Journal of Statistical Software

March 2011, Volume 39, Issue 5. *http://www.jstatsoft.org/*

Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent

Noah Simon
Stanford University

Jerome Friedman
Stanford University

Trevor Hastie
Stanford University

Rob Tibshirani
Stanford University

Abstract

We introduce a pathwise algorithm for the Cox proportional hazards model, regularized by convex combinations of ℓ_1 and ℓ_2 penalties (elastic net). Our algorithm fits via cyclical coordinate descent, and employs warm starts to find a solution along a regularization path. We demonstrate the efficacy of our algorithm on real and simulated data sets, and find considerable speedup between our algorithm and competing methods.

Keywords: survival, Cox model, lasso, elastic net.

1. Introduction

Consider the usual survival analysis framework. We have data of the form $(y_1, x_1, \delta_1), \ldots, (y_n, x_n, \delta_n)$ where y_i, the observed time, is a time of failure if δ_i is 1 or right-censoring if δ_i is 0. As in regression, $\mathbf{x}_i$ is a vector of potential predictors $(x_{i,1}, x_{i,2}, \ldots, x_{i,p})$. We further let $t_1 < t_2 < \ldots < t_m$ be the increasing list of unique failure times, and $j(i)$ denote the index of the observation failing at time t_i. One potential problem of interest is to study the relationship between predictor variables and survival time. Commonly, the Cox proportional hazards model (Cox 1972) is used to approach this problem. The Cox model assumes a semi-parametric form for the hazard

$$h_i(t) = h_0(t)e^{x_i^\top \beta}$$

where $h_i(t)$ is the hazard for patient i at time t, $h_0(t)$ is a shared baseline hazard, and β is a fixed, length p vector. Inference is then made via the partial likelihood

$$L(\beta) = \prod_{i=1}^{m} \frac{e^{x_{j(i)}^\top \beta}}{\sum_{j \in R_i} e^{x_j^\top \beta}}$$

where R_i is the set of indices, j, with $y_j \geq t_i$ (those at risk at time t_i). Inference made with the partial likelihood ignores all information between failure times. For ease of notation the above formula assumes that the y_i are unique, however it can be suitably altered for the case of ties (see Section 2.5). By maximizing the partial likelihood, one can estimate β. The beauty of this approach is that it allows estimation of β while ignoring h_0. For classical problems, with many more observations than predictors, the Cox model performs well. However, problems with $p > n$, lead to degenerate behavior; to maximize the partial likelihood, all of the β_i are sent to $\pm\infty$. To combat this problem, Tibshirani (1997) proposed the use of an L1 (lasso) penalty in the Cox model. This both provides a well defined solution, and a solution with few nonzero β_i. Even in the $n > p$ case, if p is sufficiently close to n, this may better estimate β than the unpenalized Cox model. Gui and Li (2005) developed an algorithm to fit this model using Newton Raphson approximations and the lasso path solution to the penalized least squares problem provided by the adjusted LARS (least angle regression) solution of Efron *et al.* (2004).

More recently Zou and Hastie (2005) proposed the elastic net for linear regression; to maximize the likelihood subject to the constraint $\alpha \sum |\beta_i| + (1-\alpha) \sum \beta_i^2 \leq c$. $\alpha = 1$ gives the lasso penalty. Park and Hastie (2007a) applied this to the Cox model and proposed an efficient algorithm to solve this problem along a path of c and α values. Their algorithm exploits the near piecewise linearity of the coefficients to approximate the solution at different constraints, then numerically maximizes the likelihood for each constraint via a Newton iteration initialized at the approximation. Goeman (2010a) also attacked this problem. He developed a hybrid algorithm, combining gradient descent and Newton's method.

In this paper we instead employ cyclical coordinate descent. This method has been applied to penalized regression and in particular, elastic net penalties; recently by Friedman *et al.* (2010), van der Kooij (2007) and Wu and Lange (2008). Friedman *et al.* (2010) also recognized the strength of employing warm starts to solve the problem along a path of constraint values.

We build on the work of Friedman *et al.* (2010) and develop a fast algorithm to fit the Cox model with elastic net penalties. The time-ordered structure of the partial likelihood required the development of special risk set updates for the procedure. In addition, we present a method for selecting a well behaved path of λ values. We further include a number of checks on deviance and step size to make efficient use of CPU time. We show via simulation that our algorithm is both efficient and stable for small and large problems. In time trials we show that it is significantly faster than the Hastie-Park algorithm and the Goeman algorithm. Our algorithm scales efficiently allowing us to solve much larger problems than previously possible. We also provide a publicly available R (R Development Core Team 2010) implementation in the package **glmnet** (Friedman *et al.* 2011), available from the Comprehensive R Archive Network (CRAN) at `http://CRAN.R-project.org/package=glmnet`.

In Section 2 we introduce our algorithm to fit the Cox model with elastic net penalties. In Section 3 we look at the stability of our algorithm and run time trials on simulated data and one real microarray example.

