Lesson 5 - Feature Selection – Lasso Regression

There are two common reasons for want to choose a subset of available features

1. Efficiency
   If we have a very large number of features, then our calculations become burdensome. Even making a prediction, if the model contains billions of coefficients, would be difficult.
2. Interpretability
   We want only those features that contribute to predictability so that we can better interpret the results. If the model is littered with unimportant coefficients then we may not know why our model works and so we have a hard time interpreting results.

Greedy Solutions

In these solutions, we formulate a discrete number of potential feature sets and test each one, then choose the one that best reduces validation error (as measured using a validation set or cross validation or some other validation algorithm).

All Subsets Algorithm

In this algorithm, we are formulating all possible combinations of features and choosing the one that best reduces validation error.

Assuming D features;

- Start with zero features and calculate training error.
- For each of the D features, train on the single feature, then choose the feature with the lowest training error.
- For all possible feature pairs, calculate the training error for each pair, then choose the pair with the lowest training error.
- Continue in this fashion, add one feature at a time, training on all possible combinations of those d features, and then choosing the combination of d features with the lowest training error.
- Our last set will be all d features and there is only one of those, so we pick that set and calculate it's training error.

Two things to remember;

- The resulting features are not necessarily subsets of each other (except for the last set of D features). For example, in our house example the best single feature is square feet floor space and the best pair of features is number of bedrooms and number of bathrooms.

- The resulting d+1 features sets can be over-trained. We need to estimate the general error to choose the best set of features. We can do several ways;
    - If we have enough data, we can set aside a validation set to calculate validation on.
    - If we don't have enough data for a validation set, we can use some kind of K-fold Cross Validation.
    - There are other suitable validation algorithms beyond these two, but they are not discussed in this course.

Cost of Training all Possible Subsets

If we want to train all possible combinations of D features, this requires 2^D trainings (assuming the constant feature is included in D, if not it is D+1).

- If we have only the constant feature this is 2^1 = 2 trainings
- If we have 8 features, this is 2^8 = 256
- If we have 30 features, this is 2^30 ~ 1 billion trainings
- if we have 1000 features this is 2^1000 ~ 1x10^301
- It is not uncommon to have 1000 features or more.

So clearly we cannot train on every possible subset in many cases because it is computationally infeasible.

Rather than use an exhaustive search, we can use some kind of greedy algorithm that does not wait for all subsets in order to choose results.

Forward Stepwise Greedy Algorithm

Effectively, we are going to test a combination, choose the best, then keep this set of features in the next iteration where we add one more feature.

Assuming D features;

- Start with zero features and calculate training error.
- For each of the D features, train on the single feature, then choose the feature with the lowest training error. We will call this feature 1.
- For all possible feature pairs using Feature 1 and each one of the other features, calculate the training error for each pair, then choose the pair with the lowest training error. This pair will move to the next iteration and the other pairs will be 'dropped'.
- Continue in this fashion, add one feature to the previous iterations' best combinations of d features, training on all possible combinations of those d+1 features, and then choosing the combination of d+1 features with the lowest training error.
- Our last set will be all d features and there is only one of those, so we pick that set and calculate it's training error.

In this greedy algorithm, we keep the best combination at each iteration, then find out which one of the remaining features will produce the lowest training error when added to it.

Things to note

- The greedy solution starts with the same 'no features' sets as the All Subsets algorithm.
- From iteration to iteration the error never increases.
  Because we are only ever adding a single feature to another set, if that feature negatively affects the error, then the coefficient will be trained to zero, so not net error is added.
- The greedy solution eventually meets the All Subsets solution, when we are using all features in the final combination.

When do we stop the greedy procedure?

- NOT based on training error, since this would lead to overfitting
- NOT based on test error, since this will not give us a real sense of generalization error.
- We do it based on the validation error. So as before, we need either a validation set or we need to use cross-validation or we need some other validation algorithm that does not use our test set.

