



## Combining $l^1$ and Higher Order $l^p$ Penalization in Regression Models

Austin David Brown  
University of Minnesota

---

### Abstract

We create the Penalized Regression on Steroids (**pros**) package to combine  $l^1$  penalized regression with higher order  $l^p$  penalizations built upon the elastic-net idea. The package is able to fit regression models with penalizations ranging from  $l^1$  to  $l^{10}$ .

*Keywords:* Penalized Regression, C++, Python, R.

---

### 1. Introduction

In statistics and probability theory it is common to impose moment assumptions on a random variable  $X : \Omega \rightarrow \mathbb{R}^n$  such as  $E(|X|^k) < \infty$  for  $k \in \mathbb{N}$ . These constraints correspond to the  $L^p$  spaces which allow control over the width and the height of such random variables. Constraints of this type may also be motivated geometrically. Consider for example an elastic-net penalty  $Q(x) = \frac{1}{2}|x| + \frac{1}{2}|y| + \frac{1}{2}|x|^2 + \frac{1}{2}|y|^2 \leq 1$  shown on the left and a new penalty  $P(x) = \frac{1}{2}|x| + \frac{1}{2}|y| + \frac{1}{2}|x|^4 + \frac{1}{2}|y|^4 \leq 1$  shown on the right.

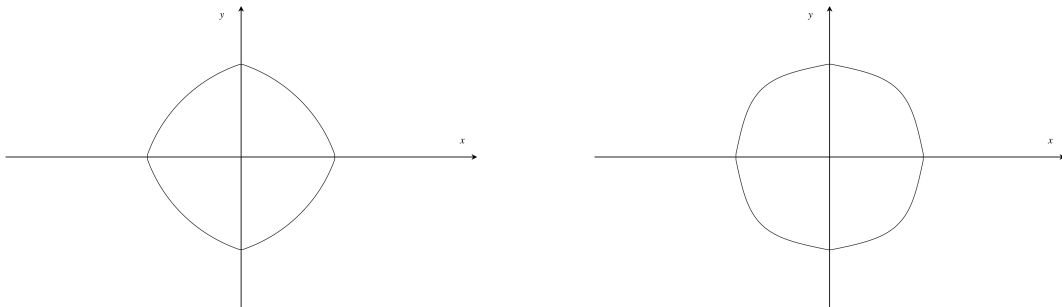


Figure 1: A particular elastic-net penalty shown on the left and a new penalty shown on the right shown to "bow" out the penalization while retaining convexity.

It seems reasonable that a scientist may want more control over the height and width of the penalization while still retaining sparse solutions in regression models. In this project, we build the Penalized Regression on Steroids (**pros**) package to expand upon the idea of the elastic-net by [Zou and Hastie \(2005\)](#), which we believe approximately encapsulates this idea.

## 2. Penalized Regression Implementation

In this section, we discuss how the **pros** package was implemented. In particular, the construction of the problem and the algorithms used. First, we formulate the regression model with a new penalization. Let

$$L_\lambda(\beta) = \frac{1}{2} \|y - \beta_0 \mathbf{1} - X\beta\|_2^2 + \lambda P(\beta)$$

be the objective to minimize with penalty

$$\lambda P(\beta) = \lambda \alpha_0 \|\beta\|_1 + \lambda \sum_{k=1}^5 \alpha_k \|\beta\|_{2k}^{2k}$$

where  $y \in \mathbb{R}^n$ ,  $X \in M_{n \times p}(\mathbb{R})$ , and  $(\beta_0, \beta) \in \mathbb{R}^p$ ,  $\lambda \in \mathbb{R}_+$  and  $\alpha$ 's are convex combinations. This penalty is convex and completely separable which allows for coordinate-wise optimization. The coordinate descent algorithm is very successful for the elastic-net penalty due to the **glmnet** package by [Simon, Friedman, Hastie, and Tibshirani \(2011\)](#). Further, with the elastic-net penalty, the coordinate wise minimization yields an analytic, closed form solution at each iteration avoiding any need for a line search or step size. The penalty that we propose lacks an analytic solution and thus a new algorithm is needed.

### 2.1. Optimization Algorithms and Implementation

The first algorithm utilized was the subgradient coordinate algorithm shown below.

---

#### Algorithm 1: Subgradient Coordinate Algorithm

---

Choose  $\beta^0 \in \mathbb{R}^p$ , tolerance  $\delta > 0$ ,  $R > 0$ , and maximum iterations  $N$ .