2. Algorithm

We again consider the survival framework of Section 1. For the time being we assume no ties in failure/censoring time. We wish to find β which maximizes

$$L(\beta) = \prod_{i=1}^{m} \frac{e^{x_{j(i)}^\top \beta}}{\sum_{j \in R_i} e^{x_j^\top \beta}}$$

subject to our constraint: $\alpha \sum |\beta_i| + (1-\alpha) \sum \beta_i^2 \leq c$. Maximizing the partial likelihood is equivalent to maximizing a scaled log partial likelihood,

$$\frac{2}{n}\ell(\beta) = \frac{2}{n}\left[\sum_{i=1}^{m} x_{j(i)}^\top \beta - \log\left(\sum_{j \in R_i} e^{x_j^\top \beta}\right)\right]$$

We scale by a factor of $2/n$ for convenience. Hence, if we consider the Lagrangian formulation, our problem becomes

$$\hat{\beta} = \mathrm{argmax}_\beta \left[\frac{2}{n}\left(\sum_{i=1}^{m} x_{j(i)}^\top \beta - \log\left(\sum_{j \in R_i} e^{x_j^\top \beta}\right)\right) - \lambda P_\alpha(\beta)\right] \tag{1}$$

where,

$$\lambda P_\alpha(\beta) = \lambda \left(\alpha \sum_{i=1}^{p} |\beta_i| + \frac{1}{2}(1-\alpha) \sum_{i=1}^{p} \beta_i^2\right)$$

is known as the elastic net penalty. It is a mixture of the ℓ_1 (lasso) and ℓ_2 (ridge regression) penalties. The lasso penalty (Tibshirani 1996) tends to choose only a few nonzero coefficients. While often desirable, this can cause problems. If two predictors are very correlated, the lasso will pick one and entirely ignore the other.

On the other hand, ridge regression scales all the coefficients towards 0, but sets none to exactly zero. This helps to regularize in problems with $p > n$, but does not give a sparse solution. However, ridge regression better handles correlated predictors. If two predictors are very correlated, ridge regression will tend to give them equal weight.

The elastic net combines the strengths of the two approaches. For fixed λ, as α changes from 0 to 1 our solutions move from more ridge-like to more lasso-like, increasing sparsity but also increasing the magnitude of all non-zero coefficients. With $\alpha = 0.95$ (or even closer to 1), the elastic net behaves very similarly to the lasso, only removing degenerate behavior due to extreme correlations.

2.1. Basic algorithm

Our strategy for maximizing Equation 1 is very similar to the standard Newton Raphson algorithm. However, at each iteration instead of solving a general least squares problem, we solve a penalized reweighted least squares problem.

Let X denote the design matrix, β the coefficient vector, and $\eta = X\beta$. Let $\dot{\ell}(\beta)$, $\ddot{\ell}(\beta)$, $\ell'(\eta)$, $\ell''(\eta)$ denote the gradient and Hessian of the log-partial likelihood with respect to β and η respectively. A two term Taylor series expansion of the log-partial likelihood centered at $\tilde{\beta}$ has the form

$$\begin{aligned}\ell(\beta) &\approx \ell(\tilde{\beta}) + (\beta - \tilde{\beta})^\top \dot{\ell}(\tilde{\beta}) + (\beta - \tilde{\beta})^\top \ddot{\ell}(\tilde{\beta})(\beta - \tilde{\beta})/2 \\ &= \ell(\tilde{\beta}) + (X\beta - \tilde{\eta})^\top \ell'(\tilde{\eta}) + (X\beta - \tilde{\eta})^\top \ell''(\tilde{\eta})(X\beta - \tilde{\eta})/2\end{aligned}$$

where $\tilde{\eta} = X\tilde{\beta}$. Simple algebra gives us

$$\ell(\beta) \approx \frac{1}{2}\left(z(\tilde{\eta}) - X\beta\right)^\top \ell''(\tilde{\eta})\left(z(\tilde{\eta}) - X\beta\right) + C(\tilde{\eta}, \tilde{\beta})$$

where

$$z(\tilde{\eta}) = \tilde{\eta} - \ell''(\tilde{\eta})^{-1}\ell'(\tilde{\eta})$$

and $C(\tilde{\eta}, \tilde{\beta})$ does not depend on β.

One difficulty arises in the computation of $\ell''(\tilde{\eta})$. Because this is a full matrix it would require computation of $O(n^2)$ entries. In order to speed up the algorithm, we instead replace $\ell''(\tilde{\eta})$ by a diagonal matrix with the diagonal entries of $\ell''(\tilde{\eta})$. We denote the ith diagonal entry of $\ell''(\tilde{\eta})$ by $w(\tilde{\eta})_i$. We rely on the argument of Hastie and Tibshirani (1990, Chapter 8) that this substitution works because the optimal β will remain a fixed point of the algorithm, and the off diagonal entries of $\ell''(\tilde{\eta})$ are small in comparison to the diagonal.

