Paper 3297-2015

Lasso Regularization for Generalized Linear Models in Base SAS® Using Cyclical Coordinate Descent

Robert Feyerharm, Beacon Health Options

ABSTRACT

The cyclical coordinate descent method is a simple algorithm that has been used for fitting generalized linear models with lasso penalties by Friedman et al. (2007). The coordinate descent algorithm can be implemented in Base SAS® using array processing to perform efficient variable selection and shrinkage for GLMs with the L1 penalty (the lasso).

INTRODUCTION

Ever since its introduction in 1996 by Robert Tibshirani, the lasso (least absolute shrinkage and selection operator) has proven to be a powerful tool in predictive modeling. The lasso is a linear regression model subject to a constraint on the magnitude of the coefficients (an L1 penalty) which enables the lasso to perform both coefficient shrinkage and variable selection. The lasso is particularly effective with data containing correlated variables, and can even be applied to over determined data where the number of variables exceeds the number of observations (*p>N*).

The lasso has been extended from linear models to generalized linear models such as logistic regression and Poisson regression. In addition a number of related techniques have been developed, including the elastic net which benefits from the properties of both L1 and L2 (ridge regression) penalties.

A number of statistical software packages have implemented the lasso and related methods such as the elastic net, including SAS, R and Stata. While SAS offers only limited access to this technique with the `glmselect` procedure, all of the coding building blocks necessary to write your own lasso routine are supported in Base SAS®. Indeed, SAS is an ideal environment for applying the lasso to big data problems given it doesn't suffer the memory restrictions found with competing languages such as R.

This paper describes a SAS program that utilizes the simple but effective cyclical coordinate descent method developed by Jerome Friedman et al. (2010) to apply the lasso to generalized linear models. The code for two GLMs is included here: the simple linear model with a Gaussian link and the two-class logistic regression model. The basic coordinate descent algorithm, and the efficient array processing code described herein, may be adapted to GLMs with other link functions, such as the Poisson regression.

THE LASSO AND ELASTIC NET

The lasso finds coefficient estimates for linear regression models by minimizing the residual sum of squares of the residuals while satisfying the constraint that the **sum of the absolute values** of the regression coefficients does not exceed a predetermined constant (i.e., the L1 penalty).

$$\hat{\beta}^{lasso} = \min_{\beta} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 \quad (1)$$

$$\text{where } \sum_{j=1}^{p} |\beta_j| \leq t \,. (2)$$

In so doing, the lasso conveniently performs coefficient shrinkage comparable to the ridge regression as well as variable selection by reducing coefficients to zero. By sacrificing a small amount of bias in the predicted response variable in order to decrease variance, the lasso achieves improved predictive accuracy compared with ordinary least squares (OLS) models, particularly with data containing highly correlated predictor variables or in over determined data where *p>N*.

A second form of penalized regression, the elastic net (Zou and Hastie, 2005), will also be considered in this paper as it does not introduce significantly more complexity into the coordinate descent code. The elastic net is a compromise between the lasso and the ridge regression's L2 penalty. The elastic net finds coefficient estimates for linear regression models by minimizing the residual sum of squares of the residuals while satisfying the constraint that the **sum of the squared values** of the regression coefficients must not exceed a predetermined constant (i.e., the L2 penalty).