Set  $k \leftarrow 0$

Set the step size  $h \leftarrow \frac{R}{\sqrt{1+N}}$

**repeat**

    Permute  $I = \{1, \dots, p\}$

**for**  $i \in I$  **do**

$\beta_i^{k+1} \leftarrow \beta_i^k - hg^i$  where  $g_i^k \in (\partial L_\lambda(\beta^k))_i$

$k \leftarrow k + 1$

**end**

**until** *Until the objective difference is less than  $\delta$ ;*

---

The drawbacks of this algorithm include lack of the descent property, no good stopping criterion, and many possible choices for the subgradient. The step size is optimal and chosen due to [Nesterov \(1998\)](#) with a worst case convergence rate of  $O(\frac{1}{\sqrt{k}})$ . Ultimately, no line search can be implemented and the step size must be tuned by the user, which may be difficult for inexperienced users. Another drawback is that this particular stopping criterion is expensive at  $O(n^2)$  flops at each iteration.

Due to the separability of the penalization, a better algorithm is proximal gradient coordinate descent shown below.

---

**Algorithm 2:** Proximal Gradient Coordinate Descent

---

Choose  $\beta^0 \in \mathbb{R}^p$  and tolerance  $\delta > 0$ ;

Set  $k \leftarrow 0$

**repeat**

    Randomly permute  $I = \{1, \dots, p\}$

**for**  $i \in I$  **do**

        Set the step size  $h_i > 0$  or use line search.

$\beta_i^{k+1} \leftarrow (\mathbf{prox}_{h_i L})_i(\beta_i^k - h_i \langle X_i, y - X\beta \rangle)$

$k \leftarrow k + 1$

**end**

**until** *Until the Moreau-Yoshida mapping  $M_{h_k, f} < \delta$ ;*

---

A major benefit of this algorithm is that it recovers the worst case convergence of  $O(\frac{1}{k})$  even though the objective is not differentiable. The convergence analysis of coordinate descent algorithms is explored by [Nesterov \(2010\)](#). The stopping criterion chosen here is cheap to compute at  $O(p)$  flops. Finally, since this is a descent method, line search may be implemented, which potentially makes the algorithm easier to use by inexperienced users.

## 2.2. Cross-Validation Algorithms and Implementation

In order to tune the Lagrangian dual variable  $\lambda$ , K-fold cross validation is implemented. To improve the speed of cross-validation, a warm start algorithm is used defined below.

---

**Algorithm 3:** Warm Start Cross-Validation

---

Choose a sequence of Lagrangian dual variables  $\lambda_1, \dots, \lambda_N$ , and initial value  $\beta^0$ .

Order  $\lambda_{(1)}, \dots, \lambda_{(N)}$  descending.

$\beta^{Warm} \leftarrow \beta^0$

**for**  $k \in 1, \dots, N$  **do**

$\beta^k \leftarrow$  by Cross-Validation with  $\lambda_{(k)}$  warm started with  $\beta^{Warm}$ .

$\beta^{Warm} \leftarrow \beta^k$

**end**

---

A difficult problem in general is choosing the sequence of  $\lambda$ 's. In **glmnet**, the default is to choose a sequence of length 100 starting from the first  $\lambda_{max}$  where  $\beta$  is completely zero to the lower value  $.001\lambda_{max}$  on a log-scale. The lower  $\lambda$  value in the sequence is a real difficulty and this implementation does not fully address this. Due to time, we implement a similar but cruder default choice. We choose the lower  $\lambda_{min} = cp$  where  $p$  number of columns in the predictor matrix  $X$  and  $c$  is chosen to be  $\frac{3}{2}$ . We then create a sequence of length 100 from this lower value. A comparison of the default choices is explored in a subsequent section.

## 2.3. Step Size Rules and Line Search Implementation

The lack of an analytic solution introduces a necessary step size rule in order to fit a regression model with the new penalization. A good line search method due to [Beck and Teboulle \(2009\)](#) requires an evaluation of the objective function over the entire dataset at each iteration. This has a computational complexity of  $O(n^3)$  flops at each iteration. This was attempted, but

never fully implemented for this reason. Instead, a user defined step size rule is required. Thus, poor step size choices do not guarantee convergence. This is a suboptimal choice, but allows the potential for the package to be applied to large datasets.

We introduce a possible alternative algorithm to address the issue.

