}}\mathbf{w}_{\mathcal{A}}^{(i)}.$$

We can then express $\boldsymbol{\gamma}^{(-i)}$ in terms of an adjustment to $\boldsymbol{\gamma}$,

$$\boldsymbol{\gamma}^{(-i)}(\lambda) = \boldsymbol{\gamma}(\lambda) + \delta\boldsymbol{\gamma}^{(i)}(\lambda)$$

where

$$\begin{aligned}\delta\boldsymbol{\gamma}^{(i)}(\lambda) &= -\frac{y^{(i)} - \mathbf{x}_{\mathcal{A}}^{(i)\prime}\boldsymbol{\gamma}(0)}{1 - h_{\mathcal{A}}^{(i)}}\mathbf{w}_{\mathcal{A}}^{(i)} - \lambda\,\frac{\mathbf{w}_{\mathcal{A}}^{(i)\prime}\mathbf{s}}{1 - h_{\mathcal{A}}^{(i)}}\mathbf{w}_{\mathcal{A}}^{(i)} = a_{\mathcal{A}}^{(i)}(\lambda)\mathbf{w}_{\mathcal{A}}^{(i)} \\ \text{with}\quad a_{\mathcal{A}}^{(i)}(\lambda) &= -\frac{y^{(i)} - \mathbf{x}_{\mathcal{A}}^{(i)\prime}\boldsymbol{\gamma}(0)}{1 - h_{\mathcal{A}}^{(i)}} - \lambda\,\frac{\mathbf{w}_{\mathcal{A}}^{(i)\prime}\mathbf{s}}{1 - h_{\mathcal{A}}^{(i)}}.\end{aligned}$$

Similarly, we have

$$\begin{aligned}\xi^{(-i)}(\lambda) &= \mathbf{X}_{-\mathcal{A}}^{(-i)\prime}\left[\mathbf{y}^{(-i)} - \mathbf{X}_{\mathcal{A}}^{(-i)}\boldsymbol{\gamma}^{(-i)}(\lambda)\right] \\ &= \mathbf{X}_{-\mathcal{A}}'\left[\mathbf{y} - \mathbf{X}_{\mathcal{A}}\boldsymbol{\gamma}^{(-i)}(\lambda)\right] - \left[y^{(i)} - \mathbf{x}_{\mathcal{A}}^{(i)\prime}\boldsymbol{\gamma}^{(-i)}(\lambda)\right]\mathbf{x}_{-\mathcal{A}}^{(i)} \\ &= \xi(\lambda) + \delta\xi^{(i)}(\lambda)\end{aligned}$$

with

$$\delta\xi^{(i)}(\lambda) = -\mathbf{X}_{-\mathcal{A}}'\mathbf{X}_{\mathcal{A}}\delta\boldsymbol{\gamma}^{(i)}(\lambda) - \left[y^{(i)} - \mathbf{x}_{\mathcal{A}}^{(i)\prime}\boldsymbol{\gamma}^{(-i)}(\lambda)\right]\mathbf{x}_{-\mathcal{A}}^{(i)}.$$

If $\mathbf{Q}_{\mathcal{A}}$ and $\mathbf{R}_{\mathcal{A}}$ denote the QR factorization of $\mathbf{X}_{\mathcal{A}}$, then we can write the equations as

$$h_{\mathcal{A}}^{(i)} = \left\|\mathbf{R}_{\mathcal{A}}^{-1\prime}\mathbf{x}_{\mathcal{A}}^{(i)}\right\|^2, \qquad \mathbf{w}_{\mathcal{A}}^{(i)} = \mathbf{R}_{\mathcal{A}}^{-1}\mathbf{R}_{\mathcal{A}}^{-1\prime}\mathbf{x}_{\mathcal{A}}^{(i)}, \quad \text{and}$$

$$\delta\xi^{(i)}(\lambda) = -a_{\mathcal{A}}^{(i)}(\lambda)\left(\mathbf{Q}_{\mathcal{A}}'\mathbf{X}_{-\mathcal{A}}\right)'\begin{pmatrix}\mathbf{R}_{\mathcal{A}}^{-1\prime} \\ \mathbf{0}\end{pmatrix}\mathbf{x}_{\mathcal{A}}^{(i)} - \left[y^{(i)} - \mathbf{x}_{\mathcal{A}}^{(i)\prime}\boldsymbol{\gamma}^{(-i)}(\lambda)\right]\mathbf{x}_{-\mathcal{A}}^{(i)}.$$