Complexity of Forward Stepwise

- We examine up to D levels of complexity (D steps)
- Within each, we train from 1 to D combinations during a step.
- **Total number of models examined is (D^2+D)/2**
- So the complexity is $O(D^2)$
- This is significantly less that the $O(2^D)$ that the All Subsets algorithm took for large D.

Other Greedy Algorithms

- Backward Stepwise
  Start with a full set and figure out which feature to remove.
- Forward Backward Stepwise
  Do a forward stepwise, but backup every once and a while and remove features that may have become redundant after other features were added.
- Many more.

Using L1 Regularization for Feature Selection

We've seen how we can use regularization in the form of Ridge Regression to avoid over-fitting. In Ridge Regression, our cost of fit includes the L2norm squared term and it's tuning parameter λ;

$$\text{Ridge Regression Cost of Fit} = RSS(w) + \lambda \|\vec{w}\|_2^{\,2}$$

The L2 regularization encourages small coefficients. However, it does not produce zero coefficients. While in the limit the coefficients are zero, any finite value of λ will allow for non-zero coefficients.

We already now that zero coefficients mean that the feature is not used in the model. Eliminating features improves efficiency by decreasing the computations required to use the model and it increases interpretability by eliminating features that to not contribute to the predictive power of the model. So sparse features are desirable.

Why not just use a threshold value and eliminate features whose |w| is less than the threshold? The problem here is that it is common that features are statistically correlated – they are related is some numerically predictable way. For instance, the number of bathrooms and the number of showers. Furthermore, Ridge Regression will distribute the coefficient magnitude across these correlated features. So number of bathrooms and number of showers might have the same coefficient, but these are half of what it would be if only one of those features were included in the model. This means that if we eliminate coefficients based on a threshold then we may be eliminating a bunch of correlated features because their coefficients are below the threshold, but in doing so we've removed important predictive power from the model.

Lasso Regression uses regularization to drive non-important features all the way to zero, the remaining features are the selected features. Instead of using the L2norm, Lasso regression uses the L1norm (the sum of the absolute value of each coefficient) and a tuning parameter;

$$\text{Lasso Regression Cost of Fit} = RSS(w) + \lambda \|w\|_1$$

The L1norm is $\|w\|_1 = |w_0| + |w_1| + \cdots + |w_D|$

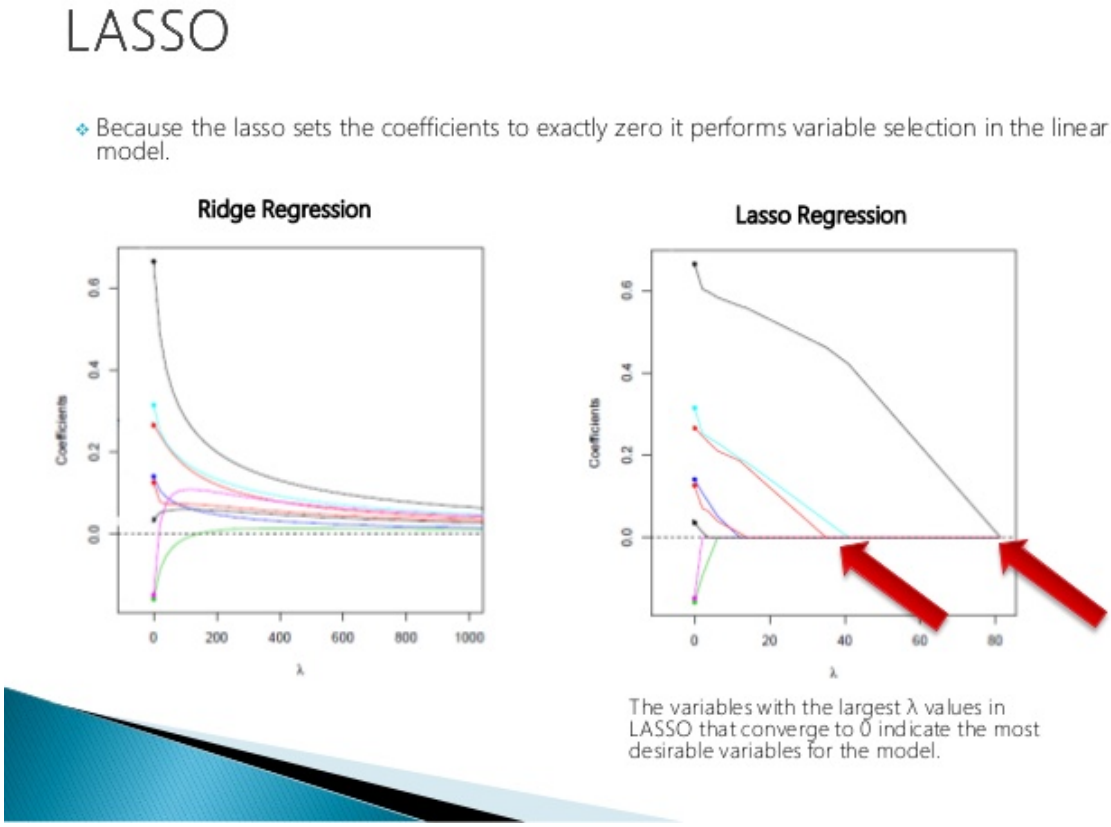
As with Ridge Regression, we search over a continuous range of possible λ values. This is in contrast to the greedy algorithms that used a discrete number of possible solutions.

