Regression Week 4 – Ridge Regression (L_2 Regularization)

Can we detect when a model is overfit?
- Overfit models tend to have coefficients that are very, very large.
- Models with many features which are very flexible, can easily be overfit. So models where D is large:

$$y_i = \sum_{j=1}^{D} w_j h_j(x_i) + \varepsilon_i$$

i	The index of the observation and features in the dataset (ith row of H)
j	The index of the term in the regression function
y_i	The ith observation (known output value) in the dataset
w_j	The jth coefficient in the regression function
$h_j(x_i)$	The jth function in the regression as a function of the ith row of features.
$w_j h_j(x_i)$	This product of the jth coeffient and jth function in the regression is the predicted value for the ith row of features. This is also know simply as $\hat{y}_i$.
ε_i	The error term for the ith regression prediction.

How does the number of points in our data set affect over-fitting?
- If we have only a small number of points, then models become rapidly over-fitted as model complexity increases. So a low ratio of data to complexity can result in an over-fit model. (Basically, the model function can vary wildly between our data points – there aren't enough points to constrain it. With a dense set of data, the function must conform to many more points, so will not vary wildly.
- Another way to think about it – as we approach all the possible points of data in the world, then our model will have to match a truer subset of the world.

Modifying the Cost-of-fit metric to address over-fitting
- How well our function fits the data
  Our prior quality metric was simply the residual sum of squares; how close our predictions match the training data.
- Magnitude of Coefficients
  Now we also want to add a term that takes into account model complexity, so we can bias quality towards simpler models. In particular we have noted that over-fitted models tend to have large coefficients.

Total Cost = measure of fit + measure of coefficient magnitude

So if both terms are small, then our cost is low. If either term is high, our cost is high.

Measure of fit is residual sum of squares;

$$RSS(w) = \sum_{i=1}^{N} (y_i - h(\vec{x_i})^T w)^2 = \sum_{i=1}^{N} \left(y_i - \hat{y}_i(w)\right)^2$$

Possible measures of the magnitude of the coefficients:
- a simple sum does not work, because we can get large positive and negative coefficients that sum to something close to zero.

- We can try the sum of the absolute value of the coefficients;

$$|w_0| + |w_1| + |w_{20}| + \cdots |w_D| = \sum_{j=0}^{D} |w_j| \triangleq \|\vec{w}\|_1$$

  This is defined as the L_1 norm of the vector. This is a good measure of the magnitude of the coefficients and is used in Lasso Regression.
- We can also look at the sum of squares of the coefficients;

$$w_0{}^2 + w_1{}^2 + w_2{}^2 + \cdots w_d{}^2 = \sum_{J=0}^{D} w_j{}^2 \triangleq \|\vec{w}\|_2{}^2$$

  This is defined as the L_2 norm squared. This is what is used in Ridge Regression.

So for Ridge Regression, we use this cost of fit;

$$\text{Total Cost} = RSS(w) + \|\vec{w}\|_2{}^2$$

To adjust the balance between RSS and the L2norm terms in the cost-of-fit we add a tuning parameter λ:

$$\text{Total Cost} = RSS(w) + \lambda\|\vec{w}\|_2{}^2$$

What happens as the tuning parameter changes:

- **When $\boldsymbol{\lambda = 0}$, then we are left with just RSS(w)**
- **When $\boldsymbol{\lambda = \infty}$,** then we have two cases based on w:
  - When $\widehat{w} \neq 0$, then result is infinity.
  - **When $\widehat{\boldsymbol{w}} = \mathbf{0}$, then result is RSS(0), which means all coefficients are zero (essentially just leaving the noise term in our model).**

When $0 < \lambda < \infty$, then $0 \leq \|\widehat{w}\|_2{}^2 \leq \|\widehat{w}^{ls}\|_2{}^2$, where $\|\widehat{w}^{ls}\|_2{}^2$ is the L2norm squared of the model only using RSS(w) (also called the least squares model, hence the ls).

In terms of the bias-variance trade-off

When $\boldsymbol{\lambda}$ is large:

- **The cost of complexity (large coefficients) is large, so we tend towards smaller coefficients**. In the limit, all coefficients are zero, when lambda is infinity, w goes to zero). So **large $\boldsymbol{\lambda}$ results in high bias/low variance model**. (when all coefficients are zero, the variance is zero because the model does not change no matter what data you give it).

