Elastic Net and Variable Selection

Part A: Weighted Elastic Nets

We will be considering a regularized linear regression cost function of the form [1] :

$$J_1(\lambda, \beta) = ||y - X\beta||_2^2 + \lambda_2||\beta||_2^2 + \lambda_1||\beta||_1 \tag{1}$$

Where λ_2 controls the ridge regression, and λ_1 the lasso regularization. Here y is n-dimensional, and β will be d-dimensional. Thus the regularized solution for β becomes:

$$\hat{\beta} = \underset{\beta}{argmin} J(\lambda, \beta) \tag{2}$$

1.

In this 'naive' elastic net model, we can rewrite the loss as a regular lasso loss with augmented data. Consider the following alterations to the data:

$$\tilde{X} = c\begin{pmatrix} X \\ \sqrt{\lambda_2} I_d \end{pmatrix}, \quad \tilde{y} = \begin{pmatrix} y \\ 0_d \end{pmatrix} \tag{3}$$

Where I_d is the identity d x d matrix, and $\tilde{y}$ is now n+d dimensional, and for simplicity later, we assume $c = (1 + \lambda_2)^{-1/2}$. If we further assume that $\beta = c\tilde{\beta}$, we can write an augmented lasso regularization:

$$J_2(\tilde{\beta}) = ||\tilde{y} - \tilde{X}\tilde{\beta}||_2^2 + c\lambda_1||\tilde{\beta}||_1 \tag{4}$$

We wish to show that this is equal to the original naive elastic net loss. First we note that c is strictly positive, and so we can take it in/out of absolute values. Let's first expand $J_1(c\beta)$:

$$\begin{aligned} J_1(c\beta) &= (y - Xc\beta)^T(y - Xc\beta) + \lambda_2 c^2\beta^T\beta + \lambda_1||c\beta||_1 \\ &= y^Ty + c^2\beta^TX^TX\beta - 2cy^TX\beta + \lambda_2 c^2\beta^T\beta + c\lambda_1||\beta||_1 \end{aligned} \tag{5}$$

Keeping in mind the solution for β is:

$$\hat{\beta} = \underset{\beta}{argmin} J_1(\beta) \tag{6}$$

[1] Note that some people prefer to weight the OLS cost by 1/n. This isn't necessary (although sometimes numerically nicer), but just rescales the values of λ_1 and λ_2, which we are free to choose anyways.

Next, let's expand $J_2(\beta)$:

$$\begin{aligned} J_2(\beta) &= (\tilde{y} - \tilde{X}\beta)^T(\tilde{y} - \tilde{X}\beta) + c\lambda_1||\beta||_1 \\ &= \tilde{y}^T\tilde{y} + \beta^T\tilde{X}^T\tilde{X}\beta - 2\tilde{y}^T\tilde{X}\beta + \lambda_1 c||\beta||_1 \\ &= \begin{pmatrix} y^T & 0_d^T \end{pmatrix}\begin{pmatrix} y \\ 0_d \end{pmatrix} + \beta^T c\begin{pmatrix} X^T & \sqrt{\lambda_2}I_d \end{pmatrix} c\begin{pmatrix} X \\ \sqrt{\lambda_2}I_d \end{pmatrix}\beta - 2\begin{pmatrix} y^T & 0_d^T \end{pmatrix} c\begin{pmatrix} X \\ \sqrt{\lambda_2}I_d \end{pmatrix}\beta + \lambda_1 c||\beta||_1 \\ &= y^Ty + c^2\beta^T\left[X^TX + \lambda_2 I_d^2\right]\beta - 2cy^TX\beta + \lambda_1 c||\beta||_1 \\ &= y^Ty + c^2\beta^TX^TX\beta - 2cy^TX\beta + \lambda_2 c^2\beta^T\beta + c\lambda_1||\beta||_1 \end{aligned} \tag{7}$$

And so we see that $J_1(c\beta) = J_2(\beta)$. Thus if we minimize $J_2(\beta)$ with respect to β, this is the same as minimizing $J_1(c\beta)$ with respect to β.

Thus we see that:

$$\begin{aligned} \underset{\beta}{argmin} J_1(c\beta) &= \underset{\beta}{argmin} J_2(\beta) \\ \frac{1}{c}\underset{\beta}{argmin} J_1(\beta) &= \underset{\beta}{argmin} J_2(\beta) \\ \hat{\beta} &= c(\underset{\beta}{argmin} J_2(\beta)) \end{aligned} \tag{8}$$

Where we have used the fact that, letting $z = ax$:

$$\begin{aligned} \hat{x_1} = \underset{x}{argmin} f(x) \rightarrow df(\hat{x_1})/dx = 0 \\ \hat{x_2} = \underset{x}{argmin} f(ax) \rightarrow df(a\hat{x_2})/dx = 0 \\ df(a\hat{x_2})/dx = df(\hat{z})/dz \cdot dz/dx = adf(\hat{z})/dz = 0 \end{aligned}$$

Which implies that $\hat{z} = \hat{x_1}$ giving us $a\hat{x_2} = \hat{x_1}$, or:

$$\frac{1}{a}\underset{x}{argmin} f(x) = \underset{x}{argmin} f(ax) \tag{9}$$

Thus, if we find the solution to J_2, we can simply rescale this by c to find the true solution to the elastic net:

$$\hat{\beta}_{EN} = c\hat{\beta}_{Lasso} \tag{10}$$

2.