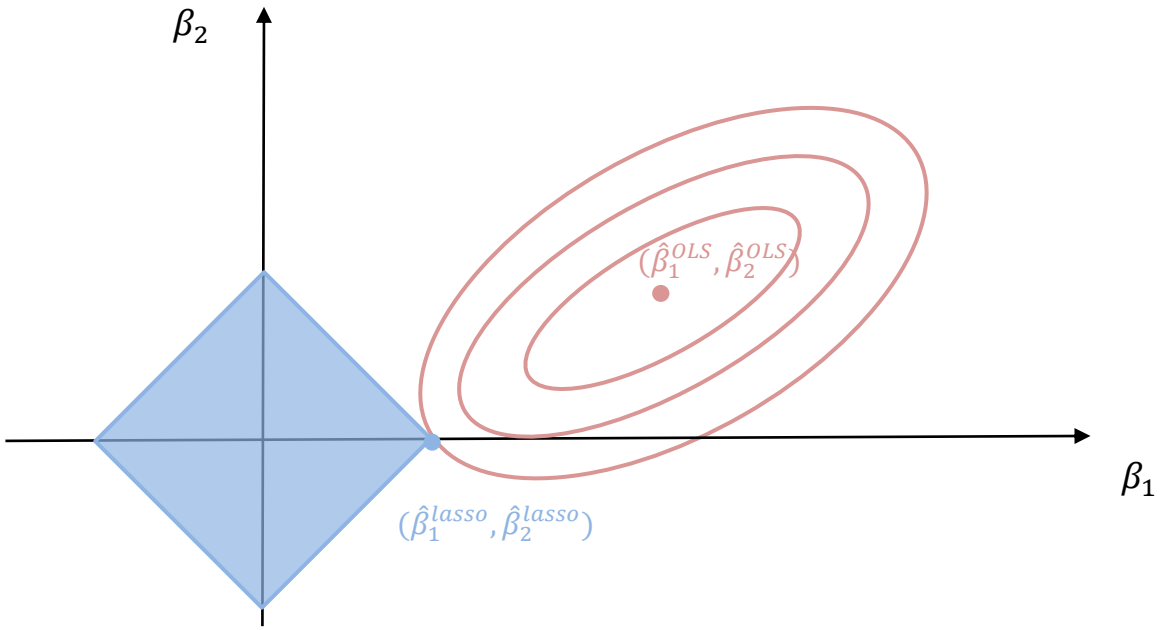
$$\hat{\beta}^{elasticnet} = \min_{\beta} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 (3)$$

$$\text{where } \sum_{j=1}^{p} (\alpha\beta_j^2 + (1-\alpha)|\beta_j|) \leq t \,. (4)$$

The elastic net penalty has advantages over the lasso where there are many correlated features, as the lasso will arbitrarily pick one variable and drop the rest from a family of correlated variables. Setting $\alpha = 0$ produces a ridge regression penalty, which will retain all variables in the model. Setting $\alpha = 1$ produces the lasso penalty.

Figure 1, borrowed from Tibshirani (1996), is a schematic illustrating how the lasso estimates coefficients in a two parameter OLS model. The standard OLS solution is the point ($\hat{\beta}_1^{OLS}, \hat{\beta}_2^{OLS}$) located at the "bottom" of the convex bowl of the sum of squared residuals for varying values of $\hat{\beta}_1$ and $\hat{\beta}_2$. The lasso solution ($\hat{\beta}_1^{lasso}, \hat{\beta}_2^{lasso}$) is located on the corner of the region satisfying the constraint $|\hat{\beta}_1| + |\hat{\beta}_2| \leq t$. Here the lasso algorithm has effectively performed variable selection by reducing the value of $\hat{\beta}_2$ to zero.

Figure 1. Parameter estimation with the lasso.



CYCLICAL COORDINATE DESCENT

Initially, quadratic programming techniques were implemented to solve lasso regression models (Tibshirani 1996). The cyclical coordinate descent method was first proposed by Fu (1998). Compared with quadratic programming, coordinate descent has the advantage of being both fast and simple to implement. More recently, Friedman, et al. (2010) elaborated an algorithm that applies coordinate descent to least squares and logistic regression models with lasso and elastic net penalties, and outlined an approach for finding lasso solutions to other generalized linear models using coordinate descent. Likewise, this paper will describe two coordinate descent algorithms for:

1) Least squares solutions for linear regression models with lasso or elastic net penalties, and
2) Iteratively reweighted least squares solutions for logistic regression models with lasso or elastic net penalties.

Least Squares Coordinate Descent

The coordinate descent algorithm for linear regression models finds the least squares solution for the penalized linear model $E(Y|X = x) = \beta_0 + X^T\beta$:

$$\min_{\beta} f(\boldsymbol{\beta}) = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda\sum_{j=1}^{p}\left[\frac{1}{2}(1-\alpha)\beta_j^2 + \alpha\left|\beta_j\right|\right], (5)$$

where we have *N* observations (x_i, y_i) and the y_i and x_{ij} are standardized so that $\sum_{i=1}^{N} y_i = 0$, $\sum_{i=1}^{N} x_{ij} = 0$, $\frac{1}{N}\sum_{i=1}^{N} y_i^2 = 1$ and $\frac{1}{N}\sum_{i=1}^{N} x_{ij}^2 = 1$ for $j = 1, \ldots, p$.