Thus, our algorithm is

1. Initialize $\tilde{\beta}$, and set $\tilde{\eta} = X\tilde{\beta}$.
2. Compute $\ell''(\tilde{\eta})$, and $z(\tilde{\eta})$.
3. Find $\hat{\beta}$ minimizing
$$\frac{1}{n}\sum_{i=1}^{n} w(\tilde{\eta})_i (z(\tilde{\eta})_i - x_i^\top\beta)^2 + \lambda P_\alpha(\beta) \tag{2}$$
4. Set $\tilde{\beta} = \hat{\beta}$ and, $\tilde{\eta} = X\hat{\beta}$.
5. Repeat steps $2-4$ until convergence of $\hat{\beta}$.

The minimization in step 3 is done by cyclical coordinate descent, which will be described in Section 2.2.

2.2. Penalized least squares

We have reduced our problem to repeatedly solving the penalized, weighted least squares problem (2)

$$\hat{\beta} = \operatorname{argmin}_\beta \frac{1}{n}\sum_{i=1}^{n} w(\tilde{\eta})_i (z(\tilde{\eta})_i - x_i^\top\beta)^2 + \lambda P_\alpha(\beta) \tag{3}$$

Let $M(\beta)$ denote the objective function in Equation 2. Now consider a coordinate descent step for minimizing $M(\beta)$. Suppose we have estimates for β_l for all $l \neq k$ and would like to minimize our objective in β_k. We compute the derivative

$$\frac{\partial M}{\partial \beta_k} = \frac{1}{n}\sum_{i=1}^{n} w(\tilde{\eta})_i x_{ik} (z(\tilde{\eta})_i - x_i^\top\beta) + \lambda\alpha \cdot \operatorname{sgn}(\beta_k) + \lambda(1-\alpha)\beta_k$$

when $\beta_k \neq 0$, where $\operatorname{sgn}(\beta_k)$ is 1 if $\beta_k > 0$, and -1 if $\beta_k < 0$ and, for the sake of completeness, 0 if $\beta_k = 0$. From here, a simple calculation (Friedman *et al.* 2007) shows that the coordinate solution is given by

$$\hat{\beta_k} = \frac{\mathrm{S}\left(\frac{1}{n}\sum_{i=1}^{n} w(\tilde{\eta})_i x_{i,k}\left[z(\tilde{\eta})_i - \sum_{j\neq k} x_{ij}\beta_j\right], \lambda\alpha\right)}{\frac{1}{n}\sum_{i=1}^{p} w(\tilde{\eta})_i x_{ik}^2 + \lambda(1-\alpha)} \tag{4}$$

with

$$S(x,\lambda) = \operatorname{sgn}(x)(|x|-\lambda)_{+} \tag{5}$$

$$w(\tilde{\eta})_k = \ell''(\tilde{\eta})_{k,k} = \sum_{i\in C_k}\left[\frac{e^{\tilde{\eta}_k}\sum_{j\in R_i} e^{\tilde{\eta}_j} - \left(e^{\tilde{\eta}_k}\right)^2}{\left(\sum_{j\in R_i} e^{\tilde{\eta}_j}\right)^2}\right] \tag{6}$$

$$z(\tilde{\eta})_k = \tilde{\eta}_k - \frac{\ell'(\tilde{\eta})_k}{\ell''(\tilde{\eta})_{k,k}} = \tilde{\eta}_k + \frac{1}{w(\tilde{\eta})_k}\left[\delta_k - \sum_{i\in C_k}\left(\frac{e^{\tilde{\eta}_k}}{\sum_{j\in R_i} e^{\tilde{\eta}_j}}\right)\right] \tag{7}$$

and C_k is the set of i with $t_i < y_k$ (the times for which observation k is still at risk).

Thus, we solve for β_k, combining our usual least squares coordinate wise solution with proportional shrinkage from the ℓ_2 penalty, and soft thresholding from the ℓ_1. Applying Equation 4 to the coordinates of β in a cyclic fashion until convergence minimizes Objective (2). This algorithm was proposed by van der Kooij (2007) for usual linear regression, and applied to logistic and multinomial generalized linear models by Friedman *et al.* (2010).

2.3. Pathwise solution

We will usually be interested in models for more than one value of λ. Toward this end, for fixed α, we compute the solutions for a path of λ values. We begin with λ sufficiently large to set $\beta = 0$, and decrease λ until we arrive near the unregularized solution. By employing warm starts this procedure is efficient and increases the stability of our algorithm.