We now wish to consider the **weighted** elastic net (WEN). To start, we simply write out the loss function, and minimization problem for the WEN. We recall that weighted-least squares

has the form[2]:

$$\begin{aligned} RSS &= (y - X\beta)^T W (y - X\beta) \\ &= ||W^{1/2}(y - X\beta)||_2^2 \end{aligned} \tag{11}$$

Where W is a diagonal matrix, and each entry is the reciprocal of the variance for each observed response:

$$W_{ii} = w_i = \frac{1}{\sigma_i^2} \tag{12}$$

We may simply augment this with the elastic net regularizers to determine the loss function:

$$J_1(\beta) = ||W^{1/2}(y - X\beta)||_2^2 + \lambda_2||\beta||_2^2 + \lambda_1||\beta||_1 \tag{13}$$

Where we have a similar minimization problem as before:

$$\hat{\beta} = \underset{\beta}{argmin} J_1(\beta) \tag{14}$$

3.

Next we show that this can also be written as a lasso regression problem as well. To do this, we follow the intuition of question 1. However, first we rewrite y and X:

$$\bar{X} = W^{1/2}X, \quad \bar{y} = W^{1/2}y \tag{15}$$

So that we have:

$$J_1(\beta) = ||\bar{y} - \bar{X}\beta||_2^2 + \lambda_2||\beta||_2^2 + \lambda_1||\beta||_1 \tag{16}$$

This is now identical to the original elastic net formulation, and so we can use the same augmentation to find a lasso solution:

$$J_2(\tilde{\beta}) = ||\tilde{y} - \tilde{X}\tilde{\beta}||_2^2 + c\lambda_1||\tilde{\beta}||_1 \tag{17}$$

Where we now have:

$$\tilde{X} = c\begin{pmatrix} W^{1/2}X \\ \sqrt{\lambda_2}I_d \end{pmatrix}, \quad \tilde{y} = \begin{pmatrix} W^{1/2}y \\ 0_d \end{pmatrix} \tag{18}$$

And:

$$\hat{\beta}_{EN} = c\hat{\beta}_{Lasso} \tag{19}$$

as before.

[2]Note that if we wish to compare between test and training data, the RSS must be divided by the sum of the weights to make a fair comparison.

4.

An improved version of the elastic net 'rescales' the parameter solution to partially undo the shrinkage that is caused by including the regularizing functions. This is done by rescaling the lasso solution to give the new optimal solution:

$$\begin{aligned} \hat{\beta} &= \sqrt{1+\lambda_2}\hat{\beta}_{Lasso} \\ &= (1+\lambda_2)\hat{\beta}_{EN} \end{aligned} \tag{20}$$

We may do the same thing with the WEN and scale $\hat{\beta}$ in the same way. Lets do this, and reconsider the cost function we are minimizing:

$$\begin{aligned} J_1(\beta) &= ||\bar{y} - \bar{X}\beta||_2^2 + \lambda_2||\beta||_2^2 + \lambda_1||\beta||_1 \\ &= (y - X\beta)^T W(y - X\beta) + \lambda_2\beta^T\beta + \lambda_1||\beta||_1 \\ &= y^T W y + \beta^T X^T W X\beta - 2y^T W X\beta + \lambda_2\beta^T\beta + \lambda_1||\beta||_1 \\ &= y^T W y + \beta^T(X^T W X + \lambda_2 I_d)\beta - 2y^T W X\beta + \lambda_1||\beta||_1 \end{aligned} \tag{21}$$

Now note that if we let $\hat{\beta}$ be the optimal solution to some loss, J, we find that it must be:

$$\begin{aligned} \hat{\beta_{EN}} &= \underset{\beta}{argmin} J_1(\beta) \\ \hat{\beta} &= (1+\lambda_2)\underset{\beta}{argmin} J_1(\beta) \\ &= \underset{\beta}{argmin} J_1\left(\frac{\beta}{1+\lambda_2}\right) \end{aligned} \tag{22}$$

Thus, the improved version of the elastic net will minimize (dropping terms that are constant with respect to β and rescaling the entire function by an overall constant):

$$J(\beta) = \beta^T\left(\frac{X^T W X + \lambda_2 I_d}{1+\lambda_2}\right)\beta - 2y^T W X\beta + \lambda_1||\beta||_1 \tag{23}$$