In Lasso Regression, the tuning parameter governs scarcity of features vs fit to our training data. It does this because it defines a 'pointy' shape along each axis

– in the case of 2 coefficients it is a diamond with points along each axis. Because of the pointy shape, it is more likely to intersect with the Least Squares solution (which is an ellipsoid) at one of the pointy parts, which is on an axis. Since is it on an axis, it then has minimized the value that is along that axis to zero.

The videos do an excellent job of show the how this works for both the ridge objective (L2norm squared) and the Lasso objective (L1norm).

Below is an illustration comparing the coefficient paths for Ridge Regression and Lasso Regression taken from this presentation by Derek Kane. It shows how the values of the coefficients vary as the tuning parameter λ is varied from zero to positive values. You can see that rather than all coefficients being reduced, individual coefficients tend to go to zero as the tuning parameter (and so the effect of the L1norm penalty) are increased. This results in sparse coefficients.



Here is an illustration of the L1norm and Least Squares contour plot taken from this blog post by xbinworld (it is in simplified Chinese, so if you want to read the post, use Google Chrome and translate it.) The idea is that the ellipsis of the least squares solution is most likely to intersect with the diamond of the L1norm solution at an axis. When this happens, the coefficient associated with that axis

has gone to zero. (I encourage you to view the lesson videos, they discuss these ideas very clearly).

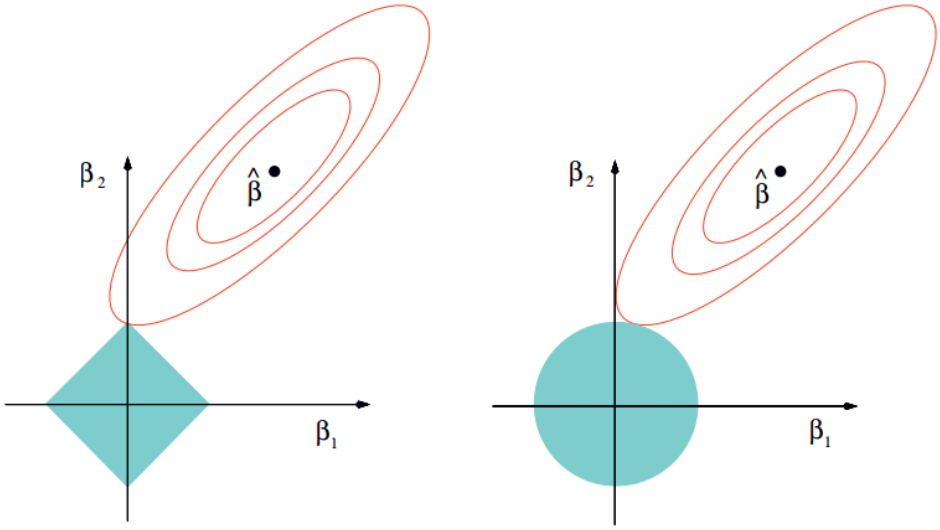


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are the contours of the least squares error function.*