In Equation 4 notice that if $\frac{1}{n}\sum_{i=1}^{n} w_i(0)x_{ij}z(0)_i < \alpha\lambda$ for all j, then $\beta = 0$ minimizes the objective M. Thus, we set our first λ to be

$$\lambda_{\max} = \max_j \frac{1}{n\alpha}\sum_{i=1}^{n} w_i(0)x_{ij}z(0)_i.$$

We do not solve all the way to the unregularized solution. When $p > n$ the unregularized solution is undefined ($\hat{\beta}$ shoots off to ∞). Even near $\lambda = 0$ the solution is poorly behaved. We have examples where the last few λ values account for more than 99% of the algorithm's runtime. We argue that it is acceptable to ignore solutions with λ near 0, as any reasonable model selection criteria (commonly cross-validation) will choose a much more regularized model. We set $\lambda_{\min} = \epsilon\lambda_{\max}$, and compute solutions over a grid of m values between $\lambda_{\min}$ and $\lambda_{\max}$, where $\lambda_j = \lambda_{max}(\lambda_{\min}/\lambda_{\max})^{j/m}$ for $j = 0,\ldots,m$. In our implementation, the default value for m is 100. The default value for ϵ depends on whether or not $n \geq p$; for $n < p$, we default to $\epsilon = 0.05$, for $n \geq p$, $\epsilon = 0.0001$ Newton step algorithms can be unstable and have no convergence guarantee without step size optimization. This is only a problem if the inital parameter estimate is far from the optimal value. Because we have an exact starting solution at the beginning of our path and employ warm starts at each new λ, our initial estimate and solution at each λ are never far apart. Hence, algorithm is well behaved. For computational speed, we have opted against employing divergence checks in our implementation. Thus far, we have not come across any divergence problems.

2.4. Risk set updates

One obvious bottleneck in our algorithm is the computation of w_k and z_k in Equations 6 and

(7). For each i in C_k we need to calculate $\sum_{j \in R_i} e^{\tilde{\eta}_k}$, the sum of hazards of those still at risk. Because both C_k and R_i have $O(n)$ elements, this is naively an $O(n^2)$ calculation. However, if we note that

$$\sum_{j \in R_{i+1}} e^{\tilde{\eta}_j} = \sum_{j \in R_i} e^{\tilde{\eta}_j} - \sum_{j \in R_i \text{ and } j \notin R_{i+1}} e^{\tilde{\eta}_i}$$

then we need only calculate the full sum, $\sum_{j \in R_i} e^{\tilde{\eta}_k}$, for the first index, i, in C_k. For each subsequent i, we subtract off the contribution from observations with failure/censoring times between time $i-1$ and time i. Thus, the calculation is reduced to $O(n)$. These updates explain, in part, the incredible increase in efficiency our algorithm sees over competing algorithms in section 3 for large n.

2.5. Weights and ties

In the above algorithm, we have assumed that every failure/censoring time was unique and gave equal weight to each observation in the partial likelihood. It is straightforward to include ties and assign different weights to observations. We use the Breslow approximation of the partial likelihood for ties (Breslow 1972) and the corresponding weight extension (integer weights correspond to repeated measurements). The partial likelihood becomes

$$L(\beta) = \prod_{i=1}^{m} \frac{\exp(\sum_{j \in D_i} \omega_j \eta_j)}{\left(\sum_{j \in R_i} \omega_j e^{\eta_j}\right)^{d_i}} \tag{8}$$

where D_i is the set of indices of failure at time t_i, ω_i is the weight associated with observation i and $d_i = \sum_{j \in D_i} \omega_j$ is the sum of weights at time t_i. This changes w_k and z_k

$$\begin{aligned} w(\eta)_k &= \sum_{i \in C_k} d_i \left[\frac{\omega_k e^{\eta_k} \sum_{j \in R_i} \omega_j e^{\eta_j} - (\omega_k e^{\eta_k})^2}{\left(\sum_{j \in R_i} \omega_j e^{\eta_j}\right)^2} \right] \\ z(\eta)_k &= \eta_k + \frac{1}{w_k} \left[\omega_k \delta_k - \sum_{i \in C_k} \left(\frac{\omega_k e^{\eta_k} d_i}{\sum_{j \in R_i} \omega_j e^{\eta_j}} \right) \right]. \end{aligned}$$

The rest of the algorithm remains the same. The addition of weights and ties does not seriously affect the computational cost.

Note, scaling the weights similarly scales the log-partial likelihood. In our implementation, to provide comparability between partial likelihoods with different weight vectors, we standardize the weights to sum to 1. Note, this is why we scale the log likelihood by $1/n$ in the "unweighted problem".