It is recommended you run the algorithm on a training sample and then fit the final model to the records in the remaining validation data in order to compare model performances for varying values of λ and α.

Cross-validation (either 5-fold or 10-fold) is also commonly used for smaller datasets that prohibit a single training/validation split.

At its simplest, the coordinate descent procedure for solving (5) iterates through an outer loop, a middle loop, and an inner loop in the following steps:

1. The outer loop of the coordinate descent algorithm will step through a sequence of λ values in the lasso (or λ and α values in the elastic net) and repeat steps 2.-5. Usually λ values ranging over a semi-logarithmic scale (e.g., .001, .005, .01, .05, .1,…) are chosen. The sequence of α penalty values typically fall between 0 and 1. Note that the final coefficient estimates for each value of λ_t may be used as the initial value (as a "warm start") for the subsequent λ_{t+1} in the outer loop sequence.
2. Within each iteration of the outer loop, the middle loop repeats the coordinate descent update process $i = 1,2, \ldots, k$ times, a sufficient number of updates for the algorithm to converge on a solution.
3. Within each iteration of the middle loop, the inner loop updates each of the regression coefficients, one at a time. The formula at the heart of the least squares coordinate descent update algorithm is found by computing the gradient (first derivative) of $f(\boldsymbol{\beta})$ at $\beta_j = \tilde{\beta}_j$, setting $\frac{\partial f(\beta)}{\partial \beta_j} = 0$ and solving for β_j^{new}:

$$\beta_j^{new} \leftarrow \frac{S\left(\frac{1}{N}\sum_{i=1}^{N} x_{ij}\left(y_i - \tilde{y}_i^{(j)}\right), \lambda\alpha\right)}{1 + \lambda(1-\alpha)}, (6)$$

   where $\tilde{y}_i^{(j)} = \tilde{\beta}_0 + \sum_{l \neq j} x_{il}\tilde{\beta}_l$ is the fitted value excluding the $x_{ij}\tilde{\beta}_j$ term, and $S(z, \lambda\alpha)$ in the numerator of (6) is the *soft-thresholding operator* which arises from taking the derivative of the absolute value penalty term $|\beta_j|$ in (5):

$$S(z, \lambda\alpha) = \begin{cases} z - \lambda\alpha \text{ if } z > 0 \text{ and } \lambda\alpha < |z| \\ z + \lambda\alpha \text{ if } z < 0 \text{ and } \lambda\alpha < |z| \\ 0 \text{ if } \lambda\alpha \leq |z|. \end{cases}$$

   Note that once a parameter estimate has been reduced to zero, that parameter need no longer be updated in the inner loop. For the first iteration of the outer loop, the initial estimates of $\tilde{\beta}_0, \tilde{\beta}_1, \ldots, \tilde{\beta}_p$ may be found using `proc reg`. For subsequent iterations the parameter estimates may be obtained from the prior solution to λ_{t-1}. During following iterations of the middle loop the current values of $\tilde{\beta}_0, \tilde{\beta}_1, \ldots, \tilde{\beta}_p$ are used as initial estimates.
4. Set $\tilde{\beta}_j = \beta_j^{new}$ and repeat step 3 to update the next coefficient β_{j+1}^{new} .