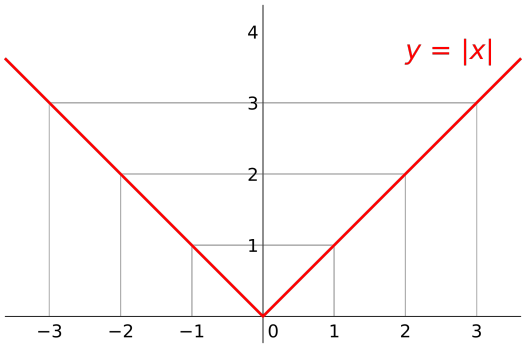
The Lasso Solution

In Least Squares and Ridge Regression, we used two techniques to solve

- Closed Form Solution
  Here we took the derivative (gradient) of the cost function, set it to zero and solved for w.
- Gradient Descent
  Here we used an iterative algorithm to find the minimum by testing solutions, incrementing based on the slope at the solution and stopping when the slope was within some tolerance of zero.

Closed Form Lasso Solution

For the Lasso solution, in addition to RSS(w), which we know how to solve for, we have the L1norm term, $\|w\|_1$. This is the sum of the absolution values of each coefficient. See the plot of absolute value taken from Wikipedia;



Note the shape; looks like part of a diamond! The derivatives along the lines are -1 and 1 (the slope of the line). But at the axis, where it goes to zero, there is no derivative (there is no continuous rate of change). So there is no way to take the derivative there, so there is no closed form solution based on the gradient (although there is another way to do this with a concept known as subgradients).

Even if we could compute the derivative, there is no actual closed form solution.

Gradient Descent

As we've seen, there is no gradient solution for the absolute value function, so simply retooling our gradient descent algorithm won't work either. Another way to think about this is that we depended on the slope gradually going to zero in order to find the minimum. In the case of the L1norm, the slope will stay a constant (with 1 or -1, then the gradient will become instantly incalculable (at the zero point). So our gradient descent won't work (but subgradients could).

Coordinate Descent

An alternative algorithm, which is generally applicable beyond our Lasso object, is Coordinate Descent. The goal, like gradient descent, is to minimize some function g;