When $\boldsymbol{\lambda}$ is small:

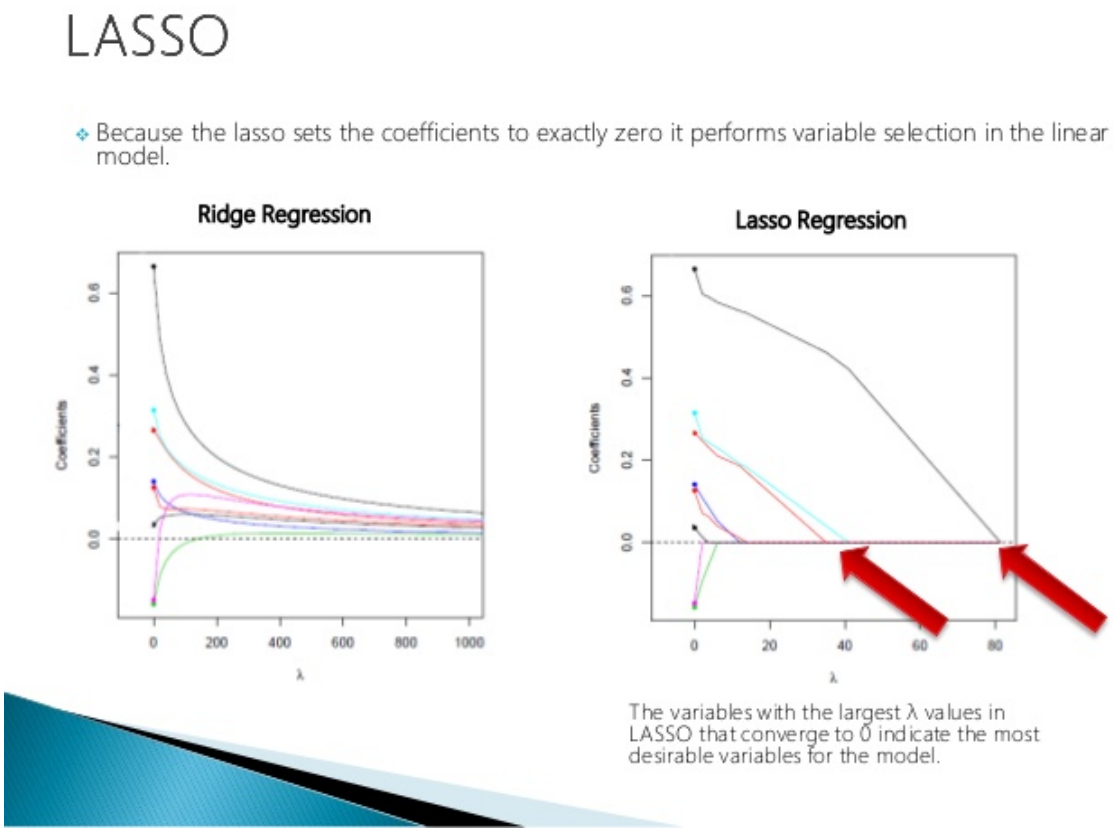
- The cost of complexity becomes small, we tend to allow larger coefficients. In the limit, **the cost tends towards RSS(w), so it tends towards the least squares fit**.

So we use λ to control model complexity and to the bias/variance trade-off.

NOTE: At the boundaries of our data it we generally have very few points and so it is difficult to control the fit in these areas; this results in boundary effects.

For each prospective λ value, we will get a different set of estimated coefficients, $\widehat{w}$. We can **plot the coefficients** (normalized, so all coefficients have similar 'units') **against $\boldsymbol{\lambda}$**. This is called the **coefficient path**.

Below is a slide taken from this presentation by Derek Kane that compares the coefficient paths for Ridge Regression (L2norm squared cost) and Lasso Regression (L1norm code). For now, focus on Ridge Regression and notice that all coefficients are reduced as the tuning parameter gets larger, but none go to zero.



Fitting the Ridge Regression Model

This Is the ML Algorithm box in the Machine Learning workflow.

Rewrite the total cost in matrix notation.
Remember the least squares model is for N observations in matrix form is:

$$\vec{y} = H\vec{w} + \vec{\varepsilon}$$

So the RSS(w) for the least squares model is;

$$RSS(w) = \sum_{i=1}^{N} (y_i - h(\vec{x_i})^T w)^2 = (y_i - Hw)^2 = (y_i - Hw)^T (y_i - Hw)$$

Now we want to include the L2 norm squared factor and λ, so Total cost is

$$RSS(w) + \lambda \|\vec{w}\|_2{}^2 = (y_i - Hw)^T (y_i - Hw) + \lambda \|\vec{w}\|_2{}^2$$