<u>Logistic Regression Coordinate Descent</u>

The coordinate descent algorithm for logistic regressions finds the iteratively reweighted least squares (IRLS) solution for the penalized Taylor approximation of the log-likelihood of the logistic regression model $\log[E\left(\frac{\hat{p}}{1-\hat{p}}\right)] = \beta_0 + X^T\beta$ by minimizing the following function:

$$\min_{\beta}\left\{-loglike_Q(\boldsymbol{\beta}) = \frac{1}{2N}\sum_{i=1}^{N} w_i\left(z_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + C\left(\tilde{\boldsymbol{\beta}}\right)^2 + \lambda\sum_{j=1}^{p}\left[\frac{1}{2}(1-\alpha)\beta_j^2 + \alpha|\beta_j|\right]\right\}, (7)$$

where $z_i = \tilde{\beta}_0 + X^T\tilde{\boldsymbol{\beta}} + \frac{y_i - \tilde{p}(x_i)}{\tilde{p}(x_i)(1-\tilde{p}(x_i))}$ is the working response for observation i, $w_i = \tilde{p}(x_i)(1-\tilde{p}(x_i))$ is the weight for observation i, $\tilde{p}(x_i) = \Pr(Y=1|X=x) = \frac{1}{1+e^{-(\tilde{\beta}_0 + X^T\tilde{\boldsymbol{\beta}})}}$ is the predicted probability, and $C(\tilde{\boldsymbol{\beta}})^2$ is a constant term. All terms are evaluated at the current estimates of $\tilde{\beta}_0, \tilde{\beta}_1, \ldots, \tilde{\beta}_p$. As with the least squares solution, the predictor variables are standardized.

The coordinate descent procedure for the logistic regression follows the same steps as the least squares procedure outlined above, with the exception that the coordinate update formula (6) is modified to account for the weights:

$$\beta_j^{new} \leftarrow \frac{S\left(\frac{1}{N}\sum_{i=1}^{N} w_i x_{ij}\left(z_i - \tilde{z}_i^{(j)}\right), \lambda\alpha\right)}{\sum_{i=1}^{N} w_i x_{ij}^2 + \lambda(1-\alpha)}, (8)$$

using the same soft-thresholding operator as with the least squares update step, where $\tilde{z}_i^{(j)}$ is calculated as z_i but omitting the $\beta_j x_{ij}$ term.