$$\min_{w} g(w)$$

where

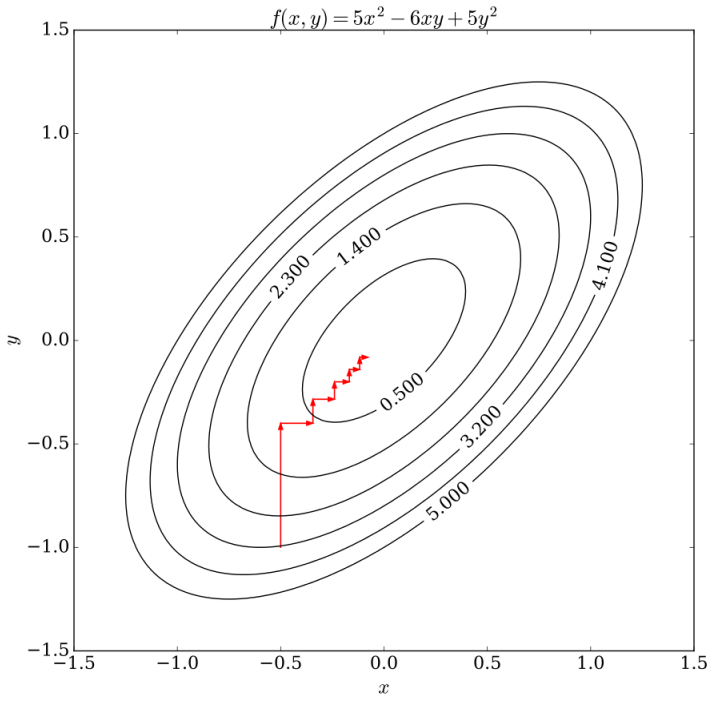
$$g(w) = g(w_0, w_0, \ldots, w_D)$$

Minimizing this function of multiple coordinates is often difficult. However, it is generally easy to minimize each coordinate if we keep the others fixed; it becomes a one dimensional optimization problem.

General Coordinate Descent

- Initialize $\widehat{w}$, commonly to zeroes, but we could do this more intelligently.
- while not converged
    - Pick a coordinate, j and minimize the coordinate value
        - $\widehat{w}_j \leftarrow \min_{\omega} g(\widehat{w}_0, \ldots, \widehat{w}_{j-1}, \omega, \widehat{w}_{j+1}, \ldots, \widehat{w}_D)$
          Note the ω is omega (not some other kind of w). Also the other coordinates are the values from previous iterations (which may be the initial value if that coordinate has not yet been picked to be minimized).

In each step we do some minimization on one coordinate, but typically that coordinate will need to be visited many times before it and the the overall function are minimized. In other words, each coordinate will be visited multiple times and each time it will be in the context of a new overall set of coordinates that is closer to the minimum. The result is a series of axis aligned changes to $g(\widehat{w})$ as the solution moves closer and closer to overall minimization. This Wikipedia article discusses this and provides this illustration of the convergence process. Importantly, we can see the steps become smaller and smaller as the algorithm approaches the solution;



Axis Aligned convergence in Coordinate Descent

Choosing the coordinate to minimize

- We can pick it randomly. This is called Random or Stochastic Coordinate Descent.
- We can simply go round robin in order. Once we get done doing a minimization step on the last coefficient, we go back to the first coefficient and continue until the overall function is minimized.

In contrast to Gradient Descent, there is no step size to choose. In Gradient Descent, if the step size was too large and the tolerance too small, the result may not converge.

This is a very useful approach for many problems

- It converges to the optimum for strongly convex functions.
- It converges on the Lasso function, which is important to this discussion.

Normalizing Features

Things become easier if we normalize each feature column. Each column is a vector of readings for that feature. But different columns have different ranges and different units. We would like to have all of these values in the same range, 0..1. To do this, we normalize the column vectors; For feature j ;

$$\underline{h}_j(x_k) = \frac{h_j(x_k)}{\sqrt{\sum_{i=1}^{N} h_j(x_i)^2}} = \frac{h_j(x_k)}{z_j}$$

$\underline{h}_j(x_k)$ is the normalized value for reading k of feature j (column j, row k in $\underline{H}$, the matrix of normalized features). The underbar indicates a normalized term.

$h_j(x_k)$ is the original (non-normalized) value for reading k of feature j (column j, row k in the original feature matrix H)

$\sqrt{\sum_{i=1}^{N} h_j(x_i)^2}$ is the vector magnitude of the jth column of the feature matrix H.

z_j is the normalizer for the jth feature (the scaling factor for the jth feature).

We must scale the test data in the exact same way we scale the training data. This means that we must apply the same normalizer to each column in the test data that we apply to the corresponding column of the training data. (This would imply that we want to calculate the normalizer on all the data before it is split into training, validation and test, so we can guarantee each value will fall into the range 0..1, which is important to our assumptions regarding simplification of the expressions).

Coordinate Descent for Least Squares

Remember that the Least Squares objective is to minimize the sum of the squared residuals between the known and predicted values.

$$RSS(w) = \sum_{i=1}^{N} \left(y_i - \sum_{j=0}^{D} \underline{h}_j(x_i) w_j \right)^2$$

The objective is to minimize this. We do this by finding where the gradient goes to zero (we find the bottom of the convex curve that is RSS(w)).