Part B: WEN Gradient

Because of the presence of $||\beta||_1$ in our loss function, it is not differentiable when any $\beta_j = 0$. However, we can still determine the subdifferential set for J, which defines the set of all tangent vectors that produce lines that lie entirely below the function for a given point. If a function is differential, then the subdifferential set reduces to the value of the gradient at that point. First we note that the rest of the cost function is differential, so we will deal with

this first. To make our lives slightly simpler, we will redefine a few matrices and vectors:

$$A \equiv 2\frac{X^T W X + \lambda_2 I_d}{1+\lambda_2} \tag{24}$$

$$b \equiv 2X^T W y \tag{25}$$

$$J(\beta) = \frac{1}{2}\beta^T A\beta - b^T\beta + \lambda_1||\beta||_1 \tag{26}$$

$$= J_A(\beta) + J_B(\beta) \tag{27}$$

The first part of the gradient becomes:

$$\frac{\partial J_A}{\partial \beta} = A\beta - b \tag{28}$$

Or in the more useful component form:

$$\begin{aligned}\frac{\partial J_A}{\partial \beta_j} &= \sum_i A_{ji}\beta_i - b_j \\ &= A_{jj}\beta_j + \sum_{i\neq j} A_{ji}\beta_i - b_j \\ &= a_j\beta_j - c_j \end{aligned} \tag{29}$$

Where we have defined new parameters that do not depend on β_j:

$$a_j = A_{jj}, \quad c_j = -\sum_{i\neq j} A_{ji}\beta_i + b_j \tag{30}$$

We can now put this together with the absolute value. Here, we use the definition of the subgradient for $|x|$:

$$\partial_x|x| = \begin{cases}\{-1\} & x<0 \\ [-1,1] & x=0 \\ \{1\} & x>0\end{cases} \tag{31}$$

Thus, the subdifferential becomes:

$$\partial_j(J) = a_j\beta_j - c_j + \lambda_1\partial_j|\beta_j| \tag{32}$$

$$= \begin{cases}\{a_j\beta_j - c_j - \lambda_1\} & \beta_j<0 \\ [-c_j-\lambda_1, -c_j+\lambda_1] & \beta_j=0 \\ \{a_j\beta_j - c_j + \lambda_1\} & \beta_j>0\end{cases} \tag{33}$$

Depending on the value of c_j, this leads to three distinct solution regions for β_j that lead to 0 being in the subdifferential set for J:

$$\beta_j = \begin{cases} \frac{c_j+\lambda_1}{a_j} & c_j < -\lambda_1 \\ 0 & -\lambda_1 \leq c_j \leq \lambda_1 \\ \frac{c_j-\lambda_1}{a_j} & c_j > \lambda_1 \end{cases} \tag{34}$$

Although we cannot explicitly solve for β_j analytically, we can see that β_j will still do variable selection, as some subset of β_j will typically be chosen to be zero as long as the condition for c_j is satistifed. If we choose a larger λ_1, more parameters will be forced to zero.

Part C: Optimization

Now that we have a form for the (sub)-gradient of our loss function, we wish to actually iteratively solve for the correct solution, given parameters $\lambda_{1,2}$. In the accompanying notebook, we consider three different methods. In all three methods, we initialize the starting β using ordinary least squares (otherwise I find that the solutions don't always converge). It's also worth nothing that at this point, there's been no mention of the intercept that is typically present (and un-regularized), when doing a regression. This can be dealt with in two ways: assume the data is centered, so that $\sum y_i = 0$ and thus $\beta_0 \sim 0$, or iteratively solving for it. Because it does not show up in the regularizers, the loss-minimizing solution for β_0 is:

$$\beta_0 = \frac{1}{n}\sum_i (y_i - (X\beta)_i) \tag{35}$$

Thus we can also do gradient descent steps where we first update β_0, then update β using a modified $\tilde{y} = y - \beta_0$, and proceed using the methods discussed below.

1a. Sub-gradient Descent

In this method, we simply just do gradient descent, where instead of using the gradient, we use *any* subgradient within the set. Because this method does not necessarily always cause a reduction in the overall loss, we also need to keep track of whether a previous step was actually a better solution. The subgradient is chosen so that $\partial|x| = sign(x)$, where

$sign(0) \equiv 0$ is a typical choice for the non-differential position.

Algorithm 1: Sub-Gradient Descent

Result: Optimized β_{best}
initialize $\lambda_1,\ \lambda_2,\ max\_iter,\ \alpha$;
initialize β using OLS;
initialize $J_{best} = J(\beta), \beta_{best} = \beta$;
initialize iter $= 0$;
while $iter < max\_iter$ **do**
  $g^{(k)} = \partial J(\beta^{(k)})$;
  $\beta^{(k+1)} = \beta^{(k)} - \alpha \cdot g^{(k)}$;
  if $J(\beta^{(k+1)}) < J_{best}$ **then**
    $J_{best} = J(\beta^{(k+1)})$;
    $\beta_{best} = \beta^{(k+1)}$;
  end
  $iter+ = 1$;
end

1b. Coordinate Descent

This method uses the idea that, for a given β_j, we can determine an analytic solution for β_j, if we assume the rest of the $\beta_{i\neq j}$ are held fixed. These solutions were given explicitly in Eq. 34. Note that the condition for β_j can be written more compactly, using the soft function notation:

$$\beta_j = soft(c_j/a_j, \lambda_1/a_j) \tag{36}$$

$$soft(a,b) \equiv sign(a) \cdot (|a| - \lambda_1)_+ \tag{37}$$

Thus the coordinate descent algorithm becomes:

Algorithm 2: Coordinate Descent

Result: Optimized β_{best}
initialize $\lambda_1,\ \lambda_2,\ max\_iter$;
initialize β using OLS;
initialize iter $= 0$;
while $iter < max\_iter$ **do**
  for j *in shuffle(1:d)* **do**
    $\beta_j = soft(c_j/a_j, \lambda_1/a_j)$
  end
  $iter+ = 1$;
end

This ends up being much faster than subgradient descent, only taking a few iterations over all the β_j to converge on a solution.