SAS CODE

The SAS code for the cyclical coordinate descent algorithms minimizes overhead and execution time by reading the data into temporary arrays and then updating parameter estimates by cycling through the arrays within a single data step. Hence, contingent upon the data fitting within your machine's memory, large datasets may be processed efficiently.

```
*************************************************************************
*      OLS regression coordinate descent code for lasso (alpha=1)     *
*************************************************************************;

*** Declare macro variables ***;
%let nobs=1000000;
%let numvars=10;
%let numiter=1000;
%let lambdanum=12;
%let alphanum=1;

*** Declare pregenerated macro variables for initial parameter values and variable
mean, stnd deviation values ***;
%macro parmlist; %do i = 0 %to &numvars; &&p&i %end; %mend parmlist;
%macro meanlist; %do i = 0 %to &numvars; &&mean&i %end; %mend meanlist;
%macro stdlist; %do i = 0 %to &numvars; &&std&i %end; %mend stdlist;

*** Begin data step ***;
data coord_descent_output (keep=alpha lambda parm_unstnd_1-parm_unstnd_&numvars.);

*** Declare arrays ***;
array xx[&numvars.] x1-x&numvars.;
array x_[&nobs.,&numvars.] _temporary_;
array y_[&nobs] _temporary_;
array p_[0:&numvars.] (%parmlist);
array z_[&numvars.] (&numvars.*0);
array yhat_[&numvars.] (&numvars.*0);
array mean_[0:&numvars.] (%meanlist);
array std_[0:&numvars.] (%stdlist);
array parm_unstnd_[0:&numvars.] (%eval(&numvars. + 1)*0);
array lambda_[1:12] (.001 .005 .01 .05 .1 .2 .5 1 10 20 50 100);
array alpha_[1] (1);
```

```
*** Load X and y data into two and one dimensional arrays ***;
do _n_ = 1 to &nobs.;
  set training_sample nobs=nobs;
  do j=1 to &numvars.;
    x_[_n_,j] = xx[j];
    y_[_n_] = y;
  end;
end;

*** Coordinate descent routine ***;
do g=1 to &alphanum; * Outer loop for alpha ;
do h=1 to &lambdanum; * Outer loop for lambda ;
do i=1 to &numiter; * Middle loop ;
do j=1 to &numvars; * Inner loop ;
  if p_[j]^=0 then do; * Bypass calculations and proceed to p_[j+1] if p_[j]=0 ;
    z_[j] = 0;
    do _nn_ = 1 to &nobs; * Loop across observations to calculate summations;
       yhat_[j] = p_[0] - p_[j]*x_[_nn_,j];
       do k = 1 to &numvars;
         yhat_[j] = sum(yhat_[j], p_[k]*x_[_nn_,k]);
       end;
       z_[j] = sum(z_[j], (y_[_nn_] - yhat_[j])*x_[_nn_,j]);
    end;

    * Soft-thresholding operator;
    if (z_[j]/&nobs > 0 and alpha_[g]*lambda_[h] < abs(z_[j]/&nobs))
       then p_[j] = (z_[j]/&nobs - alpha_[g]*lambda_[h])/(1 + lambda_[h]
       alpha_[g]*lambda_[h]);
    else if (z_[j]/&nobs < 0 and alpha_[g]*lambda_[h] < abs(z_[j]/&nobs))
       then p_[j] = (z_[j]/&nobs + alpha_[g]*lambda_[h])/(1 + lambda_[h] -
       alpha_[g]*lambda_[h]);
    else if alpha_[g]*lambda_[h] >= abs(z_[j]/&nobs) then p_[j] = 0;
  end; * End obs loop ;
end; * End p_[j]=0 bypass loop ;
end; * End inner loop ;
end; * End middle loop ;

*** Calculate "destandardized" regression parameters and output results ***;
parm_unstnd_[0] = p_[0];
do l=1 to &numvars.;
       parm_unstnd_[0] = parm_unstnd_[0] - p_[l]*mean_[l]/std_[l];
       parm_unstnd_[l] = std_[0]*p_[l]/std_[l];
end;
parm_unstnd_[0] = std_[0]*parm_unstnd_[0] + mean_[0];
alpha=alpha_[g];
lambda=lambda_[h];
output coord_descent_output;

end; * End outer loop for lambda ;
end; * End outer loop for alpha ;

run;
```

```
*************************************************************************
*      Logistic regression coordinate descent code for lasso (alpha=1)     *
*************************************************************************;

*** Declare macro variables ***;
%let nobs=1000000;
%let numvars=10;
%let numiter=1000;
%let lambdanum=12;
%let alphanum=1;

*** Declare pregenerated macro variables for initial parameter values and variable
mean, stnd deviation values ***;
%macro parmlist; %do i = 0 %to &numvars; &&p&i %end; %mend parmlist;
%macro meanlist; %do i = 0 %to &numvars; &&mean&i %end; %mend meanlist;
%macro stdlist; %do i = 0 %to &numvars; &&std&i %end; %mend stdlist;

*** Begin data step ***;
data coord_descent_output (keep=alpha lambda parm_unstnd_1-parm_unstnd_&numvars.);

*** Declare arrays ***;
array xx[&numvars.] x1-x&numvars.;
array x_[&nobs.,&numvars.] _temporary_;
array y_[&nobs] _temporary_;
array p_[0:&numvars.] (%parmlist);
array z_[&numvars.] (&numvars.*0);
array yhat_[&numvars.] (&numvars.*0);
array mean_[0:&numvars.] (%meanlist);
array std_[0:&numvars.] (%stdlist);
array parm_unstnd_[0:&numvars.] (%eval(&numvars. + 1)*0);
array lambda_[1:12] (.001 .005 .01 .05 .1 .2 .5 1 10 20 50 100);
array alpha_[1] (1);

*** Load X and y data into two and one dimensional arrays ***;
do _n_ = 1 to &nobs.;
  set training_sample nobs=nobs;
  do j=1 to &numvars.;
    x_[_n_,j] = xx[j];
      y_[_n_] = y;
  end;
end;

*** Coordinate descent routine ***;
do g=1 to &alphanum; * Outer loop for alpha ;
do h=1 to &lambdanum; * Outer loop for lambda ;
do i=1 to &numiter; * Middle loop ;
  do j=1 to &numvars; * Inner loop ;
  if p_[j]^=0 then do; * Bypass calculations and proceed to p_[j+1] if p_[j]=0 ;
       z_j = 0; z_ne_j = 0; z = 0; sum_wtx_sq = 0;
       do _nn_ = 1 to &nobs; * Loop across observations to calculate summations;
          yhat = p_[0];
          do k=1 to &numvars;
                yhat = sum(yhat, p_[k]*x_[_nn_,k]);
          end;

          * Calculate weights within minimum and maximum limits to avoid coefficient
            divergence issues ;
          proby = 1/(1 + exp(-yhat));
          proby_ne_j = 1/(1 + exp(-(yhat - p_[j]*x_[_nn_,j])));
          if proby <= .00001 then do;
                proby = 0;
                weight = .00001;
          end;
```

```
            else if proby >= .99999 then do;
                  proby = 1;
                  weight = .00001;
            end;
            else weight = proby*(1 - proby);
            if proby_ne_j <= .00001 then do;
                  proby_ne_j = 0;
                  weight_ne_j = .00001;
            end;
            else if proby_ne_j >= .99999 then do;
                  proby_ne_j = 1;
                  weight_ne_j = .00001;
            end;
            else weight_ne_j = proby_ne_j*(1 - proby_ne_j);

            * Soft-thresholding operator ;
            z_j = yhat + (y_[_nn_] - proby)/weight;
            z_ne_j = yhat - p_[j]*x_[_nn_,j] + (y_[_nn_] - proby_ne_j)/weight_ne_j;
            z = sum(z, weight*x_[_nn_,j]*(z_j - z_ne_j));
            sum_wtx_sq = sum(sum_wtx_sq, weight*(x_[_nn_,j])**2);
      end; * End obs loop ;
    end; * End p_[j]=0 bypass loop ;
    end; * End inner loop ;
end; * End middle loop ;

*** Calculate "destandardized" regression parameters and output results ******;
parm_unstnd_[0] = p_[0];
do l=1 to &numvars.;
       parm_unstnd_[0] = parm_unstnd_[0] - p_[l]*mean_[l]/std_[l];
       parm_unstnd_[l] = p_[l]/std_[l];
end;
alpha=alpha_[g];
lambda=lambda_[h];
output coord_descent_output;

end; * End outer loop for lambda ;
end; * End outer loop for alpha ;

run;
```