In the **Coordinate Descent** algorithm, we **minimize the $\nabla RSS(w)$ one coordinate at a time;** we choose one coordinate **and hold the other coordinates (w_{-j}, which means all coordinates != j) constant** while we descend on that chosen coordinate. Mathematically, we **take the partial derivative of RSS(w) with respect to the chosen coordinate (j) *on the normalized features***.

$\frac{\partial}{\partial w_j} RSS(w)$ $= -2 \sum_{i=1}^{N} \underline{h}_j(x_i) \left[y_i - \sum_{j=0}^{D} \underline{h}_j(x_i) w_j \right]$

y_i	the ith known output value.
$\underline{h}_j(x_i)$	the jth normalized feature for the ith reading; the jth column in the ith row of normalized feature matrix.
$\underline{h}_j(x_i) w_j$	the prediction for the jth coordinate of the ith normalized feature
$\sum_{j=0}^{D} \underline{h}_j(x_i) w_j$	the ith predicted output value (summation form).
$y_i - \sum_{j=0}^{D} \underline{h}_j(x_i) w_j$	the residual between the ith known output value the ith predicted output

Separating out the constant features;

$\frac{\partial}{\partial w_j} RSS(w)$ $= -2 \sum_{i=1}^{N} \underline{h}_j(x_i) \left[y_i - \sum_{j=0}^{D} \underline{h}_j(x_i) w_j \right]$

separate jth term from the summation

$= -2 \sum_{i=1}^{N} \underline{h}_j(x_i) \left[y_i - \sum_{j=0,k\neq j}^{D} \left(\underline{h}_k(x_i) w_k \right) - \underline{h}_j(x_i) w_j \right]$

multiply the $h_j(x_i)$ to each term

$= -2 \sum_{i=1}^{N} \underline{h}_j(x_i) \left(y_i - \sum_{j=0,k\neq j}^{D} \left(\underline{h}_k(x_i) w_k \right) \right) - w_j \left[\underline{h}_j(x_i)^2 \right]$

separate the terms summed over N

$= -2 \left(\sum_{i=1}^{N} \underline{h}_j(x_i) \left(y_i - \sum_{j=0,k\neq j}^{D} \left(\underline{h}_k(x_i) w_k \right) \right) + \sum_{i=1}^{N} w_j \left[\underline{h}_j(x_i)^2 \right] \right)$

multiply the -2 factor and pull out the constant w_j

$$= -2\sum_{i=1}^{N} \underline{h}_j(x_i)\left(y_i - \sum_{j=0,k\neq j}^{D}\left(\underline{h}_k(x_i)w_k\right)\right) + 2w_j \sum_{i=1}^{N}\left[\underline{h}_j(x_i)^2\right]$$

Things to note at this point;

$\sum_{i=1}^{N}\left[\underline{h}_j(x_i)^2\right]$	this is the sum of the squared values for the jth normalized feature – the magnitude of the normalized column vector. The magnitude of a normalized vector is 1, so this simplifies to 1.
$\sum_{j=0,k\neq j}^{D}\left(\underline{h}_k(x_i)w_k\right)$	this is the ith predicted value for all the features except the jth feature.
$y_i - \sum_{j=0,k\neq j}^{D}\left(\underline{h}_k(x_i)w_k\right)$	this is the ith residual if we exclude the jth feature.