---

**Algorithm 4:** Step Size Tuning Avoidance Algorithm

---

Choose initial value  $\beta^0$ , initial step size  $h$ , and maximum iterations  $N$ .

**repeat**

$h \leftarrow \frac{1}{2}h$

$\beta \leftarrow$  by Proximal Gradient Coordinate Descent with step size  $h$

**until** *Until Proximal Gradient Coordinate Descent convergences;*

---

If the maximum iterations are not too large, this may allow users to avoid having to tune a step size while retaining good performance. We can also modify this algorithm to be run in parallel. Currently, this is not implemented due to time constraints for the project.

### 3. The Penalized Regression on Steroids Package

#### 3.1. Package Architecture

The package **pros** is built using C++ and the Eigen library by [Guennebaud, Jacob \*et al.\* \(2010\)](#) is utilized for fast matrix computations. This is analogous to using Fortran with LAPACK by [Anderson, Bai, Dongarra, Greenbaum, McKenney, Du Croz, Hammarling, Demmel, Bischof, and Sorensen \(1990\)](#). The **pros** library is not dependent on R. To interact with R, 2 interfacing layers are needed: an R to C interface and an R program defining user callable functions. The benefit is that many other popular programming languages can be interfaced easily. The following diagram illustrates this idea showing that the **pros** package is more general than just an R package.

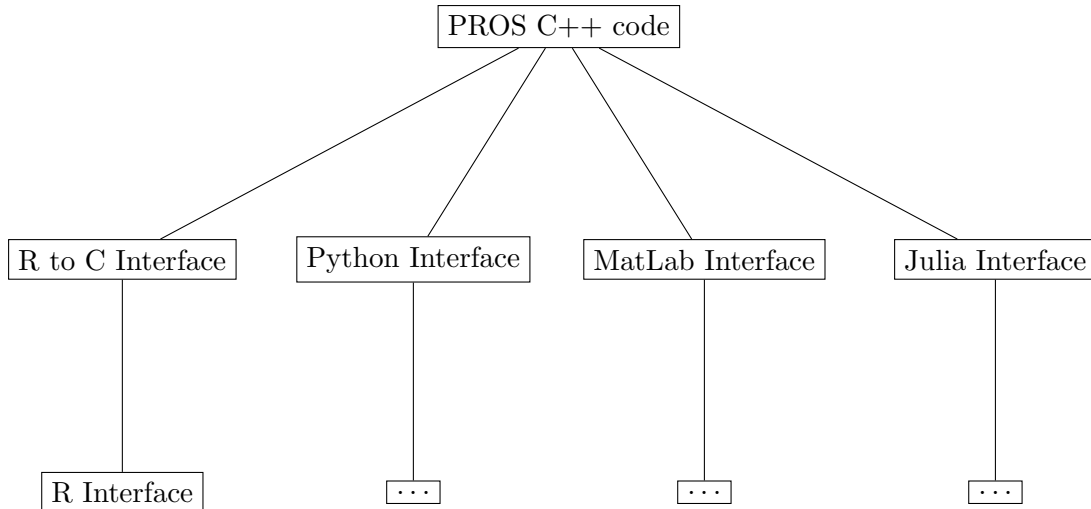


Figure 2: Illustrates interfacing the **pros** package with other languages.

The R interface in its simplest form is comprised of 2 functions. A single fitting function with

prediction

```
> fit <- pros(X, y, alpha)
> predict(fit, X)
```

and a cross-validation function with prediction

```
> cv <- cv.pros(X, y, alpha)
> predict(cv, X)
```

We illustrate the usage of these functions in the following sections.

### 3.2. The Regression Fitting Function

The **pros** function is used to fit a single regression model with a specified  $\lambda$  for penalization. The signature of this function at the time of this paper is as follows:

- **X** is an  $n \times m$ -dimensional matrix of the data.
- **y** is an  $n$ -dimensional vector of response values.
- **alpha** is a 6-dimensional vector of the convex combination corresponding to the penalization:
  - $\alpha_1$  is the  $l^1$  penalty.
  - $\alpha_2$  is the  $l^2$  penalty.
  - $\alpha_3$  is the  $l^4$  penalty.
  - $\alpha_4$  is the  $l^6$  penalty.
  - $\alpha_5$  is the  $l^8$  penalty.
  - $\alpha_6$  is the  $l^{10}$  penalty.
- **lambda** is the Lagrangian dual penalization parameter.
- **step\_size** is a tuning parameter defining the step size. Larger values are more aggressive and smaller values are less aggressive.
- **algorithm** is the optimization algorithm
  - **proximal\_gradient\_cd** uses proximal gradient coordinate descent.
  - **subgradient\_cd** uses subgradient coordinate descent.
- **max\_iter** is the maximum iterations the algorithm will run regardless of convergence.
- **tolerance** is the accuracy of the stopping criterion.
- **random seed** is the random seed used in the algorithms.