The above equations show that $\boldsymbol{\gamma}^{(-i)}$ and $\xi^{(-i)}$ be computed as updates to $\boldsymbol{\gamma}$ and ξ. And given the factor matrix $\mathbf{F}_{\mathcal{A}}$, $\boldsymbol{\gamma}$, and ξ, the leave-one-out functions $\boldsymbol{\gamma}^{(-i)}$ and $\xi^{(-i)}$ can be computed big-O of

$$\big(\text{num active regressors}\big) \times \big(\text{num regressors}\big).$$

Thus, when leave-one-out path segments share the same set of active regressors, they can be computed with much lower cost than if treated independently. To capitalize on this observation, we can maintain a computation cache that maps a set of active regressors to a shared factor matrix. A natural data structure to use is a hash map indexed with bitsets to represent active regressors. Then we can maintain a second level of caching that maps a pair $(\mathcal{A}, \mathbf{s})$ to common base parametric equations $\boldsymbol{\gamma}$ and ξ where $\mathcal{A}$ is a vector of active regressors and $\mathbf{s}$ is a sign vector. We can access and update such a cache with the functions

$$\texttt{lookup}(\texttt{CACHE}, \mathcal{A}, \mathbf{s}) := \begin{cases}\text{Let } (\mathcal{A}', \mathbf{s}') \text{ denote a paired active regressor and sign vector in } \texttt{CACHE} \text{ that matches } (\mathcal{A}, \mathbf{s}) \text{ up to a reordering.} \\ \text{Let } \mathbf{F}_{\mathcal{A}'} \text{ denote the cached factor matrix for } \mathcal{A}' \text{ and let } (\boldsymbol{\gamma}_{\mathcal{A}',\mathbf{s}'},\ \xi_{\mathcal{A}',\mathbf{s}'}) \text{ denote the cached parametric equations for } (\mathcal{A}', \mathbf{s}') \text{ as defined in Step 3 of Algorithm S.} \\ \text{Return } (\mathcal{A}',\ \mathbf{s}',\ \mathbf{F}_{\mathcal{A}'},\ \boldsymbol{\gamma}_{\mathcal{A}',\mathbf{s}'},\ \xi_{\mathcal{A}',\mathbf{s}'}).\end{cases}$$

and

$$\texttt{update}(\texttt{CACHE}, \mathcal{A}, \mathbf{s}, \mathbf{F}_{\mathcal{A}}, \texttt{ACT}, s') :=$$

$$\begin{cases}\text{If } \texttt{ACT} = (\texttt{ACTIVATE}, j), \text{ then add the regressor for } j \text{ to } \mathcal{A} \text{ and append } s' \text{ to } \mathbf{s}. \text{ If there is not already an entry for } \mathcal{A} \text{ in } \texttt{CACHE}, \text{ copy } \mathbf{F}_{\mathcal{A}} \text{ if necessary, update using Algorithm A, and store in } \texttt{CACHE}. \\ \text{If } \texttt{ACT} = (\texttt{DEACTIVATE}, j), \text{ remove the } j\text{-th entry from } \mathcal{A} \text{ and } \mathbf{s}. \text{ If there is not already an entry for } \mathcal{A} \text{ in } \texttt{CACHE}, \text{ copy } \mathbf{F}_{\mathcal{A}} \text{ if necessary, update using Algorithm D, and store in } \texttt{CACHE}. \\ \text{Rearrange } \mathcal{A} \text{ and } \mathbf{s} \text{ to match the ordering used for its cached factor matrix. If there is not already an entry for the pair } (\mathcal{A}, \mathbf{s}), \text{ compute } (\boldsymbol{\gamma}_{\mathcal{A},\mathbf{s}},\ \xi_{\mathcal{A},\mathbf{s}}) \text{ using the equations from Step 3 of Algorithm S and store in } \texttt{CACHE}. \\ \text{Return } (\mathcal{A}, \mathbf{s}).\end{cases}$$

Let's now flesh out Step L1 in greater detail.

