Linear Regression with Penalites

D2K Course Staff

2/28/2019

Introduction

A linear model obtained through ordinary least squares is one of the most ubiquitous models in statistics. It is usually one of the very first models that we ever learn about. This is largely due to its inferential nature. It is quite easy to derive its MLE estimators and interpret them for lay-people. However, we can also use linear regression to make predictions on future data like other machine learning methods. In fact, linear regression is usually one of the very first methods applied to any regression learning task. In this module, we will show an example of how linear regression can be used for prediction, as well as extensions of linear regression that can improve its predictive performance.

N.B.: we use the term "regression" for continuous outcome variables and we use "classification" for discrete outcome variables.

Linear Regression

We define the linear regression model as,

$$y = X\beta + \epsilon$$

where y and ϵ are $n \times 1$ vectors that represent the outcome variable and errors respectively, X is the $n \times p$ data matrix, and β is the $p \times 1$ vector of unknown parameters calculated via OLS. We calculate those parameters by optimizing the following equation,

$$\underset{\beta}{\operatorname{argmin}} ||y - X\beta||_2^2$$

which has the closed form solution $\hat{\beta} = (X^T X)^{-1} X^T y$.

Often we will use a metric like MSE to assess how well our model performs. People commonly calculate the MSE between the fitted $\hat{y}$'s and the actual y values to assess how well the regression models the data. However, we know that assessing a models predictive accuracy on current or "training" data is a biased method of determining the quality of a model. Instead, we should assess our model's accuracy on future or "test" data in order to get an *unbiased* measure of predictive accuracy. Calculating the MSE on the training data is a perfectly reasonable thing to do when the task is making inference on historical data, but if we are using linear regression in the context of prediction we need to be reporting prediction accuracy on *new* data. We show how to predict on new data with a linear model fitted on training data in the code example below.

We use an example dataset `mtcars` and split the data into our training/validation, query, and test sets so that we can train, compare and evaluate our final model with unseen data.

```
# Set seed for reproducibility
set.seed(123)

### Create 60-20-20 split for training/validation, query, test
### sets ###
```

```r
# Read in data, load as matrix 'x' and vector 'y'
data(mtcars)
summary(mtcars)
```

```
      mpg             cyl             disp             hp       
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
 Median :19.20   Median :6.000   Median :196.3   Median :123.0  
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
      drat             wt             qsec             vs        
 Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
 1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
 Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
 Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
 3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
 Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
       am              gear            carb      
 Min.   :0.0000   Min.   :3.000   Min.   :1.000  
 1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
 Median :0.0000   Median :4.000   Median :2.000  
 Mean   :0.4062   Mean   :3.688   Mean   :2.812  
 3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
 Max.   :1.0000   Max.   :5.000   Max.   :8.000  
```

```r
# Set mpg as outcome variable
y <- mtcars$mpg

# Set all other variables as predictors
x <- mtcars %>% select(cyl, disp, hp, drat, wt, qsec, vs, am,
    gear, carb)

# Create training/test index
tr_idx <- sample(1:nrow(x), nrow(x) * 0.6, replace = FALSE)

# Create training/validation set
x_train <- x[tr_idx, ]
y_train <- y[tr_idx]

x_test <- x[-tr_idx, ]
y_test <- y[-tr_idx]

# Create query set from test set
qr_idx <- sample(1:nrow(x_test), nrow(x_test) * 0.5, replace = FALSE)

x_query <- x_test[qr_idx, ]
y_query <- y_test[qr_idx]

x_test <- x_test[-qr_idx, ]
y_test <- y_test[-qr_idx]
```

It is important to not only create a test set, but also a query set since we will be comparing the performance of the linear regression model to other linear models we will introduce later on. Now that we have chunked

our data to keep the query and test sets "hidden" for now, we can fit our linear model on the training data before evaluating it on the query set and reporting the MSE.

```
# Combine x and y for lm function
train <- data.frame(mpg = y_train, x_train)
query <- data.frame(mpg = y_query, x_query)
test <- data.frame(mpg = y_test, x_test)

# Fit linear regression model on training data
lm_fit <- lm(mpg ~ ., data = train)

# Predict new values using the X values in the query data
y_query_lm <- predict(lm_fit, query)

# Calculate MSE on the query set
lm_mse <- sum((y_query - y_query_lm)^2)/length(y_query)
lm_mse
```

```
[1] 21.58362
```

We can see the MSE of the linear regression model on the query set above. It is hard to determine how "good" of a model this is without comparing it to other models. Next, we will examine different versions of linear regression and see if any variations perform better than OLS linear regression.