### 3.3. The Cross-Validation Function

The **cv.pros** function is used for K-fold cross-validation. The warm-start algorithm and default choices for the  $\lambda$  sequence are implemented as described in the previous section. The arguments of this function are similar to the **pros** function:

- $\mathbf{X}$  is an  $n \times m$ -dimensional matrix of the data.
- $\mathbf{y}$  is an  $n$ -dimensional vector of response values.
- $\mathbf{alpha}$  is a 6-dimensional vector of the convex combination corresponding to the penalization:
  - $\alpha_1$  is the  $l^1$  penalty.
  - $\alpha_2$  is the  $l^2$  penalty.
  - $\alpha_3$  is the  $l^4$  penalty.
  - $\alpha_4$  is the  $l^6$  penalty.
  - $\alpha_5$  is the  $l^8$  penalty.
  - $\alpha_6$  is the  $l^{10}$  penalty.
- `K_fold` is the number of folds in cross-validation.
- `lambdas` is a vector of dual penalization values to be evaluated.
- `step_size` is a tuning parameter defining the step size. Larger values are more aggressive and smaller values are less aggressive.
- `algorithm` is the optimization algorithm
  - `proximal_gradient_cd` uses proximal gradient coordinate descent.
  - `subgradient_cd` uses subgradient coordinate descent.
- `max_iter` is the maximum iterations the algorithm will run regardless of convergence.
- `tolerance` is the accuracy of the stopping criterion.
- `random seed` is the random seed used in the algorithms.

### 3.4. Possible CRAN Submission

At the time of this paper, the **pros** package could potentially be submitted to the Comprehensive R Archive Network (CRAN). According to the R check, there are no warnings. However, one issue with CRAN is that in order for the user to get high performance, they will need to update their own Makevars file. The instructions for this are outlined in the **pros** repository.

## 4. Numerical Examples

### 4.1. The Analysis Procedure

The analysis for the subsequent experiments is performed in the following way:

- The data is split into a training and test set.
- The predictor data is standardized.

- The **glmnet** library and the **pros** library are used for fitting. The **glmnet** library provides varying results for a single seed so a range of values are reported.
- 10-fold cross-validation is used to tune the Lagrangian dual variable  $\lambda$ .
- The metric for comparison is the *test mean squared error* defined as

$$\frac{1}{n} \sum_{k=1}^n \left( y_i - \hat{\beta}_0 - X_i^T \hat{\beta} \right)^2$$

where  $\hat{\beta}_0, \hat{\beta}$  are fit over the training set and  $y$  and  $X$  are the response and predictor matrix belonging to the test set.

- The source code for this analysis is openly available at the **pros** repository available here [Brown \(2018\)](#).

## 4.2. The Boston Housing Dataset

The Boston Housing data is a popular dataset from [Harrison and Rubinfeld \(1978\)](#). There are 13 predictors and the response is the median value of owner-occupied homes in \$1000's. The data is randomly split into a training set with 404 observations and a test set with 102 observations. We compare the default cross-validation implementations between the two libraries:

Penalty	Tuning	Test Mean Squared Error
Lasso (glmnet)	$\alpha = 1$	26.96866-28.45781
elastic-net (glmnet)	$\alpha = 1/2$	26.55704-26.86488
Lasso (pros)	$\alpha = 1$	26.47315
elastic-net (pros)	$\alpha = 1/2$	26.07919
$l^1$ and $l^4$ (pros)	$\alpha = 1/2$	25.5008
$l^1$ and $l^{10}$ (pros)	$\alpha = 1/2$	30.54759

Next, we compare the tuned results using **pros**:

Penalty	Tuning	Test Mean Squared Error
elastic-net (pros)	$\alpha = 0.1$	25.52578
$l^1$ and $l^4$ (pros)	$\alpha = 0.6$	25.44149
$l^1$ and $l^{10}$ (pros)	$\alpha = 0.9$	25.68279

The three different penalizations each select all of the variables and provide no sparsity in the tuned results.