Algorithm E (*Compute leave-one-out error segments*). Given $\mathbf{X} \in \mathbb{R}^{n \times p}$ and $\mathbf{y} \in \mathbb{R}^n$ such that $\operatorname{rank} \mathbf{X}_{\{1,\ldots,p'\}} = p'$, compute the leave-one-out error segments,

$$\left(\lambda_1^{(i)}, t_1^{(i)}, e_1^{(i)}\right), \ldots, \left(\lambda_{m_i}^{(i)}, t_{m_i}^{(i)}, e_{m_i}^{(i)}\right), \qquad 1 \le i \le n$$

described in Algorithm L.

E1. [Initialize.] Set $\mathcal{A}_0 \leftarrow (1, \ldots, p')'$ and $k \leftarrow 1$. Compute a factor matrix $\mathbf{F}_{\mathcal{A}_0}$ for A_0 using Algorithm A. Initialize `CACHE` to be empty; then add an entry for the pair $(\mathcal{A}_0, \mathbf{F}_{\mathcal{A}_0})$. For $i = 1, \ldots, n$, set

$$\mathcal{A}^{(i)} \leftarrow \mathcal{A}_0, \quad \mathbf{s}^{(i)} \leftarrow \mathbf{0}_{p'}, \quad \lambda_1^{(i)} = \infty, \quad \text{and} \quad t_1^{(i)} \leftarrow 0.$$

E2. [Main loop.] Until $\lambda_k^{(i)} = 0$ for all i, do Step E3 and set $k \leftarrow k + 1$.

E3. [Loop i.] Initialize `CACHE′` to be empty. For $i = 1, \ldots, n$, do Step E4. When done, set `CACHE` $\leftarrow$ `CACHE′` and return to Step E2.

E4. [Extend error path.] If $\lambda_k^{(i)} = 0$, return to Step E3. Otherwise, set

$$\mathcal{A},\ \mathbf{s},\ \mathbf{F}_{\mathcal{A}},\ \boldsymbol{\gamma},\ \xi \leftarrow \texttt{lookup}(\texttt{CACHE}, \mathcal{A}^{(i)}, \mathbf{s}^{(i)}).$$

Extract $\mathbf{R}_{\mathcal{A}}$ and $\mathbf{Q}'_{\mathcal{A}}\mathbf{X}_{-\mathcal{A}}$ from $\mathbf{F}_{\mathcal{A}}$.

Set $\mathbf{u} \leftarrow \mathbf{R}_{\mathcal{A}}^{-1\prime}\mathbf{x}_{\mathcal{A}}^{(i)}$, $h \leftarrow \|\mathbf{u}\|^2$, $\mathbf{w} \leftarrow \mathbf{R}_{\mathcal{A}}^{-1}\mathbf{u}$. Then put

$$a(\lambda) \leftarrow -\frac{y^{(i)} - \mathbf{x}_{\mathcal{A}}^{(i)\prime}\boldsymbol{\gamma}(0)}{1-h} - \lambda\,\frac{\mathbf{w}'\mathbf{s}}{1-h}, \qquad \boldsymbol{\gamma}^{(-i)}(\lambda) \leftarrow \boldsymbol{\gamma}(\lambda) + a(\lambda)\mathbf{w}, \quad \text{and}$$
$$\xi^{(-i)}(\lambda) \leftarrow \xi(\lambda) - a(\lambda)\,(\mathbf{Q}'_{\mathcal{A}}\mathbf{X}_{-\mathcal{A}})'\begin{pmatrix}\mathbf{u}\\ \mathbf{0}\end{pmatrix} - \left[y^{(i)} - \mathbf{x}_{\mathcal{A}}^{(i)\prime}\boldsymbol{\gamma}^{(-i)}(\lambda)\right]\mathbf{x}_{-\mathcal{A}}^{(i)}.$$

Put $\lambda_{k+1}^{(i)}, \texttt{ACT} \leftarrow \texttt{step}(\boldsymbol{\gamma}^{(-i)}, \xi^{(-i)}, \mathbf{s}, \lambda_k^{(i)})$ and $\hat{\boldsymbol{\beta}} \leftarrow \boldsymbol{\gamma}^{(-i)}(\lambda_{k+1}^{(i)})$.

If $k = 1$, set $e_1^{(i)} \leftarrow y^{(i)} - \mathbf{x}_{\mathcal{A}}^{(i)\prime}\left(\boldsymbol{\gamma}^{(-i)}(0)\right)$.