We can write the **L2 norm squared in vector notation** as

$$\|\vec{w}\|_2{}^2 = \vec{w}^T \vec{w}$$

or just

$$\boldsymbol{w^T w}$$

So the **Total cost is then**;

$$RSS(w) + \lambda\|\vec{w}\|_2^{\,2} = (\boldsymbol{y_i} - \boldsymbol{Hw})^{\boldsymbol{T}}(\boldsymbol{y_i} - \boldsymbol{Hw}) + \boldsymbol{\lambda w^T w}$$

Taking the **Gradient of the ridge regression cost**;

$$\nabla\left[RSS(w) + \lambda\|\vec{w}\|_2^{\,2}\right] = \nabla[(y_i - Hw)^T(y_i - Hw) + \lambda w^T w]$$

$$= \nabla[(y_i - Hw)^T(y_i - Hw)] + \lambda\nabla[w^T w]$$

$$= -\boldsymbol{2H^T(y - Hw)} + \boldsymbol{\lambda 2w}$$

Gradient of Ridge Regression Cost

$$\nabla\left[RSS(w) + \lambda\|\vec{w}\|_2^{\,2}\right] = -\boldsymbol{2H^T(y - Hw)} + \boldsymbol{\lambda 2w}$$

w	is the vector of D coefficients
H	is the matrix of N rows of observations by D columns of features
y	is the vector of N known output values
λ	is the tuning parameter for the L2norm squared term

Aside: The identity matrix of a given square dimension D is I_D. Here are some Identity matrix facts;

- Iv = v
- IA = A
- AI = A
- $A^{-1}A = I$
- $AA^{-1} = I$

Approach 1 – Closed Form Solution

Closed Form solution for the ridge regression gradient, solve for $\widehat{w}$

$$\begin{aligned}
0 &= -2H^T(y - H\widehat{w}) + \lambda 2\widehat{w} \\
0 &= -H^T(y - H\widehat{w}) + \lambda\widehat{w} \\
0 &= -H^T(y - H\widehat{w}) + \lambda I\widehat{w} \\
0 &= -H^T y + H^T H\widehat{w} + \lambda I\widehat{w} \\
H^T y &= H^T H\widehat{w} + \lambda I\widehat{w} \\
H^T y &= \widehat{w}(H^T H + \lambda I) \\
(H^T H + \lambda I)^{-1} H^T y &= \widehat{w}
\end{aligned}$$

So the **closed form solution for the ridge regression gradient** is:

Closed Form for Ridge Regression Gradient

$$\widehat{w}^{ridge} = (H^T H + \lambda I)^{-1} H^T y$$

$\widehat{w}$	is the vector of D predicted coefficients
H	is the matrix of N rows of observations by D columns of features
y	is the vector of N known output values
λ	is the tuning parameter for the L2norm squared term
I	is a DxD matrix (where D is the number of features and so coefficients)

The close form solution agrees with out previous thinking about how changing the tuning parameter lambda affects the resulting coefficients;

If $\lambda = 0$ then $\widehat{w}^{ridge} = (H^T H)^{-1} H^T y$, which is the closed for solution for least squares regression, so very low lambda tends towards the least squares solution (which can be overfit and so have very large coefficients).

If $\lambda = \infty$ then $\widehat{w}^{ridge} = 0$ because we are dividing by infinity. So very high lambda tends towards very small coefficients.

The problem with this approach is the same problem we have with the general closed form solution for least squares regression, the complexity of inverting the *($H^T H$)* term, which is a DxD matrix is $O(D^3)$, or on the order of the number of features cubed. This makes it too expensive in practice.

Approach 2 - Gradient Descent for Ridge Regression

Our element-wise gradient descent algorithm is based on the gradient of our cost function. For ridge regression the cost function has two parts; the least squares term and the L2norm squared term that includes the tuning parameter lambda. The resulting gradient is;

$$\nabla\left[RSS(w) + \lambda\|\vec{w}\|_2^{\,2}\right] = -\boldsymbol{2H^T(y - Hw) + \lambda 2w}$$

The $-\boldsymbol{2H^T(y - Hw)}$ term is the least squares term.
The $\boldsymbol{\lambda 2 w}$ term is the L2norm squared term

For the least squares gradient descent, our update of the coeffients looked like this;

$$\widehat{w}_j^{(t+1)} \leftarrow \widehat{w}_j^{(t)} - \eta\left[\sum_{i=0}^{N} -2h_j(x_i)\left(y - \hat{y}\left(w^{(t)}\right)\right)\right]$$

For ridge regression, we include the L2norm squared term with the tuning parameter;

$$\widehat{w}_j^{(t+1)} \leftarrow \widehat{w}_j^{(t)} - \eta\left[\sum_{i=0}^{N} -2h_j(x_i)\left(y - \hat{y}\left(w^{(t)}\right)\right) + 2\lambda\widehat{w}_j^{(t)}\right]$$

Which can be rearranged as

$$\widehat{w}_j^{(t+1)} \leftarrow (1 - 2\eta\lambda)\widehat{w}_j^{(t)} + 2\eta\left[\sum_{i=0}^{N} h_j(x_i)\left(y - \hat{y}\left(w^{(t)}\right)\right)\right]$$

This is an important result. The L2norm squared term and it's tuning parameter can be applied as a scaling factor to the predicted coefficient, the rest of the function is exactly the same as our least squares function.