1c. Proximal Gradient Descent

Finally, we have proximal gradient descent, where instead of directly minimizing and moving in the direction of a subgradient, we move towards the differential gradient, while attempting to remain close to the previous solution. That is, if we have a loss function of the form:

$$f(x) = L(x) + R(x) \tag{38}$$

Where L is differentiable and R is not, (but both are concave), we can define a proximal operator:

$$prox_R(y) = \underset{z}{argmin} \left(R(z) + \frac{1}{2}||z - y||_2^2 \right) \tag{39}$$

For an absolute value function, $R(x) = \lambda|x|$, the proximal operator is:

$$prox_R(x) = soft(x, \lambda) \tag{40}$$

The value of the proximal operator will be close to the previous solution, if we minimize a quadratic approximation to the loss, where at each step it is centered on the previous solution. Explicitly, this becomes:

Algorithm 3: Proximal Gradient Descent

Result: Optimized β
initialize $\lambda_1,\ \lambda_2,\ max\_iter,\ \alpha$;
initialize β using OLS;
initialize iter = 0;
while $iter < max\_iter$ **do**
  $g^{(k)} = \partial J_A(\beta^{(k)})$;
  $u^{(k)} = \beta^{(k)} - \alpha g^{(k)}$;
  $\beta^{(k+1)} = prox_{\alpha,R}(u^{(k)}) = soft(u^{(k)}, \alpha\lambda_1)$;
  $iter+ = 1$;
end

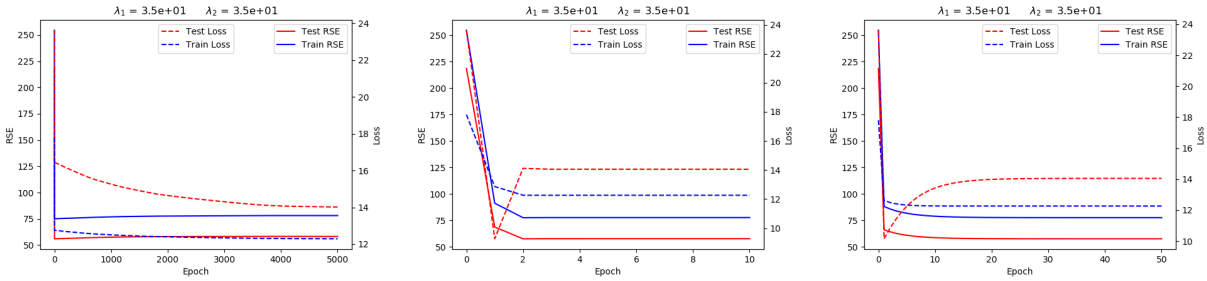
Where J_A is the differentiable component of the loss function, as we defined above. This solution also converges quickly, and has the added bonus that this minimizes all the β_j at once, unlike coordinate descent that must do one at a time.

2, 3, 4.

In the next few steps we mostly refer to the notebook where the code was run. The **Boston** dataset is used. The data is split into a training and test set, with a 70/30 split. To determine

the weights to use for the weighted elastic net, a basic OLS fit was done, and the absolute value of the residuals were regressed onto the initial target values. The weights were defined to be:

$$w_i = \frac{1}{residual(y_i)^2} \tag{41}$$



(a) Subgradient Descent (b) Coordinate Descent (c) Proximal Descent

Figure 1: Different optimization methods.

The results for representative values of λ_1 and λ_2 are shown in Fig. 1, while average results over a grid of λ_1 and λ_2 are shown in Fig. 2. The λ_i values were taken from the range [0.01, 300]. Typically, subgradient descent converged much slower than both of the other methods, and was also incapable of producing the same sparsity (number of zero parameters), as the other two methods. The other two methods appeared to be much more similar, with similar convergence rates, fitting times, and sparsity.Proximal gradient descent was a much smoother fit, whereas coordinate gradient descent tended to jump more erratically from epoch to epoch (likely because each epoch actually involves looping over all the parameters, so there can be drastic changes within this sub-loop).