Set $e_{k+1}^{(i)} \leftarrow y^{(i)} - \mathbf{x}_{\mathcal{A}}^{(i)\prime}\hat{\boldsymbol{\beta}}$ and $t_{k+1}^{(i)} \leftarrow \sum_{j=p'+1}^{p} |\hat{\beta}_j|$.

If $\texttt{ACT} = (\texttt{ACTIVATE}, j)$, set $s' = \operatorname{sign} \xi^{(-i)}(\lambda_{k+1}^{(i)})_j$; otherwise, set $s' = 0$.

If $\texttt{ACT} \neq \texttt{NULL}$, invoke

$$\mathcal{A}^{(i)},\ \mathbf{s}^{(i)} \leftarrow \texttt{update}(\texttt{CACHE}', \mathcal{A}, \mathbf{s}, \mathbf{F}_{\mathcal{A}}, \texttt{ACT}, s').$$

Return to Step E3. ∎

The cost of Algorithm E depends heavily on the extent of divergence in the leave-one-out solution paths. The more individual solution paths overlap in their active regressors, the more efficient Algorithm E will be. We can make several modifications to improve performance.

Update batching. As multiple solution paths can share the same factor matrix, we need to ensure that we don't modify a factor matrix that's used by another path. We can copy factor matrices before updating, but that leads to additional overhead. However, if we rework Step E4 so that the updates are done as a batch at the end of the loop in Step E3, then we can elide the unnecessary copies.

Early stopping. With high-dimensional data sets, solution paths can diverge substantially when there is little shrinkage (i.e. t is large). In such cases, it can be beneficial to modify Algorithm L so that instead of computing the entire LO function, Algorithm L exits early if LO exceeds a specified percentage of the best optimum found so far. We can make this possible by merging Step L1 with Step L2 so that the LO curve is built out as segments become available. We then track the best local optimum of the curve and regularly monitor LO to see if the exit condition is met.

6. Numerical Experiments. Consider a linear model

$$y = \mathbf{x}'\boldsymbol{\beta}_{\text{true}} + \varepsilon, \qquad \mathbf{x} \sim N(\mathbf{0}_p, \Sigma), \quad \text{and} \quad \varepsilon \sim N(0, \sigma^2).$$

Suppose $\hat{\boldsymbol{\beta}}$ denotes an estimator for the parameter $\boldsymbol{\beta}_{\text{true}}$ where $\hat{\boldsymbol{\beta}}$ maps sample observations $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ to an estimated regressor. For a given $\boldsymbol{\beta}$, we will have an expected out-of-sample prediction error of

$$\begin{aligned}
\text{pe}(\boldsymbol{\beta}) &= \mathbb{E}\left[(\mathbf{x}'(\boldsymbol{\beta} - \boldsymbol{\beta}_{\text{true}}) - \varepsilon)^2 \mid \boldsymbol{\beta}\right] \\
&= \mathbb{E}\left[(\boldsymbol{\beta} - \boldsymbol{\beta}_{\text{true}})'\mathbf{x}\mathbf{x}'(\boldsymbol{\beta} - \boldsymbol{\beta}_{\text{true}}) - 2\varepsilon\mathbf{x}'(\boldsymbol{\beta} - \boldsymbol{\beta}_{\text{true}}) + \varepsilon^2 \mid \boldsymbol{\beta}\right] \\
&= (\boldsymbol{\beta} - \boldsymbol{\beta}_{\text{true}})'\Sigma(\boldsymbol{\beta} - \boldsymbol{\beta}_{\text{true}}) + \sigma^2.
\end{aligned}$$

Now suppose that we estimate the prediction error of $\hat{\boldsymbol{\beta}}$ with $\widehat{\text{pe}}$ using the same observations. Then the bias and variance of $\widehat{\text{pe}}$ are given by

$$\mathbb{E}\left[\widehat{\text{pe}} - \text{pe}(\hat{\boldsymbol{\beta}})\right] \quad \text{and} \quad \mathbb{V}\left[\widehat{\text{pe}} - \text{pe}(\hat{\boldsymbol{\beta}})\right].$$