Penalized Linear Regression

A big issue with linear regression is that it treats all of the predictor variables, or columns of the data matrix X as equally predictive of the outcome variable y. This is rarely the case, and traditionally methods such as forward, backward, or step-wise selection have been used to select an optimal subset of variables for prediction. However, these methods are usually unnecessarily computationally expensive, and often infeasible when the number of predictors p is very large.

To combat this issue, different scholars determined different ways to select and shrink features based on their predictive ability algorithmically. They did so by introducing penalty terms to the OLS optimization problem using L1 and L2 norms. We investigate different variations of penalized linear regression and analyze how each methods performs on the query set to determine which model performs best.

Lasso Linear Regression

The Lasso takes the basic linear model we defined using OLS in the first example, but adds an L1 norm penalty term to the optimization problem.

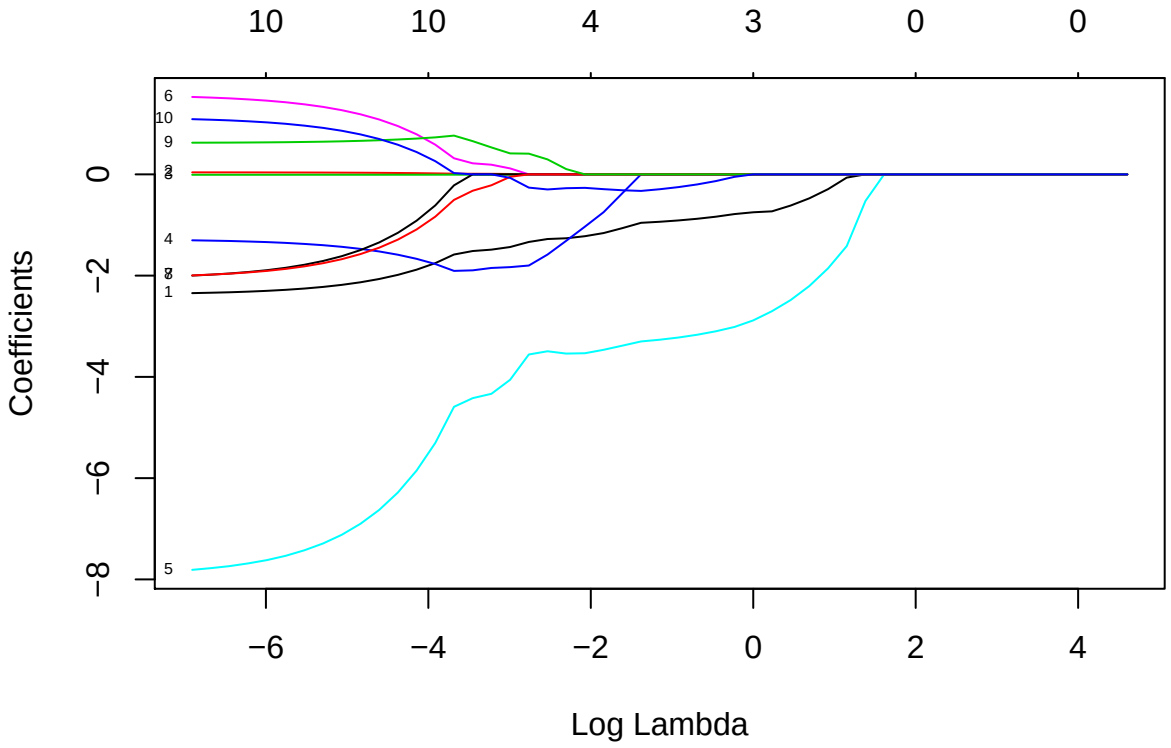
$$\underset{\beta}{\text{argmin}}||y - X\beta||_2^2 + \lambda||\beta||_1$$

As λ increases, the β values are shrunk to 0. This allows us to perform automatic feature selection during the cross-validation process. Additionally, it allows us to filter out features that are the least predictive of our outcome variable y and allows us to improve our predictive ability by reducing the noise in the model. We can visualize how λ affects the shrinkage features by plotting the coefficient values across the values of λ.