## 4.3. The Prostate Dataset Analysis

The prostate dataset is from [A. Stamey, N. Kabalin, E. Mcneal, M. Johnstone, Freiha, A. Redwine, and Yang \(1989\)](#). There are 8 predictors and the response is the log of the prostate specific antigen. We compare the default cross-validation implementations between the two libraries:

Penalty	Tuning	Test Mean Squared Error
Lasso (glmnet)	$\alpha = 1$	0.4443432-0.4646501
elastic-net (glmnet)	$\alpha = 1/2$	0.4539684-0.4633808
Lasso (pros)	$\alpha = 1$	0.4457657
elastic-net (pros)	$\alpha = 1/2$	0.4637193
$l^1$ and $l^4$ (pros)	$\alpha = 1/2$	0.4621351

Next, we compare the tuned results using **pros**:

Penalty	Tuning	Test Mean Squared Error
elastic-net (pros)	$\alpha = .99$	0.4441737
$l^1$ and $l^4$ (pros)	$\alpha = .99$	0.4441405
$l^1$ and $l^{10}$ (pros)	$\alpha = .99$	0.4441301

In the tuned case, all three penalizations select the same variables  $(\beta_1, \beta_2, \beta_4, \beta_5, \beta_8)$ , which is not the optimal choice according to [Zou and Hastie \(2005\)](#). Interestingly, all of the best values share the same  $\lambda$  value.

#### 4.4. Conclusion

It appears the new penalty can sometimes produce a minor improvement over the elastic-net penalty. However, there was much to learn in the implementation. Creating an R package is much more difficult than anticipated and CRAN is stringent on the requirements. Comparing against other packages can also be difficult. One source of confusion was noticing that **glmnet** uses a transformation for their  $\lambda$  values and these do not correspond with the typical Lasso model formulation. Thus, direct comparisons are complicated. Further adding to the complication is that **glmnet** provides varying results for fixed random seeds. More time should have been spent carefully reviewing the **glmnet** paper by [Simon et al. \(2011\)](#). Due to this, the values in the corresponding slides for this project are incorrect.

## References

- A Stamey T, N Kabalin J, E Mcneal J, M Johnstone I, Freiha F, A Redwine E, Yang N (1989). "Prostate Specific Antigen in the Diagnosis and Treatment of Adenocarcinoma of the Prostate. II. Radical Prostatectomy Treated Patients." *The Journal of urology*, **141**, 1076–83. doi:[10.1016/S0022-5347\(17\)41175-X](#).
- Anderson E, Bai Z, Dongarra J, Greenbaum A, McKenney A, Du Croz J, Hammarling S, Demmel J, Bischof C, Sorensen D (1990). "LAPACK: A Portable Linear Algebra Library for High-performance Computers." In *Proceedings of the 1990 ACM/IEEE Conference on Supercomputing*, Supercomputing '90, pp. 2–11. IEEE Computer Society Press, Los Alamitos, CA, USA. ISBN 0-89791-412-0. URL <http://dl.acm.org/citation.cfm?id=110382.110385>.
- Beck A, Teboulle M (2009). *Gradient-based algorithms with applications to signal-recovery problems*, p. 42–88. Cambridge University Press. doi:[10.1017/CB09780511804458.003](#).
- Brown A (2018). "Penalized Regression on Steroids." <http://github.com/austindavidbrown/pros>.



- Guennebaud G, Jacob B, *et al.* (2010). “Eigen v3.” <http://eigen.tuxfamily.org>.
- Harrison D, Rubinfeld D (1978). “Hedonic prices and the demand for clean air.” *J. Environ. Economics and Management*.
- Nesterov Y (1998). “Introductory Lectures On Convex Programming.”
- Nesterov Y (2010). “Efficiency of coordinate descent methods on huge-scale optimization problems.”
- Simon N, Friedman J, Hastie T, Tibshirani R (2011). “Regularization Paths for Cox’s Proportional Hazards Model via Coordinate Descent.” *Journal of Statistical Software*, **39**(5), 1–13. URL <http://www.jstatsoft.org/v39/i05/>.
- Zou H, Hastie T (2005). “Regularization and variable selection via the Elastic Net.” *Journal of the Royal Statistical Society, Series B*, **67**, 301–320.

**Affiliation:**

Austin David Brown  
University of Minnesota  
Department of Statistics (Graduate Student)  
E-mail: [Brow5079@umn.edu](mailto:Brow5079@umn.edu)