Perhaps the simplest $\widehat{\text{pe}}$ is the holdout error estimator, where one fraction of the data is used for training and the other fraction is used for error estimation. We can expect holdout to have a bias as the holdout training data set is smaller than the data set used to fit the actual model. For the case of OLS, we can work out a simple equation for the bias. Letting $\hat{\boldsymbol{\beta}}^{\text{OLS}}$ denote the OLS estimator and setting $\mathbf{e} \leftarrow \mathbf{y} - \mathbf{X}\boldsymbol{\beta}_{\text{true}}$, we have

$$\begin{aligned}
\hat{\boldsymbol{\beta}}^{\text{OLS}} - \boldsymbol{\beta}_{\text{true}} &= (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{e}, \\
\text{pe}(\hat{\boldsymbol{\beta}}^{\text{OLS}}) &= \mathbf{e}'\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\Sigma(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{e} + \sigma^2 \\
&= \text{tr}\left[(\mathbf{X}'\mathbf{X})^{-1}\Sigma(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{e}\mathbf{e}'\mathbf{X}\right] + \sigma^2, \quad \text{and} \\
\mathbb{E}\left[\text{pe}(\hat{\boldsymbol{\beta}}^{\text{OLS}}) \mid \mathbf{X}\right] &= \sigma^2\,\text{tr}\left[(\mathbf{X}'\mathbf{X})^{-1}\Sigma\right] + \sigma^2.
\end{aligned}$$

Now $(\mathbf{X}'\mathbf{X})^{-1}$ follows an inverse Wishart distribution with n degrees of freedom [1, §7.7] and has expected value $\mathbb{E}\left[(\mathbf{X}'\mathbf{X})^{-1}\right] = \frac{\Sigma^{-1}}{n-p-1}$. Thus,

$$\mathbb{E}\left[\text{pe}(\hat{\boldsymbol{\beta}}^{\text{OLS}})\right] = \frac{\sigma^2 p}{n-p-1} + \sigma^2.$$

So if we estimate the prediction error for a data set of n observations using holdout with a training set of size n', the bias will be

$$\frac{\sigma^2 p}{n'-p-1} - \frac{\sigma^2 p}{n-p-1} = \sigma^2 p \frac{n-n'}{(n'-p-1)(n-p-1)}.$$

Clearly, bias decreases with n' closer to n. But also interesting to note is the role of p: with p close to n, a larger holdout set becomes more important for reducing bias and even a slightly larger holdout set can lead to a substantial reduction in bias.

Cross-validation averages over multiple holdout estimates. While cross-validation won't improve on the bias of the holdout estimators (assuming the folds are of equal size), it can reduce the variance significantly. We can use Monte Carlo simulations to explore how well LO and other forms of v-fold cross-validation perform as estimators for the lasso.

Simulation B. For our first simulation, we'll set $n = 50$, $p = 40$, and assume the covariance matrix Σ has a Toeplitz structure with $\Sigma_{ij} = 0.5^{|i-j|}$. We set

$$(\boldsymbol{\beta}_{\text{true}})_j = \begin{cases} \frac{1}{20}, & \text{if } j \leq 20; \\ 0, & \text{otherwise;} \end{cases} \quad \text{and} \quad \sigma^2 = \boldsymbol{\beta}_{\text{true}}'\Sigma\boldsymbol{\beta}_{\text{true}} = 0.14$$

so that the model has a signal-to-noise ratio of 1. The table below shows the performance of v-fold cross-validation for various values of v and t over 100,000 instances.