For simplicity, we will call this summation ρ (rho);

$$\rho_j = \sum_{i=1}^{N} \underline{h}_j(x_i)\left(y_i - \sum_{j=0,k\neq j}^{D}\left(\underline{h}_k(x_i)w_k\right)\right)$$

So using the ρ_j term and understanding the magnitude of the normalized feature values is 1, we end up with;

$$\frac{\partial}{\partial w_j} RSS(w) \quad = -2\rho_j + 2w_j$$

To minimize this, we set it to zero and solve for w_j, which then is our estimated coordinate;

$$\frac{\partial}{\partial w_j} RSS(w) \quad = -2\rho_j + 2\widehat{w}_j = 0$$

$$2\widehat{w}_j = 2\rho_j$$

$$\boldsymbol{\widehat{w}_j = \rho_j}$$

$$\widehat{w}_j = \sum_{i=1}^{N} \underline{h}_j(x_i)\left(y_i - \sum_{j=0,k\neq j}^{D}\left(\underline{h}_k(x_i)w_k\right)\right)$$

$$\widehat{w}_j = \sum_{i=1}^{N} \underline{h}_j(x_i)\left(y_i - \hat{y}_{k\neq j}\right)$$

So the jth coordinate is simply the jth rho term.

$\left(y_i - \hat{y}_{k\neq j}\right)$	this is the ith residual without the jth feature.

So, if the jth feature does not matter, then it's contribution is close to zero and the predicted value without it will be a good prediction, therefore residual will be close to zero. That will make w_j small.

If the jth feature does matter, then the predicted value without it will be a poor prediction the residual will be large. Theoretically then, the difference between the actual and predicted value will be exactly what the jth term should contribute. A larger residual indicates that the jth feature is important and it will make for a larger w_j.

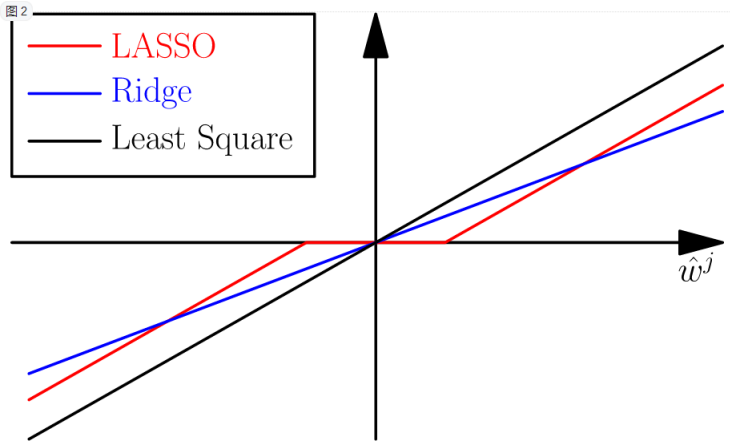
```
Least Squares Coordinate Descent with Normalized Features

1. Initialize ŵ to zeros (or some smart choice)
2. While not converged
   - For j in 0..D
     - ρ_j = Σ_{i=1}^{N} h_j(x_i)(y_i − ŷ_{k≠j})
     - ŵ_j = ρ_j
```

Lasso Regression Coordinate Descent

Here we add in the L1norm and the tuning parameter using soft thresholding.

$$\hat{w}_j = \begin{cases} \rho_j + \frac{\lambda}{2}, if\ \rho_j < -\frac{\lambda}{2} \\ 0, if\ -\frac{\lambda}{2} \leq \rho_j \leq \frac{\lambda}{2} \\ \rho_j - \frac{\lambda}{2}, if\ \rho_j > \frac{\lambda}{2} \end{cases}$$

This blog has a good illustration of how least squares, ridge regression and lasso regression solutions compare;



Here the red line is the lasso line. As it goes from left to right, the point at which is goes to zero is $(-\frac{\lambda}{2}, 0)$ and the point at which it begins to go positive is $(\frac{\lambda}{2}, 0)$.