N.B.: for the lasso, set $\alpha = 1$ for the L1 norm, we will touch on this later

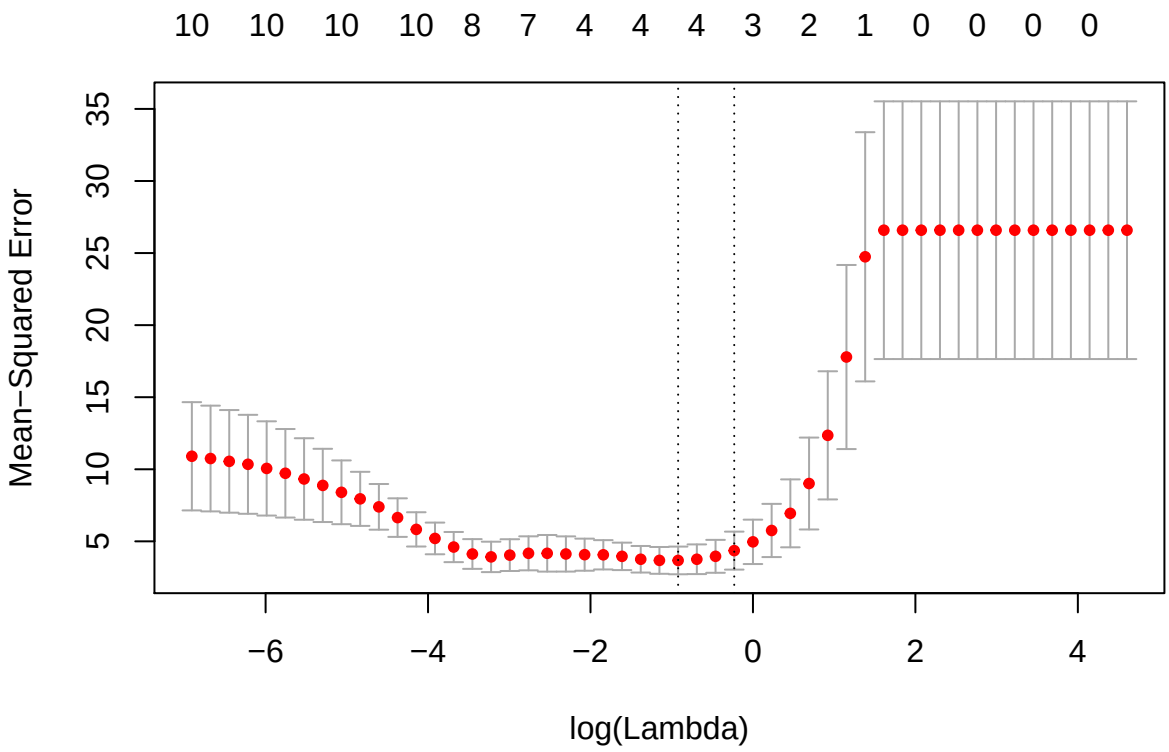
```
# Set seed for reproducibility
set.seed(123)
```

```r
# Load glmnet package
library(glmnet)

# Set lambda values
lambdas <- 10^seq(-3, 2, by = 0.1)

# Fit lasso model on training set
lasso_fit <- glmnet(as.matrix(x_train), y_train, alpha = 1, lambda = lambdas)

# Plot coefficient values vs. lambda
plot(lasso_fit, xvar = "lambda", label = TRUE)
```



You can see that as λ (or log of λ) increases, that the coefficients for each feature eventually shrink to 0. If λ is too small, we do not reduce the noise enough and we naively underfit. If λ is too large, we make our model overly complex and overfit. Thus, we can define λ as a hyper-parameter that we must tune via cross-validation. We show an example of the implementation of the Lasso using the `glmnet` package and the built in cross-validation function. We can also visualize the error rate as lambda increases.

```r
# Set seed for reproducibility
set.seed(123)

# Find optimal lambda using built-in cross validation in
# glmnet package
lasso_cv <- cv.glmnet(as.matrix(x_train), y_train, type.measure = "mse",
    alpha = 1, lambda = lambdas)

# Visualize error vs. lambda
plot(lasso_cv, col = "red")
```