	Num	Prediction	5-fold		10-fold		LO	
t	Active	Error	Bias	Var	Bias	Var	Bias	Var
0.0	0.0	0.280	0.00012	0.00314	0.00012	0.00314	0.00012	0.00314
0.1	3.3	0.258	0.00095	0.00280	0.00051	0.00282	0.00021	0.00285
0.2	5.3	0.240	0.00221	0.00265	0.00109	0.00265	0.00029	0.00267
0.3	7.2	0.226	0.00354	0.00255	0.00171	0.00253	0.00040	0.00254
0.4	9.1	0.215	0.00486	0.00250	0.00232	0.00246	0.00052	0.00245
0.5	10.9	0.206	0.00612	0.00250	0.00290	0.00244	0.00062	0.00242
0.6	12.6	0.200	0.00729	0.00254	0.00343	0.00246	0.00071	0.00244
0.7	14.4	0.197	0.00837	0.00263	0.00391	0.00254	0.00082	0.00251
0.8	16.1	0.195	0.00939	0.00277	0.00437	0.00266	0.00094	0.00264
0.9	17.7	0.196	0.01036	0.00295	0.00483	0.00284	0.00106	0.00283
1.0	19.3	0.198	0.01133	0.00317	0.00529	0.00307	0.00119	0.00307
1.1	20.8	0.202	0.01234	0.00345	0.00575	0.00334	0.00133	0.00336
1.2	22.2	0.207	0.01339	0.00376	0.00624	0.00366	0.00146	0.00371
1.3	23.5	0.212	0.01453	0.00413	0.00675	0.00403	0.00158	0.00411
1.4	24.7	0.218	0.01574	0.00454	0.00729	0.00445	0.00170	0.00456
1.5	25.8	0.225	0.01704	0.00499	0.00787	0.00491	0.00183	0.00507
1.6	26.8	0.232	0.01842	0.00549	0.00848	0.00543	0.00195	0.00563
1.7	27.8	0.240	0.01987	0.00604	0.00913	0.00601	0.00209	0.00625
1.8	28.7	0.248	0.02140	0.00664	0.00981	0.00664	0.00223	0.00693
1.9	29.5	0.256	0.02300	0.00729	0.01051	0.00734	0.00239	0.00769
2.0	30.3	0.265	0.02469	0.00799	0.01126	0.00810	0.00255	0.00853

Table 6. *Results of a Monte Carlo simulation to measure the bias and variance of v-fold cross-validation as an estimator for lasso regression prediction error for $v = 5, 10, n$ and various values of the shrinkage parameter t. Also shown for each t are the average number of active regressors and the average prediction error.*

As we would expect, the bias of LO is notably lower than the bias of 5-fold and 10-fold cross-validation. And like OLS, the differences in the biases are more pronounced with more active regressors. The differences in the variances are all minor.

For the next simulation, we investigate the accuracy of estimating the optimal value of t in terms of prediction error by optimizing LO.

Simulation O. We use the same values for n, p, and Σ as Simulation B. For each simulated data set, an oracle value t_{opt} is selected so as to optimize the true prediction error,

$$t_{\text{opt}} = \arg\min_t \left\{ \text{pe}(\hat{\boldsymbol{\beta}}(t)) \right\},$$

and an estimated value $\hat{t}_{\text{opt}}$ is selected so as to optimize LO. The table below summarizes the performance of the estimator over 10,000 instances.

	t	Num Active	Prediction Error
oracle optimum	0.819	15.93	0.194
LO optimum	0.959	15.62	0.217
delta (LO − oracle)	0.140	-0.31	0.024
stddev of delta	0.640	7.54	0.063

Table 7. *Results of a Monte Carlo simulation to measure the accuracy of using the LO minimizer to estimate the prediction error minimizer. Also shown are the average number of active regressors, the average prediction error of the LO minimum, and the average actual prediction error minimum.*

While $\hat{t}_{\text{opt}}$ performs decently, it does appear to have a bias, suggesting that there might be a better way to estimate the optimum.

7. Conclusions. LO can be an effective way to estimate prediction error for the lasso. It has lower bias than other forms of v-fold cross-validation, particularly for high-dimensional data sets; and as we saw in Simulation B, the variance of LO can be nearly identical to that of 10-fold and 5-fold cross-validation. The LO-LARS algorithm makes exact computation of LO practical for many real-world data sets and enables fully deterministic and reproducible selection of the lasso shrinkage parameter.

While I think LO is an excellent default choice for estimating prediction error, there are certain structured data sets, such as when $\mathbf{X}$ is near diagonal, for which LO is poorly suited. In such cases, *generalized cross-validation* (GCV) can sometimes provide better performance. GCV is closely related to LO; only instead of computing LO on the observations $(\mathbf{X}, \mathbf{y})$, GCV first rotates the observations using a complex unitary matrix $\mathbf{U}$,

$$\tilde{\mathbf{X}} = \mathbf{U}\mathbf{X}, \qquad \tilde{\mathbf{y}} = \mathbf{U}\mathbf{y},$$

where $\mathbf{U}$ is chosen to make $\tilde{\mathbf{X}}\tilde{\mathbf{X}}^H$ circulant; then GCV computes LO on the rotated observations $(\tilde{\mathbf{X}}, \tilde{\mathbf{y}})$. The transformation step makes GCV invariant to rotations of the observations

and can fix certain problem cases with LO (see [9]). As $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{y}}$ are complex matrices, the algorithms developed in this paper are not immediately applicable, but one might hope that there are versions that could work.