	λ_1	λ_2	SGD time	CD time	PGD time	SGD loss	CD loss	PGD loss	SGD sparsity	CD sparsity	PGD sparsity
count	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.00	100.000000	100.000000
mean	44.444000	44.444000	1.930064	0.013239	0.037026	1606.735013	1589.385900	1591.487480	0.01	3.580000	3.420000
std	90.597015	90.597015	0.233598	0.002641	0.012223	657.411098	638.784197	638.671336	0.10	4.987216	4.979108
min	0.010000	0.010000	1.672151	0.009973	0.026930	365.321303	368.098663	363.818800	0.00	0.000000	0.000000
25%	0.100000	0.100000	1.774460	0.011967	0.031913	1030.763454	1038.398131	1038.312044	0.00	0.000000	0.000000
50%	2.000000	2.000000	1.857032	0.012965	0.033907	1678.517666	1735.205989	1735.140304	0.00	0.000000	0.000000
75%	30.000000	30.000000	1.986435	0.013962	0.036902	2119.452030	2043.149874	2043.149874	0.00	8.000000	7.250000
max	300.000000	300.000000	2.943127	0.024933	0.110704	2458.463053	2497.014517	2496.964023	1.00	12.000000	12.000000

Figure 2: Optimizer averages over a grid of λ_1 and λ_2 values. Time corresponds to the amount of time needed to fit the model, loss refers to the test loss, and sparsity refers to the number of parameters that the model set to zero.

Moving forward, we will use the quickest method, coordinate descent, which seems to always converge within as little as 5 epochs.

Part D: Model Validation & Parallelization

1. Model Selection

Up next, we wish to choose the best model. We will do this using parallelized cross-validation. This is done using a grid-search over various different values of λ_1 and λ_2, and within each pair, using 10-fold cross-validation to determine the average RSE on unseen data. I chose these parameters (after some trial and error) to be in the range $[10^{-2}, 10^2]$, with logarithmically spaced samples. I also chose 30 sample points within each λ, for a total of 900 different models to scan over. This was parallelized so that each run over different $\lambda 1, 2$ pairs could be run independently, with all the resulting validation errors collected at the end. The best model could then be chosen as that with the lowest cross-validation error. Some summary results are shown below.

We learn a few important things. First, parallelization greatly speeds up the computation, as the entire grid search took ∼434 s. If we had done this without parallelizing, this would have taken much longer. Next, we also see that the model seems to preferentially prefer smaller λ, as we can see from the contour plot in Fig. 3. This might make sense, as the model is already fairly simple (with only 14 parameters but hundreds of data points), and so we may not expect much regularization to be needed to prevent over-fitting. Here we see that the preferred values tend to be pushed to the bottom left, although there is still some possibility of a large λ_1, as we see by the horizontal line that does not cross over until $\lambda_1 \sim 5$. If we look at the lowest possible RSE obtained from the grid search, this was found at $\Lambda_1 = 0.014,\ \lambda_2 = 0.01$, with an average RSE score of 2.16. These were near to the smallest parameters in our grid search, indicating it might have preferred to go even lower.

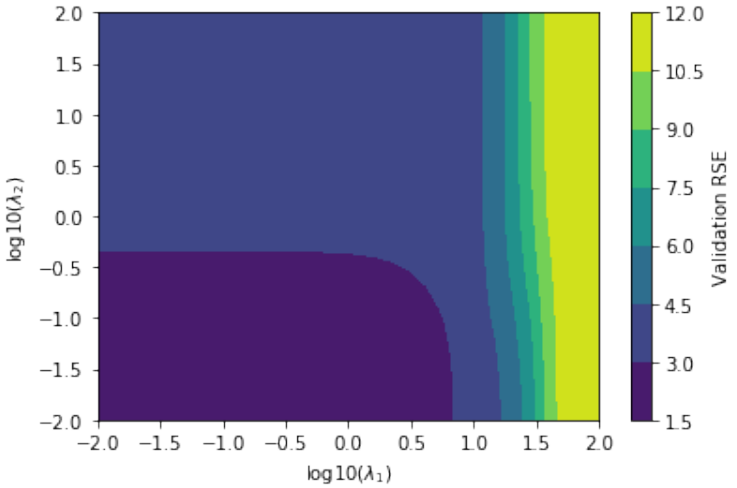


Figure 3: Contour plot for the grid search over the $\lambda_{1,2}$ parameter space. Shown is the average validation accuracy obtained from a 10-fold CV. The preference seems to be to slightly drive both λ to zero, although there is still the possibility of a reasonably large λ_1.

So overall, we find that the parallelized cross-validation over 900 grid-search models took 392s, and determined that small λ is best, but that there is some wiggle room for a large λ_1.

2. Confidence Intervals

Next we perform bootstrapping to obtain confidence intervals around λ_1 and λ_2. If we create B bootstrap samples of the original dataset (with replacement), we can calculate the average value of each parameter, $\bar{\lambda}$, and include a standard error on each parameter of the form:

$$SE_B(\lambda) = \sqrt{\frac{1}{B-1}\sum_{r=1}^{B}\left(\lambda_r - \bar{\lambda}\right)^2} \tag{42}$$

Note that in terms of actually parallelizing this, it seems much more difficult. We effectively have nested for-loops, which do not seem to play well with parallelization. I don't know the optimal method, so for now I will just do un-rolled for-loops, in which I calculate all of the samples and all of the λ all at once, and then at the end match up the correct samples to get the best λ for each sample, before averaging over all the best λ. I am sure there is a better way, I just don't know what it is yet.

The other problem with this method will be that we were hitting the edge of the grid search previously, and so we might need to alter our grid search to a more suitable region. For now I will do 20 bootstrap samples, with a 20x20 grid search spanning $[10^{-4}, 1]$. Honestly kind of stuck on how to go about this piece.