2.6. Deviance cutoff

In addition to the usual stopping criteria, we would like to terminate the algorithm early if, at some point, our model explains almost all of the variability in the observations. To this end, we define the deviance of a model with parameter β to be

$$D(\beta) = 2(\ell_{\text{saturated}} - \ell(\beta)) \tag{9}$$

where $\ell_{\text{saturated}}$ is the maximum log-partial likelihood when the η_i may vary freely (whereas usually η_i is constrained by $\eta_i = x_i^\top \beta$) and $\ell(\beta)$ is the log-partial likelihood under β. We similarly define the null deviance to be

$$D_{\text{null}} = D(0) = 2(\ell_{\text{saturated}} - \ell_{\text{null}}), \tag{10}$$

where $\ell_{\text{null}} = \ell(0)$. Note, for all ℓ in Equations 9 and 10 we use the weighted log-partial likelihood from Section 2.5 (in the unweighted case we use weights $1/n$ for each observation). This definition of deviance parallels deviance defined for likelihood-based models. At a given step we terminate the algorithm early if more than 99% of the null deviance is explained by the model. That is, if

$$D(\beta_{\text{current}}) - D_{\text{null}} \geq 0.99 D_{\text{null}}.$$

A simple calculation shows that

$$\begin{aligned} \ell_{\text{null}} &= -\sum_{i=1}^{m} d_i \log\left(\sum_{j \in R_i} w_j\right) \qquad \text{and,} \\ \ell_{\text{saturated}} &= -\sum_{i=1}^{m} d_i \log(d_i). \end{aligned}$$

In the “unweighted” case (or actually equally weighted, $1/n$ case) this reduces to $\ell_{\text{saturated}} = 0$ and $\ell_{\text{null}} = -\sum_{i=1}^{m} \log|R_i|$. This stopping criterion is another safeguard against wasting time computing solutions for values of λ too small to sufficiently regularize the problem (in the case of $n > p$ this cutoff will effectively never kick in).

2.7. Comparison

A number of other algorithms have recently been developed for the same problem. Gui and Li (2005) and Park and Hastie (2007a) both developed LARS-like algorithms, which combine approximate paths with Newton steps; though they differ in that Gui and Li (2005) use a LARS path inside each Newton approximation while Park and Hastie (2007a) use an approximate LARS path to initialize each step, then solve exactly via Newton Raphson. Goeman (2010a) instead combines gradient descent with Newton's method – using gradient-like steps until close to the solution, then quickly converging with Newton steps.

Our algorithm also uses a Newton Raphson-like method; however unlike other methods, by employing coordinatewise updates we take advantage of the sparsity of each solution. Rather than solving for all coordinates, we iterate over an active set. Because the number of nonzero coordinates is never greater than the number of observations, in the $n << p$ situation we gain significant efficiency within each Newton iteration. Furthermore, efficient computation of risk sets and updating formulas also gains us efficiency in calculating weights and pseudo-responses for each Newton step.

3. Timings

In this section we compare the runtime of our algorithm, `coxnet`, with two competing algorithms, `coxpath` and `penalized`. Because `coxnet`, `coxpath` and `penalized` all take different elastic net penalty paths (`coxpath` and `penalized` fix λ_2) all comparisons were run using only the lasso penalty ($\alpha = 1$). For our algorithm, we also consider the relationship between runtime and both number of observations and covariates. All calculations were carried out on an Intel Xeon 3 GHz processor.

We generated standard Gaussian predictor data, X, n observations on p predictors. We fixed pairwise correlation between any 2 predictors X_i and X_j at ρ, for several values of ρ. We then generated "true" survival times according to

$$Y = \exp\left(\sum_{j=1}^{p} X_j\beta_j + k \cdot Z\right)$$

where $\beta_j = (-1)^j \exp\left(-2(j-1)/20\right)$, $Z \sim N(0,1)$, and k is chosen so that the signal-to-noise ratio is 3.0. Similarly, we generated censoring times by

$$C = \exp\left(k \cdot Z\right).$$

The recorded survival time was set to be the minimum of the "true" survival and censoring times, $T = \min\{Y, C\}$. The observation was said to be censored if $C < Y$, the censoring time preceeded the "true" survival time.

Table 1 shows runtime comparisons between our coordinate descent algorithm, `coxnet`, the combination gradient descent-Newton Raphson method, `penalized` (Goeman 2010a) from the package **penalized** (Goeman 2010b), and the LARS-like algorithm, `coxpath` (Park and Hastie 2007a) from the package **glmpath** (Park and Hastie 2007b). All of these algorithms were implemented in the R language. `coxnet` does all computation in Fortran, while `coxpath` does some computation in R, but frequently calls Fortran routines for numerical optimization. `penalized` does all computation in R. To make the algorithms more comparable, we made `coxnet` solve for the path of λ chosen by `coxpath` with $\lambda_{\text{min}} = 0.05\lambda_{\text{max}}$ fixed for both. Unfortunately `penalized` does not allow for a user indicated λ path. Instead we used the same λ_{min} as the other algorithms and let `penalized` choose a path with number of λ values equal to that of the `coxpath` path. In Table 1 we see that `coxnet` is significantly faster than `coxpath` and `penalized`. In particular, it handles large n more efficiently than both and large p much more efficiently than `penalized` (though somewhat more efficiently than `coxpath` as well). We also note a slight curiousity: `coxnet` runs faster for more highly correlated predictors. This is counter-intuitive and the opposite is seen in Friedman *et al.* (2010) for cyclical coordinate descent applied to standard linear and logistic regression.