References

[1] Theodore Anderson. An introduction to multivariate statistical analysis, third edition. *Wiley-Interscience,* 2003.

[2] Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. *Statistical Surveys,* 4:40–79, 2010.

[3] Pierre Bayle, Alexandre Bayle, Lucas Janson, and Lester Mackey. Cross-validation confidence intervals for test error. *Advances in Neural Information Processing Systems,* 33:16339–16350, 2020.

[4] Peter Bühlmann, Karkus Kalisch, and Lukas Meier. High-dimensional statistics with a view towards applications in biology. *Annual Review of Statistics and its Application,* 1:255–278, 2014.

[5] Prabir Burman. A comparative study of ordinary cross-validation, v-fold cross-validation and the repeated learning-testing methods. *Biometrika,* 76(3):503–514, 1989.

[6] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *Annals of Statistics,* 32(2):407–499, 2004.

[7] Santosa Fadil and Symes W. William. Linear inversion of band-limited reflection seismograms. *SIAM Journal on Scientific and Statistical Computing,* 7(4):1307–1330, 1986.

[9] Gene Golub, Michael Heath, and Grace Wahba. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics,* 21(2):215–223, 1979.

[8] Gene H. Golub and Charles F. Van Loan. Matrix computations, fourth edition. *The John Hopkins University Press,* 2013.

[10] Jian-Ming Lee, Shehui Zhang, Soumitra Saha, Sonia Anna, Can Jiang, and John Perkins. RNA expression analysis using an antisense *Bacillus subtilis* genome array. *Journal of Bacteriology,* 183(24):7371–7380, 2001.

[11] Michael Osborne, Brett Presnell, and Berwin Turlach. On the lasso and its dual. *Journal of Computational and Graphical Statistics,* 9(2):319–337, 2000.

[12] Kamiar R. Rad and Arian Maleki. A scalable estimate of the out-of-sample prediction error via approximate leave-one-out cross-validation. *Journal of the Royal Statistical Society Series B,* 82(4):965–996, 2020.

[13] Kamiar R. Rad, Wenda Zhou, and Arian Maleki. Error bounds in estimating the out-of-sample prediction error using leave-one-out cross-validation in high-dimensions. *International Conference on Artificial Intelligence and Statistics,* PMLR 2424–2434, 2020.

[14] Carl Rasmussen and Christopher Williams. Gaussian processes for machine learning. *The MIT Press,* 2006.

[15] Ralph Rockafellar. Convex analysis. *Princeton University Press,* 1970.

[16] Saharon Rosset, Ji Zhu, and Trevor Hastie. Boosting as a regularized path to a maximum margin classifier. *Journal of Machine Learning Research,* 5:941–973, 2004.

[17] Seth Stein and Michael Wysession. An introduction to seismology, earthquakes, and earth structure. *Blackwell Publishing,* 2003.

[18] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society B,* 58(1):267–288, 1996.

[19] Ryan J. Tibshirani. The lasso problem and uniqueness. *Electronic Journal of Statistics,* 7:1456–1490, 2013.

[20] Ryan Tibshirani and Jonathan Taylor. The solution path of the generalized lasso. *Annals of Statistics,* 39(3):1335–1371, 2011.

[21] Sethu Vijayakumar and Stefan Schaal. Locally weighted projection regression: an O(n) algorithm for incremental real time learning in high dimensional space. *Proc. ICML 2000,* 1079–1086, 2000.

[22] Shuaiwen Wang, Wenda Zhou, Haihao Lu, Arian Maleki, and Vahab Mirrokni. Approximate leave-one-out for fast parameter tuning in high dimensions. *Proceedings of the 35th International Conference on Machine Learning,* PMLR 80:5228–5237, 2018.

[23] Haolin Zou, Arnab Auddy, Kamiar R. Rad, and Arian Maleki. Theoretical analysis of leave-one-out cross-validation for non-differentiable penalties under high-dimensional settings. *arXiv preprint arXiv:2402.08543,* 2024.

