Other penalized regression methods & Cross-validation

In the last videos, we have covered Lasso, one of the penalized regression algorithms. Here we are going to cover some more algorithms.

The idea of penalized regression is to choose the coefficients $\hat{\beta}$ so as to avoid overfitting in the sample. This is achieved by penalizing the size of the coefficients by various penalty functions. Ideally, we should choose the level of penalization to minimize the out-of-sample or test data mean squared prediction error.

Let's start with the Ridge Regression,

Ridge Regression :

The Ridge method estimates coefficients by penalized least squares, where we minimize the sum of squared prediction error plus the penalty term given by the sum of the squared values of the coefficients times a penalty level λ. You can see this in the formula given here,

$$\hat{\beta}(\lambda) = \arg\min_{b \in \mathbb{R}^p} \sum_{i=1}^{n} (Y_i - b'X_i)^2 + \lambda \sum_j b_j^2$$

If we look at the formula, we notice that similarly to Lasso, the Ridge penalty presses down or penalizes the coefficients without sacrificing too much fit. In contrast to Lasso, Ridge penalizes the large values of coefficients much more aggressively and small values much less aggressively.

Because Ridge penalizes small coefficients very lightly, the Ridge fit is never sparse, and unlike Lasso, Ridge does not set estimated coefficients to zero and so it does not do the variable selection. Because of this property, ridge predictor, $\hat{\beta}X$ is especially well suited for prediction in the "dense" models, where the β_j's are all small, without necessarily being approximately sparse. In this case, it can easily outperform the Lasso predictor. Finally, we note that, in practice, we can choose the penalty level λ in Ridge by cross-validation.

Let's understand what is sparse and dense,
Sparse matrix/array or whatever is by definition when your matrix contains mostly zeros and few non-zero entries. On the other hand, a dense matrix/array is when you have a few zeros.

For example, when you apply LASSO regression, the sparsity of your learned coefficients depends on the amount of the penalty (lambda). The higher the penalty, the more sparse coefficients you get i.e the non-zero coefficients (selected variables). For example, if you have 100 independent variables in your regression, the LASSO may return only 10 non-zero variables. That means 10 non-zero variables and 90 zero variables. This is exactly what the meaning of sparsity is. In the case of Ridge, you will get a dense matrix because it never makes any coefficient to Zero.

Elastic Net :

The **elastic net** algorithm uses the combination of both lasso and Ridge. As you see in this formula,

$$\hat{\beta}(\lambda_1, \lambda_2) = \arg\min_{b \in \mathbb{R}^p} \sum_i (Y_i - b'X_i)^2 + \lambda_1 \sum_j b_j^2 + \lambda_2 \sum_j |b_j|$$

Here in the above equation, The second term corresponds to Ridge and the last term corresponds to the lasso. So we can say elastic net penalizes large coefficients as well as small coefficients. The penalty function has two penalty levels λ1 and λ2, which could be chosen by cross-validation in practice. By selecting different values of penalty levels λ1 and λ2, we can have more flexibility for building a good prediction rule than with just Ridge or Lasso.

Different cases for tuning values of lambda1 and lambda2.

1. If lambda1 and lambda2 are set to be 0, Elastic-Net Regression equals Ordinary least squares.
2. If lambda1 is set to be 0, Elastic-Net Regression equals Lasso Regression.
3. If lambda2 is set to be 0, Elastic-Net Regression equals Ridge Regression.
4. If lambda1 and lambda2 are set to be infinity, all weights are shrunk to zero

So, we should set lambda1 and lambda2 somewhere in between 0 and infinity.

We can have more flexibility for building a good prediction rule than with just Ridge or Lasso. We also note that the Elastic Net does perform variable selection unless we completely shut down the Lasso penalty by setting penalty level λ2 = 0. Elastic Net works well in regression models, where regression coefficients are either approximately sparse or "dense".

Lava Regression:

Lava is a combination of both lasso and Ridge. The difference over here is in this we are splitting the coefficient β(beta's) into γ(gamma) plus δ(delta), It's beta = gamma + delta. So we are doing this for all coefficients of the model and for all the split coefficients we are penalizing γ(gamma) like in Ridge and δ(delta) like in Lasso. There are two corresponding penalty levels λ1 and λ2, which can be chosen by cross-validation in practice. The penalty term which we are going to add is the square of delta and the absolute value of gamma.