3.1. Scaling in n and p

In addition to comparing our algorithm to `coxpath`, we would like to know how it scales in n and p. We simulated data as before, with $\rho = 0.5$ fixed, and a variety of n and p. For each n, p pair we solved for a path of 100 λ values with $\lambda_{\text{min}} = 0.05\lambda_{\text{max}}$. Figure 1 shows runtimes for fixed p as n changes, and for fixed n as p changes. From these plots we can see that the runtime is relatively linear in n and p.

3.2. Cross validation

Once we have calculated a path of solutions it is necessary to select an optimal λ. Model selection is often done by k-fold cross validation - splitting the data in k pieces, using $k - 1$

	Correlation					
	0	0.1	0.2	0.5	0.9	0.95
	$n = 100$, $p = 2000$					
`coxnet`	3.1	3.7	3.0	2.4	1.1	0.8
`coxpath`	22.4	27.6	24.8	23.0	13.9	9.9
`penalized`	104.6	195.2	194.8	171.2	92.6	25.8
	$n = 100$, $p = 5000$					
`coxnet`	10.8	12.3	10.2	11.0	2.9	1.9
`coxpath`	39.7	50.5	43.5	46.0	58.5	49.4
`penalized`	188.2	344.4	446.9	1161.6	228.8	144.8
	$n = 200$, $p = 1000$					
`coxnet`	4.0	4.0	3.3	3.9	0.9	0.6
`coxpath`	147.1	129.6	192.2	229.4	67.4	98.2
`penalized`	61.7	137.0	345.6	415.1	44.0	38.7
	$n = 500$, $p = 50$					
`coxnet`	0.1	0.1	0.1	0.2	0.4	0.2
`coxpath`	83.2	87.1	94.7	103.7	93.4	423.0
`penalized`	6.9	10.8	10.4	11.4	10.6	34.1
	$n = 100$, $p = 10,000$					
`coxnet`	30.8	29.6	29.4	21.6	4.8	4.9
`coxpath`	88.1	84.2	81.1	73.3	150.3	129.8
`penalized`	394.8	613.7	1512.6	792.1	713.5	968.1
	$n = 200$, $p = 40,000$					
`coxnet`	271.3	265.6	252.1	251.8	140.3	101.2
	$n = 100,000$, $p = 500$					
`coxnet`	61.9	66.4	66.5	67.5	50.9	49.8

Table 1: Contains timings (secs) for `coxnet` and `coxpath` with lasso penalty; total time for λ path averaged over 3 trials.

of those to build the model and validating on the kth (often via the predictive likelihood); cycling through this procedure, validating on each of the k pieces in turn, and then averaging or summing the k different deviances. Cross validation in `coxnet` works in a similar fashion, but with a few subtle differences. Consider the extreme case $k = n$, or leave one out cross validation. The partial likelihood on the left out sample is either ill-defined (if the left out sample was right censored) or identically 1 for all β. In this case cross validation tells us nothing about the optimality (or lack thereof) of our model. Because the partial likelihood is not as nicely separable as the Gaussian log likelihood (or any exponential family) naively

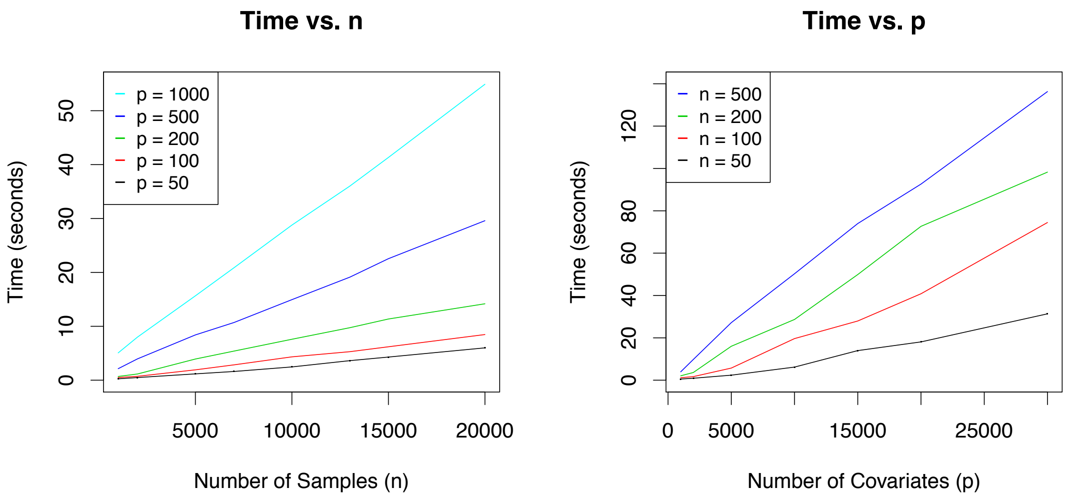

Figure 1: Plots of the timings (seconds) with $\rho = 0.5$ for lasso penalty; total time for 100 λ values, with $\lambda_{min} = 0.05\lambda_{max}$, averaged over 4 trials.

splitting up the observations and attempting traditional cross validation leads to a loss in efficiency.