A Benchmarks

This appendix provides source code to run the benchmarks referenced in §2. The benchmarks use the python package `bbai` version 1.13.0 (`https://github.com/rnburn/bbai`) to run the LO-LARS algorithm, and all benchmarks share the common timing code below.

```
from bbai.glm import Lasso
```

```
import time
import numpy as np

def bench(X, y, early_exit=np.inf):
  model = Lasso(fit_intercept=True, loo_mode='t',
              early_exit_threshold=early_exit)

  # ignore the first run
  model.fit(X, y)

  # average time to fit over 5 runs
  elapse = 0
  N = 5
  for _ in range(N):
    t1 = time.time()
    model.fit(X, y)
    elapse += time.time() - t1
  elapse /= N

  print('avg elapse:', elapse)
```

Diabetes. The diabetes data set measures diabetes progression in 442 patients over the course of a year. It comes from [6] and is available from scikit-learn.

```
from sklearn.datasets import load_diabetes
from sklearn.preprocessing import StandardScaler
import numpy as np

data = load_diabetes()
X = data['data']
y = data['target']
n, p = X.shape

# standard benchmark
bench(StandardScaler().fit_transform(X), y)

# quadratic benchmark
pp = p * (p-1)//2 - 1 + 2*p
Xp = np.zeros((n, pp))
cnt = 0
for j, f in enumerate(data['feature_names']):
  Xp[:, cnt] = X[:, j]
  cnt += 1
  # don't include the squared term for sex
  if f != 'sex':
    Xp[:, cnt] = X[:, j]**2
    cnt += 1
  for jp in range(j+1, p):
    Xp[:, cnt] = X[:, j] * X[:, jp]
    cnt += 1
bench(StandardScaler().fit_transform(X), y)
```

California housing. This data set provides median house value for different districts in California. It is available from scikit-learn.

```
from sklearn.datasets import fetch_california_housing
from sklearn.preprocessing import StandardScaler
from common import bench
import numpy as np

X, y = fetch_california_housing(return_X_y=True)
n, p = X.shape

# standard benchmark
bench(StandardScaler().fit_transform(X), y)

# quadratic benchmark
pp = p*(p-1)//2 + 2*p
cnt = 0
Xp = np.zeros((n, pp))
for j in range(p):
  Xp[:, cnt] = X[:, j]
```

```
  cnt += 1
  for jp in range(j, p):
    Xp[:, cnt] = X[:, j] * X[:, jp]
    cnt += 1
X = Xp
bench(StandardScaler().fit_transform(X), y)
```

Riboflavin. This data set measures riboflavin production in bacteria cultures. It original comes from [10] and was made available by [4]. The data can be downloaded as a CSV file from http://www.annualreviews.org/doi/suppl/10.1146/annurev-statistics-022513-115545.

```
import pandas as pd
import numpy as np
from common import bench

df = pd.read_csv('riboflavin.csv')
y = df.iloc[0, 1:].to_numpy()
X = df.iloc[1:, 1:].to_numpy().T
# Note:  following [4], the data is assumed to
#        already be on the same scale

# standard benchmark
bench(X, y)

# early exit benchmark
bench(X, y, early_exit=0.01)
```

SARCOS Inverse. This data set relates to an inverse dynamics problem for a robot arm. The data set originally comes from [21] and is made available by [14] as a MAT-file at https://gaussianprocess.org/gpml/data/.

```
import scipy.io
import numpy as np
from sklearn.preprocessing import StandardScaler
from common import bench

mat = scipy.io.loadmat('sarcos_inv.mat')
X = mat['sarcos_inv']
y = X[:, 21]
X = X[:, :21]
n, p = X.shape

# standard benchmark
bench(StandardScaler().fit_transform(X), y)

# quadratic benchmark
pp = p*(p-1)//2 + 2*p
cnt = 0
Xp = np.zeros((n, pp))
for j in range(p):
  Xp[:, cnt] = X[:, j]
  cnt += 1
  for jp in range(j, p):
    Xp[:, cnt] = X[:, j] * X[:, jp]
    cnt += 1
X = Xp
bench(StandardScaler().fit_transform(X), y)

# quadratic, early exit benchmark
bench(StandardScaler().fit_transform(X), y, early_exit=0.01)
```