For my first run through, this took 3811 s (or just over an hour). The average results for each hyperparameter are shown in Fig. 4. We see that the optimal results for λ_1 and λ_2 are given by:

$$\lambda_1 = 0.005 \pm 0.009 \tag{43}$$
$$\lambda_2 = 0.002 \pm 0.002 \tag{44}$$

This gives us approximate 95% confidence intervals of [0, 0.023] and [0, 0.006], for λ_1 and λ_2, respectively (where I've truncated the lower end at 0 and used 2σ as the standard 95% bound for the upper end).

3. Combining Bootstrap Models

We could also use the bootstrap method to combine the model parameters as well if we wished. Because elastic-net leads to sparse solutions, I think you could do two things here.

The first would be variable selection. For each bootstrap model, a coefficient would either be set to zero or not. I would consider any coefficient 'eliminated' if they were removed in some percentage of all the bootstrap models (say 50%).

	sample_id	Best λ_1	Best λ_2	Best RSE
count	20.00000	20.000000	20.000000	20.000000
mean	9.50000	0.004891	0.001801	1.974548
std	5.91608	0.008720	0.002222	0.274794
min	0.00000	0.000162	0.000100	1.575216
25%	4.75000	0.000264	0.000162	1.699069
50%	9.50000	0.001129	0.000695	1.941428
75%	14.25000	0.002583	0.002976	2.143093
max	19.00000	0.033598	0.007848	2.548993

Figure 4: Bootstrap statistics for the bootstrap grid search over the hyperparameters.

Then we are left with the rest of the parameters. One option would be to average them to get coefficient values. I'm not sure it makes sense to just average them (as we have thrown out possibly relevant data in certain fits). Another possibility would be to refit the data using ridge-regression (or perhaps elastic-net again) using only the surviving parameters. Or perhaps we just leave all the parameters in from the beginning, and remove the lowest average values as being 'eliminated.' It seems like there could be many ways to interpret these coefficients.

Part E: Variable Selection

Next up we consider variable selection. Fig. 5 shows the coefficient paths for each parameter. The best fit value of $\lambda_2 = 0.002$ was used, and the vertical line shows the best value of λ_1 determined during bootstrapping.

We can use this coefficient path to approximately 'rank' the variables in order of importance. In this case, an important variable would be one that lasts longer before being eliminated by the stronger regularization coefficient. That is, even with a strong L_1 regularizer, the variable stays intact, indicating it has reasonably strong correlations with the response. In our case, that makes our list (from strongest to weakest):

1. LSTAT: % lower status of the population
2. CRIM: per capita crime rate by town
3. TAX: full-value property-tax rate per $10,000
4. PTRATIO: pupil-teacher ratio by town
5. RM: average number of rooms per dwelling
6. B: $1000(Bk - 0.63)^2$ where Bk is the proportion of blacks by town

7. NOX: nitric oxides concentration (parts per 10 million)
8. CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
9. DIS: weighted distances to five Boston employment centres
10. AGE: proportion of owner-occupied units built prior to 1940
11. ZN: proportion of residential land zoned for lots over 25,000 sq.ft.
12. RAD: index of accessibility to radial highways
13. INDUS: proportion of non-retail business acres per town

This list seems to make intuitive sense: variables I might think are important (income, crime rate, tax rate) rank high, whereas more obscure variables (highway accessibility and non-retail businesses) rank low. The 'best' model we found previously seems to include all but the INDUS (industry) variable, which remains unimportant in every model.

This plot also gives us a way to select variables (if we ignore prediction error). Within our ranking, we actually have subgroups (created due to the grouping effect discussed below). For example, LSTAT and CRIM both zero our around the same time, followed by the group of TAX, PTRATIO, RM, and B. Thus, we could choose to select one variable from each group, or perhaps choose only the most important 2 or 3 groups (similar to principal components analysis).

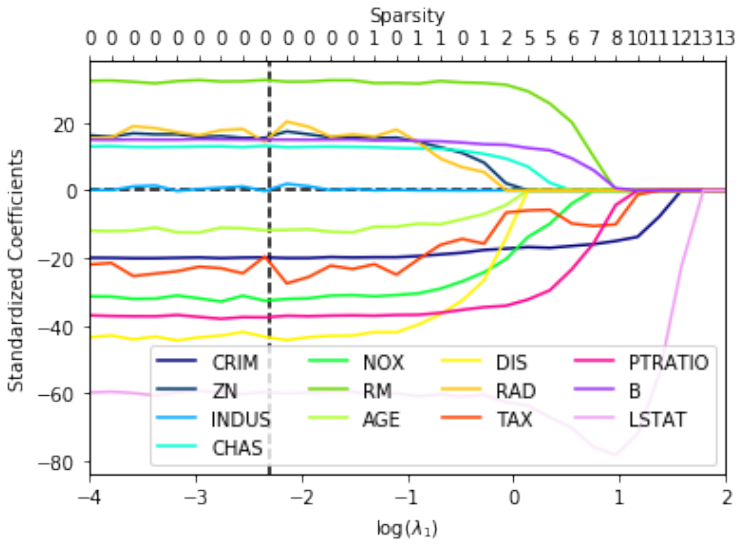


Figure 5: Coefficient paths for the optimal value of $\lambda_2 = 0.002$ determined during bootstrapping. On the top axis, the number of parameters that have been eliminated are recorded. The dotted vertical line shows the best value of λ_1.