We use a technique proposed in van Houwelingen *et al.* (2006). We split our data into k parts. Our goodness of fit estimate for a given part i and λ is

$$\hat{\mathrm{CV}}_i(\lambda) = \ell(\beta_{-i}(\lambda)) - \ell_{-i}(\beta_{-i}(\lambda)) \tag{11}$$

where ℓ_{-i} is the log-partial likelihood excluding part i of the data, and $\beta_{-i}(\lambda)$ is the optimal β for the non-left out data, found from maximizing $\ell_{-i} + \lambda||\beta||_1$. Our total goodness of fit estimate, $\hat{\mathrm{CV}}(\lambda)$, is the sum of all $\hat{\mathrm{CV}}_i(\lambda)$. We choose the λ value which maximizes $\hat{\mathrm{CV}}(\lambda)$. By using (11) – subtracting the log-partial likelihood evaluated on the non-left out data from that evaluated on the full data – we can make efficient use of the death times of the left out data in relation to the death times of all the data.

3.3. Real data

We also compare timings on a real data set, Alizadeh *et al.* (2000): gene-expression data in lymphoma patients. There were 240 patients with measurements on 7399 genes. We used the lasso penalty ($\alpha = 1$), and the path of λ values was chosen as in the simulations. We ran 10-fold cross validation over these paths to find the optimal value of λ. 10-fold cross validation run times were 574.5 seconds for `coxnet`, 18998.7 seconds for `penalized`, and 4796.3 seconds for `coxpath`. Without cross validation, times were 54.9 seconds for `coxnet`, 2524.9 seconds for `penalized` and 679.5 seconds for `coxpath`. As in the simulation results, our method shows a substantial run time improvement.

3.4. Code example

Our implementation is straightforward to run. We demonstrate `coxnet` on the data from Section 3.3. We first load the data and set up the response.

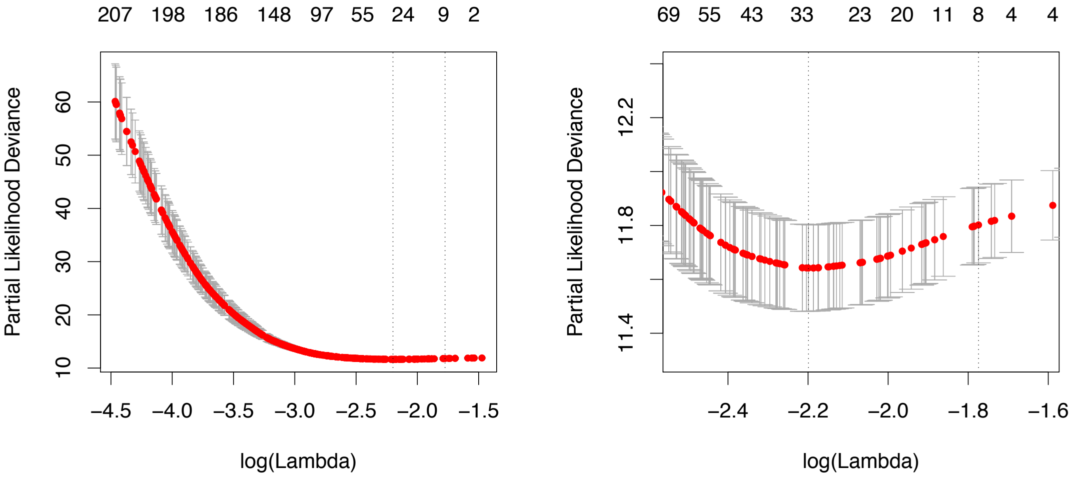

Figure 2: Plots of the cross-validated error rates. Each dot represents a λ value along our path, with error bars to give a confidence interval for the cross-validated error rate. The left vertical bar indicates the minimum error while the right shows the largest value of λ such that the error is within one standard deviation of the minimum. The top of the plot gives the size of each model. Left plot is original output, while right plot is zoomed in.

```
R> attach("LymphomaData.rda")
R> x <- t(patient.data$x)
R> y <- patient.data$time
R> status <- patient.data$status
```

We then call our functions to fit with the lasso penalty ($\alpha = 1$), and cross validate.

```
R> fit <- glmnet(x, Surv(y,status), family =  "cox", alpha = 1)
R> cv.fit <- cv.glmnet(x, Surv(y,status), family = "cox", alpha = 1)
```

The `Surv` function packages the survival data into the form expected by `coxnet`. Once fit, we can view the optimal λ value and a cross-validated error plot (Figure 2) to help evaluate our model.

```
R> cv.fit$lambda.min
```

```
[1] 0.1109075
```

```
R> plot(cv.fit)
```

The optimal λ value in this case is 0.11. We can zoom in on this value (Figure 2, right plot) to get a better look at the error curve.

```
R> plot(cv.fit, xlim = c(log(0.08), log(0.2)), ylim = c(11.3, 12.4))
```

By default our function uses 10-fold cross validation to estimate prediction error in a nearly unbiased way.