From this visualization you can see where the value of λ occurs that minimizes the MSE. We can then take this value as the optimal value of λ and predict on the query set to assess our overall model accuracy.

```
# Predict on query set
y_query_lasso <- predict(lasso_cv, newx = as.matrix(x_query),
    s = "lambda.min")

# Calculate MSE on the query set
lasso_mse <- sum((y_query - y_query_lasso)^2)/length(y_query)
lasso_mse
```

```
[1] 19.95077
```

We can see that the Lasso performs better than just OLS linear regression when predicting **mpg**. We can see which variables were zero'd out at the optimal λ penalty value.

```
coef(lasso_cv, s = "lambda.min")
```

```
11 x 1 sparse Matrix of class "dgCMatrix"
                      1
(Intercept) 36.573997387
cyl         -0.910261352
disp         .
hp          -0.001865066
drat         .
wt          -3.222566927
qsec         .
vs           .
am           .
gear         .
carb        -0.251758797
```

We can see that the variables **disp**, **drat**, **qsec**, **vs**, **am**, and **gear** were zero'd at the optimal λ. By doing

so, we were able to increase our predictive ability and perform automatic feature selection without iterating through every permutation of our variables like other feature selection algorithms.

Ridge Linear Regression

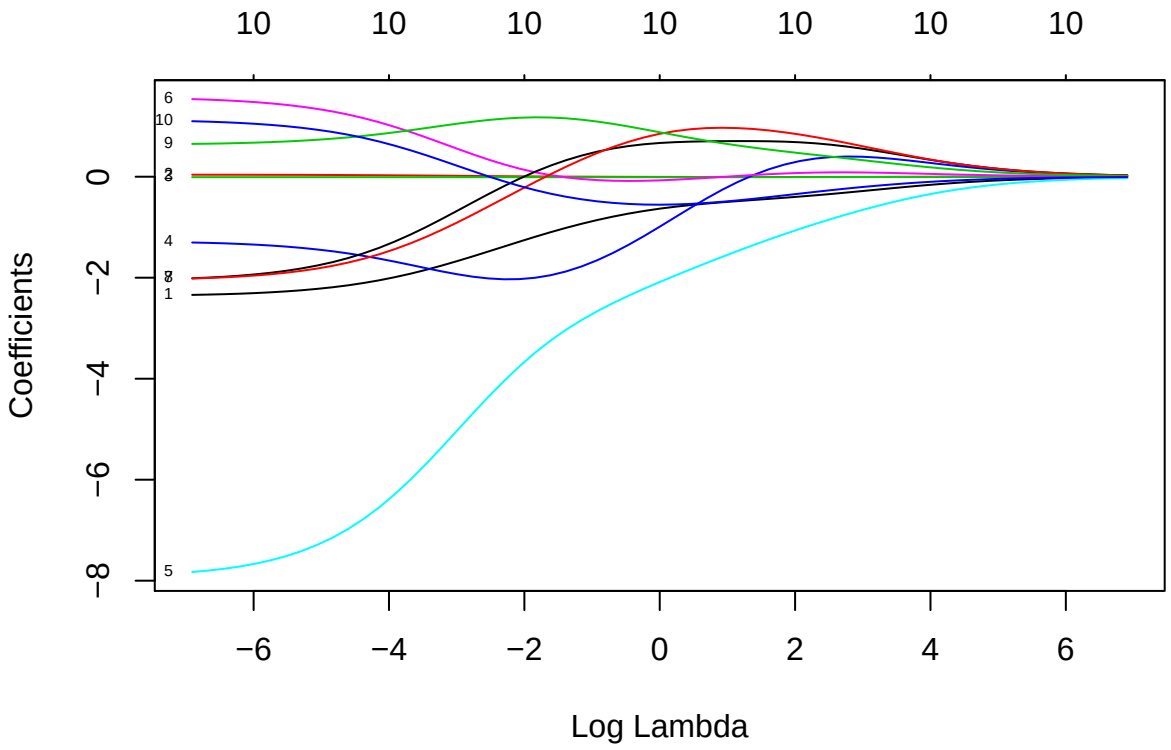
The Ridge takes the basic linear model we defined using OLS in the first example, but adds an L2 norm penalty term to the optimization problem instead of the L1 norm.