If we increase λ_2, I would expect the groups to be more evident (again, see the grouping effect discussed below). We can produce a new coefficients plot, show in Fig. 6, where we see this grouping become more and more evident as we ramp up the value of λ_2. Many of the coefficient values are pushed towards each other, forming distinct groups. This can be a useful tool by itself to help find clusters of variables that act in the same manner.

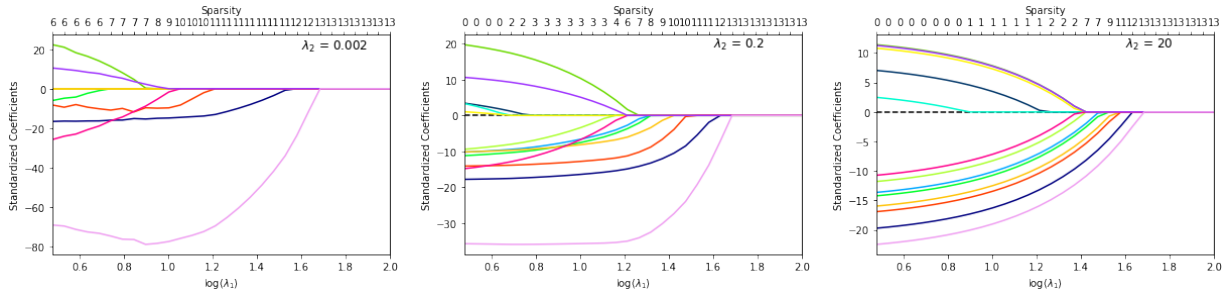


Figure 6: Coefficient paths for large values of λ_2. From left to right, these are 0.002, 0.2, and 20. On the top axis, the number of parameters that have been eliminated are recorded. The dotted vertical line shows the best value of λ_1.

All of the results we have discussed here depend on collinearity. Parameters that are correlated end up in the groups we have discussed. For example, we can look at the correlations of one of the groups that seemed to be present in our original coefficients plot: DIS, AGE, ZN, and RAD. Their correlations are shown in Fig. 7, where we do indeed see some correlations, although the multi-collinearity that might be underlying this group is harder to pick out by eye. This makes the elastic net grouping effect all the more valuable as an important variable selection tool.

Part F: Discussion

The Grouping Effect

The grouping effect arises out of the existence of multi-collinear data in your sample. If there are multiple parameters in your feature space that are highly correlated, the grouping effect will cause the resulting parameters β_i to be approximately equal. This seems to be a useful effect, as it helps us to identify collinear data, as well as keeping all terms that are actually relevant within a collinear group. It also helps to deal with the 'large p, small n' problem, in which the data matrix X is naturally singular as there is not enough data to explain all of the features. Grouping the features together (for example, by adding the ridge regression coefficient to the diagonals such that $X^T X$ is invertible) allows us to not only deal with small sample sizes, but gain insights into whether or not features are truly collinear.

This will show up in ridge regression, but not in lasso regression. Consider, for example, 2 highly correlated features, x_i and x_j in your data set, such that $x_i \sim x_j$. Without loss of

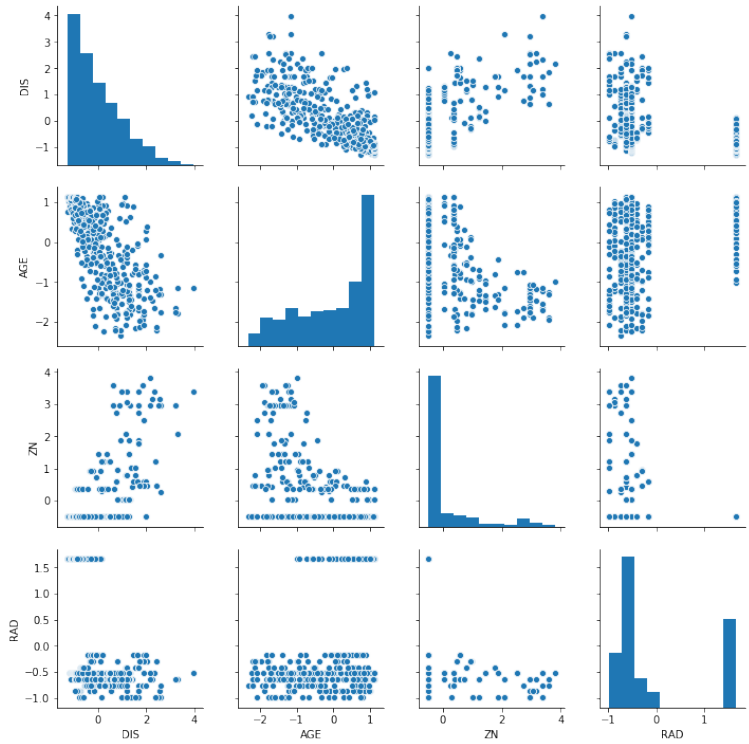


Figure 7: Correlations for the group of variables DIS, AGE, ZN, and RAD.