If $\lambda = 0$ then the scaling factor becomes 1 and we have exactly the least squares method.

When we plug this into our incremental, element-wise gradient descent, the calculation becomes;

Ridge Regression Gradient Descent Algorithm

- Choose initial values for the coefficients $w^{(1)}$. We can set them to zero or we could be smarter if we wanted to converge faster.
- Choose a tolerance ε. Choose step size η. Choose tuning parameter λ.
- t = 1
- while $\left\|\nabla RSS\left(\widehat{w}^{(t)}\right)\right\| > \varepsilon$
  - $\hat{y}^{(t)} \leftarrow Hw^{(t)}$ # predictions with current w
  - $residuals \leftarrow y - \hat{y}^{(t)}$ # residuals with current w
  - $sumOfSquares \leftarrow 0$ # we calc this incrementally
  - For j = 0 to D-1 # for each feature j
    - $partial(j) \leftarrow 2h_j(x_i) \cdot residuals$ # contribution of j to gradient
    - $w_j^{(t+1)} \leftarrow (1 - 2\eta\lambda)w_j^{(t)} - \eta \times partial(j)$ # estimate jth coefficient
    - $sumOfSquares += partial(j) \cdot partial(j)$ # contribution to sum of squares
  - $\left\|\nabla RSS\left(\widehat{w}^{(t)}\right)\right\| \leftarrow \sqrt{sumOfSquares}$ # RSS(w) for iteration t
  - t = t + 1
- return $w^{(t)}$ # return the last estimated w

D	is the number of features and coefficients
N	is the number of rows in the dataset
t	is the current iteration
j	is the index of the feature in the feature matrix AND the index of the feature coefficient the regression function.
H	is the feature matrix of N rows of observations by D columns of features
$h_j(x_i)$	is the jth feature vector (the jth column of feature matrix). It has length of N.
y	is the vector of N known output values
$\hat{y}^{(t)}$	is the vector of N predicted output values at iteration t. Sometimes written $\hat{y}\left(w^{(t)}\right)$.
$w^{(t)}$	is the vector of D estimated coefficients at iteration t
$w_j^{(t)}$	is the estimated coefficient for jth feature at iteration t
λ	is the tuning parameter for the L2norm squared term
$(1 - 2\eta\lambda)$	is the scaling fractor for the L2norm squared term and it's tuning parameter lambda. Note that this is a constant that can be calculated outside the loop.
$\left(y - \hat{y}^{(t)}\right)$	is the residual (actual – prediction) at iteration t

NOTE: partial derivative for feature j is twice the dot-product of the feature_vector(j) and the residuals vector for iteration t:

$$partial(j) \leftarrow 2\sum_{i=0}^{N} h_j(x_i)\left(y - \hat{y}^{(t)}\right)$$

We can calculate the residual outside of the feature loop for efficiency.

Assessing the Performance of our choice of the tuning parameter

We can use a validation set if we have sufficient data to create a validation set as well as the training set and the test set. We can train at a given value of the tuning parameter λ and then calculate the error against the validation set. We do this for all values of the tuning parameter that we wish to consider, and then pick the tuning parameter with the lowest validation error.

If we have a small-ish data set, then we may not have enough data to have a validation set and a test set that are large enough to be generalizable. If this is true then we need to do something different. Remember that over-fitting happens when we fit to a single data set to closely, in which case our resulting coefficients don't produce good results on the general body of data. This is more likely if our training set is small.

So rather than have one training set and one validation set, which both may be too small to be generalizable, we can use all the data in various combinations to produce multiple training/validation set pairs. For each set in turn, we train on the training set and test on the validation set. We then average the errors across all validation sets to get our cross-validation error.

K-fold Cross-Validation

We segment the data (N rows) into K segments randomly. (NOTE: we don't simply group the first K rows, then the second set of K rows, etc – the assignment to segments must be random). Each segment in turn will be used as the validation set and the other K-1 segments will be used as training data.

Note: see how improves our chances of a generalizable result because we've use more data than if we had a single small validation set.