$$\underset{\beta}{\text{argmin}}||y - X\beta||_2^2 + \lambda||\beta||_2$$

As λ increases, the β values are shrunk and **approach** 0. This doesn't allow us to explicitly select features, but it does allow us to minimize their effect on the prediction and to improve our predictive ability still by reducing the noise in the model. We can visualize how λ affects the shrinkage features by plotting the coefficient values across the values of λ.

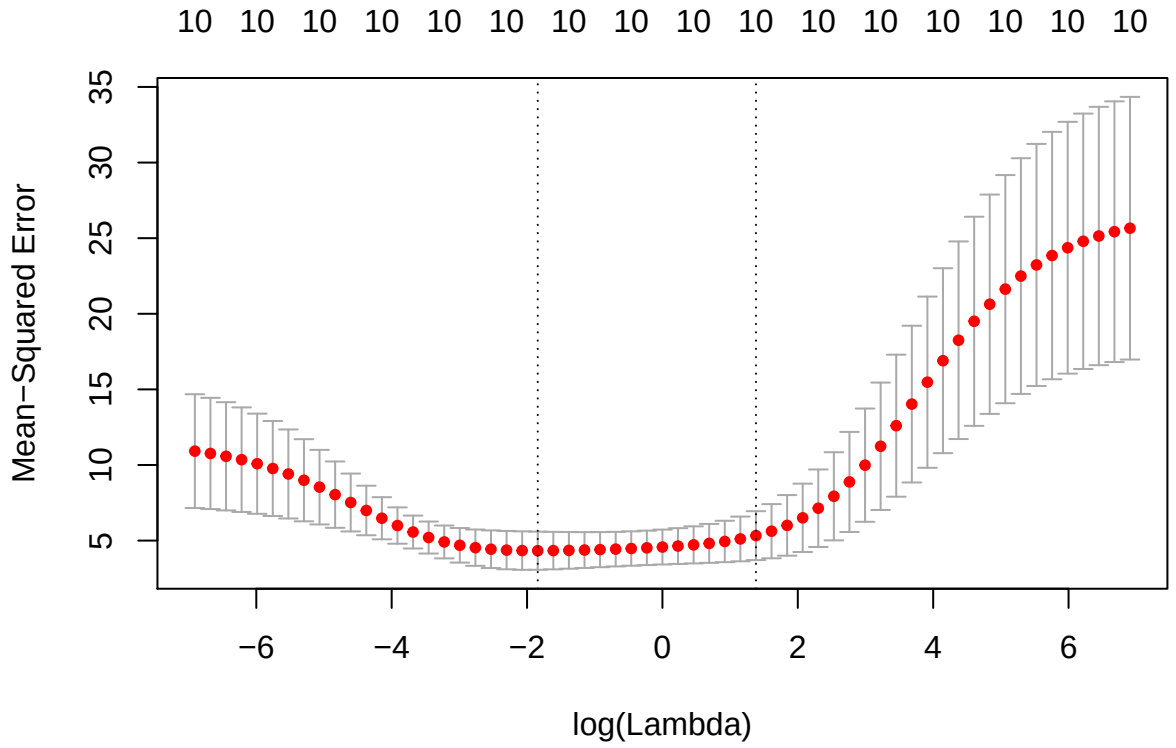
N.B.: for ridge regression, we set $\alpha = 0$ for the L2 norm, again we will touch on this later

```
# Set seed for reproducibility
set.seed(123)

# Set lambda values
lambdas <- 10^seq(-3, 3, by = 0.1)

# Fit ridge model on training set
ridge_fit <- glmnet(as.matrix(x_train), y_train, alpha = 0, lambda = lambdas)

# Plot coefficient values vs. lambda
plot(ridge_fit, xvar = "lambda", label = TRUE)
```