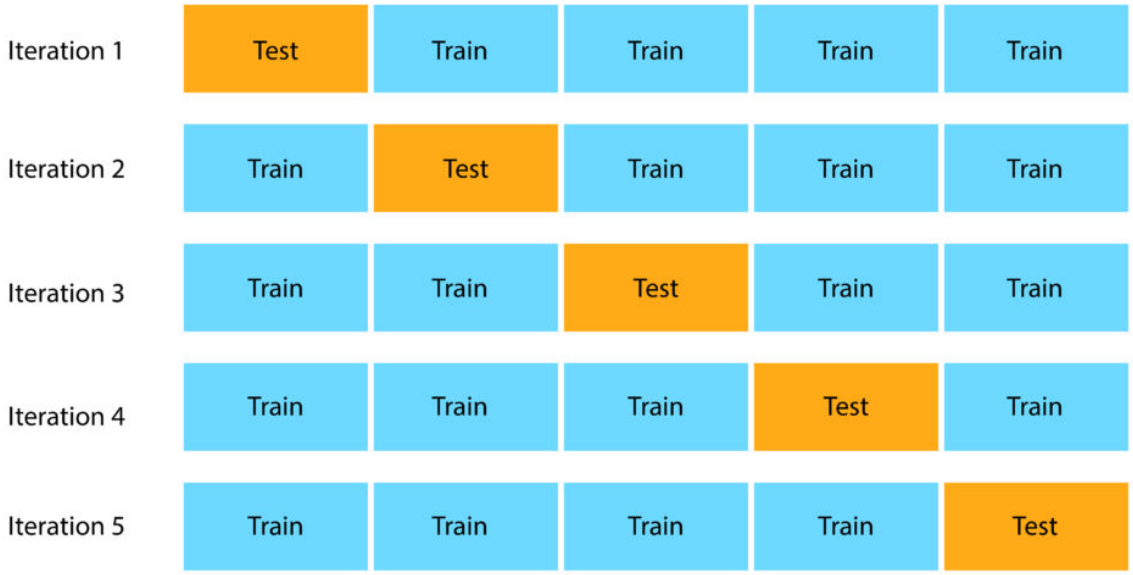
$$\hat{\beta}(\lambda_1, \lambda_2) = \hat{\gamma}(\lambda_1, \lambda_2) + \hat{\delta}(\lambda_1, \lambda_2)$$
$$= \arg \min_{\gamma+\delta \in \mathbb{R}^p} \sum_i (Y_i - (\gamma + \delta)' X_i)^2 + \lambda_1 \sum_j \gamma_j^2 + \lambda_2 \sum_j |\delta_j|$$

Because of this splitting, Lava penalizes coefficients least aggressively, compared to Ridge, Lasso, or Elastic Net, because it penalizes large coefficients like in Lasso and small coefficients like in Ridge. Lava never sets estimated coefficients to zero and so it does not do the variable selection. Lava works really well in "sparse + dense" models, where there are several large coefficients and many small coefficients, which are not necessarily sparse. In these types of models Lava significantly outperforms Lasso, Ridge, or Elastic Net.

Cross-Validation:

Cross-validation is a statistical method used to estimate the performance (or accuracy) of machine learning models. It is used to protect against overfitting in a predictive model, particularly in a case where the amount of data may be limited. It's an important and common practical tool that provides a way to choose tuning parameters such as the penalty levels. The idea of cross-validation is to rely on the repeated

splitting of the training data to estimate the potential out-of-sample predictive performance. In cross-validation, you make a fixed number of folds (or partitions) of the data, run the analysis on each fold, and then average the overall error estimate.



Cross-validation proceeds in 3 steps,

In Step 1: we divide the data into K blocks called "folds", for example in the above diagram, with K = 5, we split the data into 5 parts.

In Step 2: we begin by leaving one block out for testing and we fit a prediction rule on all other blocks. We then predict outcome observations in the left out block or testing block, and record the empirical Mean Squared Prediction Error. We repeat this process for each block.

In Step 3: we average the empirical Mean Squared Prediction Errors over blocks. We do these steps for several or many values of the tuning parameters and we choose the best tuning parameter to minimize the Averaged Mean Squared Prediction Error.

Let's understand more formally and mathematically,

$$B_1, \ldots, B_K$$

Step 1: We randomly select equal-sized blocks B1 through BK to randomly partition the observations indices 1 through n. This step is the same as step 1 above which is nothing but dividing data into k-blocks.

$$\hat{f}_{-k}(X;\theta)$$

Step 2: We then fit a prediction rule denoted by $\widehat{f_{-k}}$ of X and θ, where θ denotes tuning parameters such as penalty levels, and $\widehat{f_{-k}}$ depends only on observations that are not in block Bk. This says we are fitting any prediction rule such as ridge or lasso in this case we want to tune lambda which is denoted as θ here. We are fitting for k-1 folds.
The empirical MSE for the block of observations Bk is given by the average squared prediction error for this block as shown in the formula. In this formula m is the size of the block.

$$\text{MSE}_k(\theta) = \frac{1}{m}\sum_{i\in B_k}(Y_i - \hat{f}_{-k}(X_i;\theta))^2$$

Here we are evaluating the performance of the left-out block while training.

Step 3: The cross-validated MSE is the average of MSEs for each block, as shown in this formula. Here we are taking the average of all the scores.

$$\text{CV-MSE}(\theta) = \frac{1}{K}\sum_{k=1}^{K}\text{MSE}_k(\theta)$$

Finally, the "best" tuning parameter θ is chosen by minimizing the cross-validated MSE.

We now provide some concluding remarks,

1. First, we note that, in contrast to Lasso, the theoretical properties of Ridge and other penalized procedures are less well understood in the high-dimensional case (yet). So it is a good idea to rely on test data to assess their predictive performance.
2. Second, we note that cross-validation is a good way to choose penalty levels, but its theoretical properties are also not completely understood in the high-dimensional case (yet). So, again, it is a good idea to rely on test data to assess the predictive performance of cross-validated rules.