generality, assume they are positively correlated, if negative, simply use $x_j \to -x_j$. Now, consider the linear model that arises from a good fit. This will have terms of the form:

$$\hat{y} = X\beta = ... + x_i\beta_i + x_j\beta_j + ... \tag{45}$$

However, because of the collinearity of x_i and x_j, we can rewrite this as:

$$\hat{y} = X\beta = ... + x_i(\beta_i + \beta_j) + ... = ... + x_j(\beta_i + \beta_j) + ... \tag{46}$$

Which would also lead to approximately the same (non-penalized) cost function. Thus, let's consider the arbitrary solution where $\beta_i + \beta_j = \beta$. Then any solution of the form:

$$\hat{y} = X\beta = ... + x_i p\beta + x_j(1-p)\beta + ... \tag{47}$$

Will also be a valid (non-penalized) solution for the parameters. In the extreme case where $x_i = x_j$ exactly, then the non-penalized costs would be identical. Now consider the ridge regression penalty we would gain in these scenarios:

$$||\beta||_2^2 = \sum_k \beta_k^2 = ... + (p\beta)^2 + ((1-p)\beta)^2 + ... = ... + \beta^2(2p^2 - 2p + 1) + ... \tag{48}$$

This is minimized when we choose to set $p = 1/2$, which is the case where we choose $\beta_i = \beta_j$. This is exactly the effect of grouping, and we see that ridge regression will naturally group variables that are highly correlated.

We can contrast this with lasso regression, where grouping does not naturally occur. Again, we simply just need to look at the form of the penalty:

$$||\beta||_1 = \sum_k |\beta_k| = ... + |p\beta| + |(1-p)\beta| + ... \tag{49}$$

Here we consider 3 cases: $p > 1, 0 < 1 < p$, and $p < 0$. First, if $p > 1$, then $p > 0$ and $1 - p < 0$. Then we get:

$$|p\beta| + |(1-p)\beta| = p|\beta| - (p-1)|\beta| = |\beta| \tag{50}$$

If $0 < p < 1$, then both $p, 1 - p > 0$ and so we get:

$$|p\beta| + |(1-p)\beta| = p|\beta| + (1-p)|\beta| = |\beta| \tag{51}$$

Finally if $p < 0$, then $p < 0$ and $1 - p > 0$, and so we get:

$$|p\beta| + |(1-p)\beta| = -p|\beta| + (1-p)|\beta| = |\beta| \tag{52}$$

In all three cases, the resulting penalty is independent of our solution, and so all solutions are equally valid, with or without the lasso penalty. Thus, lasso regression does not naturally group variables, but rather it tends to arbitrarily choose one of the correlating features (typically the most correlated with y when they are not exactly collinear), and sets the rest to 0.

Elastic net gains the best of both worlds. By including lasso regression, elastic net is capable of finding sparse solutions that do not consist of every single feature in the initial data set. However, the ridge regression portion of the elastic net pushes these sparse solutions to at least group the collinear data to have similar coefficients. Thus, this form of regression acts as a big 'elastic' that shrinks the values to zero, but smoothly around groups of correlated features.

Cluster Elastic Net

In this method, the elastic net is taken a step further, where clusters of correlated features, with similar responses, are clustered together. This means that, if feature i and j are both in the same cluster, then:

$$x_i\beta_i \sim x_j\beta_j \tag{53}$$

Thus, if features within a cluster are highly correlated, then the parameters are driven toward each other. If the features are inversely correlated, then the parameters are driven toward

each other with a sign change. Finally, if the features are not overly correlated, then the model drives the feature parameters to zero. Thus you ultimately end up with clusters of features that have the same parameters that yield similar responses.

The clusters are partitioned by k-means on $x_i\beta_i$, and so they are non-overlapping clusters of features. This is beneficial as it allows features to choose clusters and shrink their parameters towards other features with similar responses, instead of being stuck between two clusters. Specifically, we benefit from this because it explicitly tells us which features lead to similar outcomes in the response. This can be useful in situations where we wish to learn about the types of features that lead to specific responses: this could be useful in, for example, learning important features when attempting to do classification problems (which features lead to class A? which to class B?).

In the context of ensemble learning, this method can be useful to help determine important features (which ones often get clustered together, how do they change the response, etc). It may also help to allow for non-overlapping clusters, as different subsets of data within the ensemble may lead to different clusters, allowing for the model to identify different correlations within the data.

Because CEN clusters features that have a similar response, it is useful when classes are imbalanced as you can focus your training on only those features that give rise to the response you are interested in. So for example, in our group project we worked on bus bunching, and whether a host of features could be used to predict whether or not buses are going to bunch up along a route. This clustered elastic net could be used to first reduce the variables to only those that typically have strong positive responses, and then use a classification method (logistic regression, LDA, SVMs, neural nets, etc) to predict when changes in these features lead to the desired response.