As you can see, the coefficient values merely **approach** 0 due to us using the L2 norm instead of the L1 norm. Just like we did for the Lasso, we must tune the value of λ using cross-validation to find the optimal value we can use for predicting on the query set.

```
# Set seed for reproducibility
set.seed(123)

# Find optimal lambda using built-in cross validation in
# glmnet package
ridge_cv <- cv.glmnet(as.matrix(x_train), y_train, type.measure = "mse",
    alpha = 0, lambda = lambdas)

# Visualize error vs. lambda
plot(ridge_cv)
```



Again, you can see where the MSE is minimized for the given values of λ. We can then take the value of λ that minimizes the MSE and use that to predict on the query set and assess the predictive ability of ridge regression compared to the previous two methods.

```
# Predict on query set
y_query_ridge <- predict(ridge_cv, newx = as.matrix(x_query),
    s = "lambda.min")

# Calculate MSE on the query set
ridge_mse <- sum((y_query - y_query_ridge)^2)/length(y_query)
ridge_mse
```

```
[1] 20.3712
```

This performs slightly worse than the Lasso, but still better than OLS linear regression. However, it is worth noting that studies have shown that Ridge regression performs better at prediction than the Lasso. It is important to try many different methods when building a predictive model.

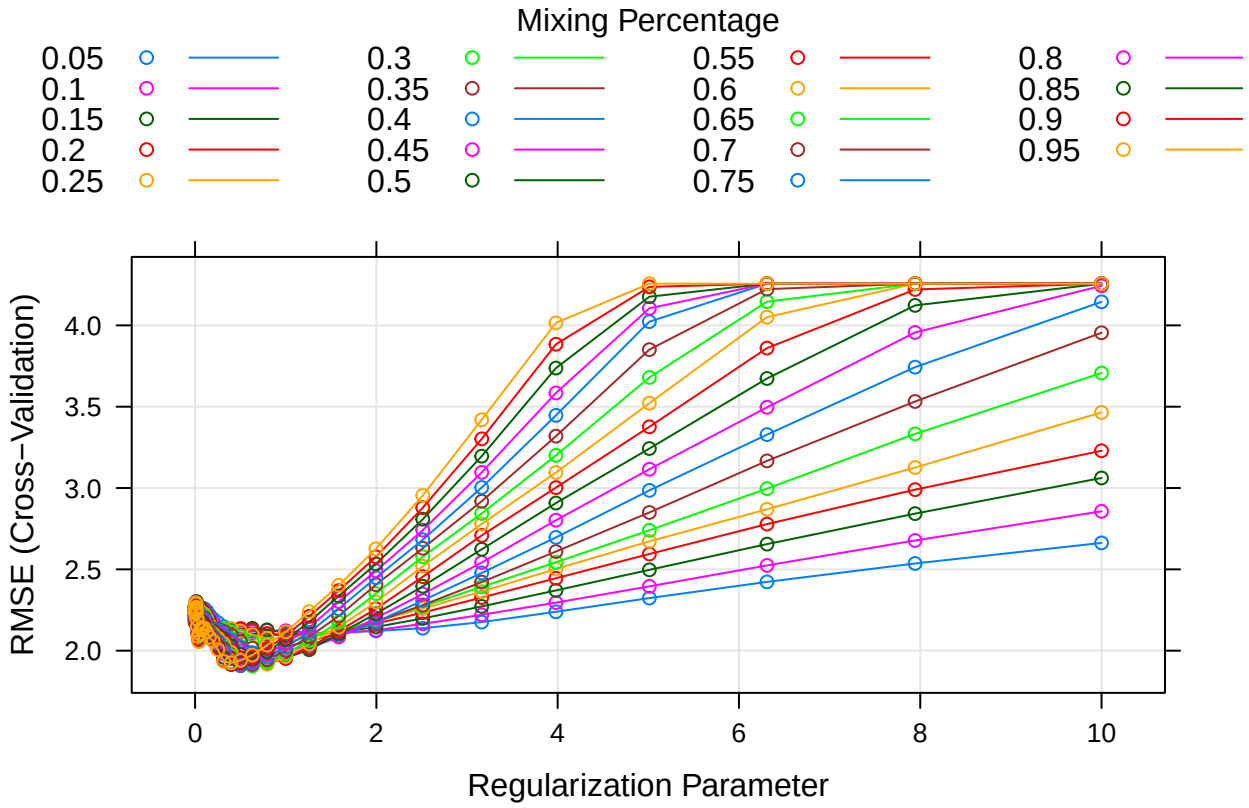
Elastic Net

Now that we have covered both the Lasso and Ridge regression, we combine the two penalty terms to form what is called the Elastic Net. The Elastic Net uses both L1 and L2 norm penalties and introduces a second hyper-parameter α that determines how much of each penalty term is induced in the optimization problem. For the Elastic Net, we write the optimization problem to calculate the coefficients as follows,

$$\underset{\beta}{\text{argmin}}||y - X\beta||_2^2 + \lambda[\alpha||\beta||_1 + (1-\alpha)||\beta||_2]$$

where α controls how much of each penalty term is implemented. Just like λ, it is a hyper-parameter that can be tuned using cross-validation to find the optimal value for prediction. We show an implementation below using the `caret` package so that we can tune alpha as well.

```
# Set seed for reproducibility
set.seed(123)

# Load caret library
library(caret)

# Set lambda values
lambdas <- 10^seq(-3, 1, by = 0.1)

# Initialize vector of alphas
alphas <- seq(0.05, 0.95, 0.05)

# Create grid of values to search over
tuneGrid <- expand.grid(alpha = alphas, lambda = lambdas)

# Create model training controls for CV
fitControl <- trainControl(method = "cv", number = 10)