CONCLUSION

My hope in translating the coordinate descent method into SAS is to make a revolutionary, but perhaps obscure, algorithm a little more accessible and transparent to SAS users without invoking additional modules beyond Base SAS®. This is very much a work in progress, and I hope to make further refinements to the code in the future. Writing a coordinate descent routine for Poisson regression will be a priority. Please feel free to contact me if you have questions or corrections.

REFERENCES

Friedman, J., Hastie, T., Höfling, H. and Tibshirani, R. (2007). “Pathwise Coordinate Optimization.” *The Annals of Applied Statistics*. Vol. 1, No. 2, pp. 302-332.

Friedman, J., Hastie, T., and Tibshirani, R. 2010. “Regularization Paths for Generalized Linear Models via Coordinate Descent.” *Journal of Statistical Software*, Volume 33 Issue 1.

Fu, W. 1998. “Penalized regressions: the bridge vs. the lasso.” *Journal of Computational and Graphical Statistics*, Volume 7, Issue 3, pp. 397-416.

Hastie, T., Tibshirani, R., and Friedman, J. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. New York City, NY: Springer Series in Statistics.

Tibshirani, R. 1996. “Regression Shrinkage and Selection via the Lasso.” *Journal of the Royal Statistical Society. Series B (Methodological)*, Volume 58 Issue 1, pp. 267-288.

Zou, H. and Hastie, T. 2005. “Regularization and variable selection via the elastic net.” *Journal of the Royal Statistical Society. Series B (Methodological)*, Volume 67 Issue 2, pp. 301-320.

ACKNOWLEDGMENTS

Many thanks go to the statisticians and programmers who frequently post on the SAS-L listserv website and the Cross Validated forum on the Stack Exchange website. They graciously answered my questions and offered assistance with the intricacies of SAS coding. In particular, I’m thankful for the clever solution to reading data into an array offered by Dan Nordlund.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Robert Feyerharm
Beacon Health Options
(860) 263-2039
Robert.Feyerharm@valueoptions.com

ATA is not a SAS data set.