Lasso Coordinate Descent with Normalized Features

1. Initialize $\hat{w}$ to zeros (or some smart choice)
2. While not converged
   - For j in 0..D
     - $\rho_j = \sum_{i=1}^{N} \underline{h}_j(x_i)\left(y_i - \hat{y}_i(\hat{w}_{k\neq j})\right)$

     # by convention, don't regularize intercept

     - $\hat{w}_j \leftarrow if\ (j = 0)\ then\ \rho_j\ else \begin{cases} \rho_j + \frac{\lambda}{2}, if\ \rho_j < -\frac{\lambda}{2} \\ 0, if\ -\frac{\lambda}{2} \leq \rho_j \leq \frac{\lambda}{2} \\ \rho_j - \frac{\lambda}{2}, if\ \rho_j > \frac{\lambda}{2} \end{cases}$

Convergence

For convex curves, the amount we change in each iteration will become smaller and smaller. We can use this fact to terminate the iteration when the amount we change from the prior iteration falls below a threshold. In fact, we want to make sure that the steps we make for all coordinates are below the threshold before we stop.

$$\text{converged} = \text{all } |w_j^t - w_j^{t-1}| < \text{threshold}$$

Unnormalized Features

Coordinate descent can be applied to unnormalized features simply by calculating the normalized as part of the process and applying it as we go;

Lasso Coordinate Descent with Unnormalized Features

1. Computer Normalizer
   $z_j = \sum_{i=1}^{N} h_j(x_i)^2$
2. Initialize $\hat{w}$ to zeros (or some smart choice)
3. While not converged
   - For j in 0..D
     - $\rho_j = \sum_{i=1}^{N} h_j(x_i)\left(y_i - \hat{y}_i\left(\hat{w}_{k\neq j}\right)\right)$
     - $\hat{w}_j = \begin{cases} \frac{\left(\rho_j + \frac{\lambda}{2}\right)}{z_j}, if\ \rho_j < -\frac{\lambda}{2} \\ 0, if\ -\frac{\lambda}{2} \leq \rho_j \leq \frac{\lambda}{2} \\ \frac{\left(\rho_j - \frac{\lambda}{2}\right)}{z_j}, if\ \rho_j > \frac{\lambda}{2} \end{cases}$

Choosing the Lasso tuning parameter

This can be done as it was for Ridge Regression;

- Fit $\hat{w}_\lambda$ on the training set
- Test the performance of various $\hat{w}_\lambda$ to select the best λ using some form of validation that does not involve the test set
  - Use a Validation set to pick the tuning parameter
  - Use Cross-Validation to pick the tuning parameter
- Assess the generalization error of the chosen $\hat{w}_\lambda$ using the test set.

Note that using validation to choose λ may lead to less sparse models. This is because validation will choose the model with the best predictive value and does not attempt to minimize the number of features while doing that, so it tends to choose smaller λ. It may be that a very good model exists with less features (larger λ), but the validation step will not find it. There are other ways to choose λ for Lasso Regression that take into account the sparseness of the resulting features.

Debiasing Lasso

Lasso shrinks coefficients relative to the Least Squares solution, which means a higher bias model with less variance. The bias in this model can be reduced by running the Least Squares on only the features with non-zero coefficients as calculated by Lasso. The result in a lower bias model that often has significantly

small MSE (Mean Squared Error) than either the straight Lasso result or the straight Least Squares result.

1. Run Lasso Regression to select features (those with non-zero coefficients)
2. Run Least Squares Regression on only the selected features to produce the final model.

This works because only the 'relevant' features are used in the Least Squares model, so 'irrelevant' features are not added that reduce the effect of relevant features.

Issues with Lasso Regression

- Lasso will tend to select amongst highly correlated features somewhat arbitrarily in that small changes to the data results in different features being chosen. This choice can be undesireable, we generally want to keep the all the correlated features.
- Often a model created with Ridge Regression has better predictive performance than a Lasso solution, but more features.

The Elastic Net algorithm (Zou & Hastie 2005) attempts to address these issues by combining an L2norm penalty with an L1norm penalty.

Other Lasso Solvers
2004..2008

- LARS – least angle regression

2008..2011

- Coordinate Descent

2011..now

- Parallel Coordinate Descent
- Parallel Stochastic Gradient Descent
- Parallel Independent Solutions with Averaging (Zhang et al 2012)
- ADMM – Alternating Directions Method of Multipliers