For a given tuning parameter we calculate a average error value across all the K validation sets and their corresponding training sets)

K-fold Cross-validation

- for k = 1 to K
  - $\text{validation}_{(k)} = \text{segment}_k$
  - $\text{training}_{(k)}$ = the other segments
  - estimate $\hat{w}_\lambda^{(k)}$, the estimated coefficients for $\text{training}_{(k)}$ and the chosen tuning parameter λ.
  - compute the $\text{error}_{(k)}(\lambda)$, the error for $\hat{w}_\lambda^{(k)}$ using the validation segment k.
- computer the average error for $\hat{w}_\lambda$ as $CV(\lambda) = \frac{1}{K}\sum_{k=1}^{K} error_{(k)}(\lambda)$

We can run this cross-validation for each value of the tuning parameter λ that we want to consider, then choose the set of estimated coefficients $\hat{w}_\lambda$ for the lambda with the lowest $CV(\lambda)$.

Choosing K

The **best approximation of general error occurs when we choose K==N**, so that each validation set is a single row of our feature matrix – it is one set of features. This makes intuitive sense because this means we will have the maximum amount of training data possible (N-1 rows in each of N training sets). It also guarantees that every set of features is used to validate. In effect, we've used all the data to train and all the data to validate.

This for of K-fold Cross Validation is called Leave-One-Out (LOO) Cross validation because we leave one set of observations out of every training set to use for the validation set.

While **LOO Cross Validation** does the best job of approximating general error, it **is computational intensive**. It requires the $CV(\lambda)$ to be calculated across N training sets for L values of λ. To mitigate the

problem **it is common to use** 5 segments (**K==5 or 5-fold Cross Validation**) **or** (**K==10 or 10-fold Cross Validation**).

Exempting the Intercept from the L2 norm squared cost

The general case that we have used thus far includes a constant term in the regression formula and we apply the tuning parameter to this term as we do with all other terms. This keeps the intercept small, but this is not generally what we want. **A large intercept does not indicated over-fitting**; in fact the actual relationship in the data may have a large intercept.

In general we want to except the intercept from the effect of the L2norm squared cost term and it's tuning parameter.

Exempting the intercept in the closed form

Conceptually, we want the RSS cost to apply to all of the coefficients, but we only want the L2norm Squared term and it's tuning factor to be applied to the rest of the coefficients.

$$RSS(w_0, w_{rest}) + \lambda \|w_{rest}\|_2^2$$

In the closed form, we can do this by changing the identity matrix so that the [0,0] element is zero (of course it is no longer an identity matrix at that point, but we get the effect we want.) So our closed form solution becomes;

$$\widehat{w}^{ridge} = \left(H^T H + \lambda I^{modified}\right)^{-1} H^T y$$

Where $I^{modified}$ is an DxD identity matrix with the [0,0] element set to zero.

Exempting the intercept in the gradient descent algorithm

In this algorithm, we handle the jth coefficient specially and only apply the least squares cost to it, but apply the ridge cost to all other coefficients;

Ridge Regression Gradient Descent Algorithm Exempt Intercept

- while $\|\nabla RSS(\hat{w}^{(t)})\| > \varepsilon$
  - $\hat{y}^{(t)} \leftarrow Hw^{(t)}$ # predictions with current w
  - $residuals \leftarrow y - \hat{y}^{(t)}$ # residuals with current w
  - $sumOfSquares \leftarrow 0$ # we calc this incrementally
  - For j = 0 to D-1 # for each feature j
    #
    # when j = 0, the constant feature gets least squares derivative only
    # otherwise the feature derivative includes the l2norm term
    #
    - if j == 0 # constant feature contributes
      - $partial(j) \leftarrow 2h_j(x_i) \cdot residuals$ # lease squares to derivative
    - else # all other features contribute
      - $partial(j) \leftarrow 2h_j(x_i) \cdot residuals + 2\lambda w_j^{(t)}$ # least squares + l2norm # terms to derivative
    - $w_j^{(t+1)} \leftarrow w_j^{(t)} - \eta \times partial(j)$ # estimate jth coefficient
    - $sumOfSquares += partial(j) \cdot partial(j)$ # jth contribution to # sum of squares
  - $\|\nabla RSS(\hat{w}^{(t)})\| \leftarrow \sqrt{sumOfSquares}$ # RSS(w) for iteration t
  - t = t + 1
- return $w^{(t)}$ # return the last estimated w

Center the Observed y Values

Another way to handle this is the center the known y values around zero. When the y values are centered around zero, it will be common for the y-intercept to be close to zero, so the fact that ridge regression tries to keep this coefficient small does not affect it much.

- Transform y to have 0 mean
- Run the ridge regression normally