4. Discussion

We have shown cyclical coordinate descent to be a very efficient algorithm for maximizing the partial likelihood with the elastic net penalty. Each coordinate step has a simple, closed form solution. By employing warm starts we have increased the stability of the algorithm and found solutions for a path of penalty parameters. In simulations and an actual dataset we have shown our algorithm to be significantly faster than the competition. A new version of **glmnet** (Friedman *et al.* 2011) incorporating `coxnet` is available on CRAN.

References

Alizadeh AA, Eisen MB, Davis RE, Ma C, Lossos IS, Rosenwald A, Boldrick JC, Sabet H, Tran T, Yu X, *et al.* (2000). "Distinct Types of Diffuse Large B-Cell Lymphoma Identified by Gene Expression Profiling." *Nature*, **403**(6769), 503–511.

Breslow NE (1972). "Contribution to the Discussion of the Paper by D.R. Cox." *Journal of the Royal Statistical Society B*, **34**, 216–217.

Cox DR (1972). "Regression Models and Life Tables." *Journal of the Royal Statistical Society B*, **34**, 187–220.

Efron B, Hastie T, Johnstone I, Tibshirani R (2004). "Least Angle Regression." *The Annals of Statistics*, **32**(2), 407–499.

Friedman J, Hastie T, Hoefling H, Tibshirani R (2007). "Pathwise Coordinate Optimization." *The Annals of Applied Statistics*, **2**(1), 302–332.

Friedman J, Hastie T, Tibshirani R (2010). "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software*, **33**(1), 1–22. URL `http://www.jstatsoft.org/v33/i01/`.

Friedman J, Hastie T, Tibshirani R (2011). "**glmnet**: Lasso and Elastic-Net Regularized Generalized Linear Models." R package version 1.5.2, URL `http://CRAN.R-project.org/package=glmnet`.

Goeman JJ (2010a). "L1 Penalized Estimation in the Cox Proportional Hazards Model." *Biometrical Journal*, **52**(1), 70–84.

Goeman JJ (2010b). "**penalized**: L1 (Lasso) and L2 (Ridge) Penalized Estimation in GLMs and in the Cox Model." R package version 0.9-32, URL `http://CRAN.R-project.org/package=penalized`.

Gui J, Li H (2005). "Penalized Cox Regression Analysis in the High-Dimensional and Low-Sample Size Settings, with Applications to Microarray Gene Expression Data." *Bioinformatics*, **25**(13), 3001–2008.

Hastie T, Tibshirani R (1990). *Generalized Additive Models*. Chapman and Hall, London.

Park MY, Hastie T (2007a). "L1-Regularization Path Algorithm for Generalized Linear Models." *Journal of the Royal Statistical Society B*, **69**, 659–677.

Park MY, Hastie T (2007b). "**glmpath**: L1 Regularization Path for Generalized Linear Models and Cox Proportional Hazards Model." R package version 0.94, URL http://CRAN.R-project.org/package=glmpath.

R Development Core Team (2010). "R: A Language and Environment for Statistical Computing." ISBN 3-900051-07-0, URL http://www.R-project.org/.

Tibshirani R (1996). "Regression Shrinkage and Selection via the Lasso." *Journal of the Royal Statistical Society B*, **58**, 267–288.

Tibshirani R (1997). "The Lasso Method for Variable Selection in the Cox Model." *Statistics in Medicine*, **16**, 385–395.

van der Kooij AJ (2007). *Prediction Accuracy and Stability of Regrsssion with Optimal Scaling Transformations*. Ph.D. thesis, Department of Data Theory, Leiden University. URL http://hdl.handle.net/1887/12096.

van Houwelingen HC, Bruinsma T, Hart AAM, van't Veer LJ, Wessels LFA (2006). "Cross-Validated Cox Regression on Microarray Gene Expression Data." *Statistics in Medicine*, **25**, 3201–3216.

Wu T, Lange K (2008). "Coordinate Descent Procedures for Lasso Penalized Regression." *The Annals of Applied Statistics*, **2**(1), 224–244.

Zou H, Hastie T (2005). "Regularization and Variable Selection via the Elastic Net." *Journal of the Royal Statistical Society B*, **67**(2), 301–320.

Affiliation:

Noah Simon, Jerome Friedman, Trevor Hastie, Rob Tibshirani
Department of Statistics
Stanford University
390 Serra Mall
Stanford CA, 94305, United States of America
E-mail: nsimon@stanford.edu, jhf@stanford.edu,
hastie@stanford.edu, tibs@stanford.edu

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 39, Issue 5 *Submitted:* 2010-06-15
March 2011 *Accepted:* 2010-10-02