# Train elastic net with cross-validation
cv_elnet <- train(mpg ~ ., data = train, method = "glmnet", metric = "RMSE",
    tuneGrid = tuneGrid, trControl = fitControl)

# Plot the error vs. lambda for all alphas
plot(cv_elnet)
```



We can see the Mean-Squared Error as a function of λ for all of the different of values of α that we searched over. We can then take our best version of the model from cross-validation and use it to predict on the query set.

```
# Predict y query values using best elastic net model
y_query_elnet <- predict(cv_elnet, query)

# Calculate MSE on the query set
elnet_mse <- sum((y_query - y_query_elnet)^2)/length(y_query)
elnet_mse
```

```
[1] 20.3513
```

We can see that in this case that the Elastic Net does not perform as well as the Lasso, but still better than OLS linear regression.

Final Model

After comparing all of our models at their optimal configuration using the cross-validation and a query set, we can now retrain our best model on all of the already seen data (i.e. traing **and** query sets) and report our final accuracy on the test set. Since our best model was the Lasso, we will use that to report our final MSE.

```
# Set seed for reproducibility
set.seed(123)

# Combine training and query sets
x_train_query <- data.frame(rbind(x_train, x_query))
y_train_query <- c(y_train, y_query)
```

```r
# Find optimal lambda on combined sets using built-in cross
# validation in glmnet package
final_lasso_cv <- cv.glmnet(as.matrix(x_train_query), y_train_query,
    type.measure = "mse", alpha = 1, lambda = lambdas)

# Predict on test set
y_test_pred <- predict(final_lasso_cv, newx = as.matrix(x_test),
    s = "lambda.min")

# Calculate MSE on the test set
test_mse <- sum((y_test - y_test_pred)^2)/length(y_test)
test_mse
```

```
[1] 10.7061
```

We can see that we actually performed better on the final test set than we did on any of the query sets. This indicates that we did a good job avoiding overfitting and that we were able to select a model that performs well on future data. The variation in MSE values between the query and the test set is also likely due to the small sample size of our data. Such differences are way less frequent when working on larger data sets.

References

Hastie, Trevor, et al. The Elements of Statistical Learning Data Mining, Inference, and Prediction. Springer, 2009.

Hastie, Trevor, and Junyang Qian. "Glmnet Vignette." Web.stanford.edu, Stanford University, 26 June 2014, web.stanford.edu/~hastie/glmnet/glmnet_alpha.html.

Tibshirani, Robert. "Regression Shrinkage and Selection Via the Lasso." Journal of the Royal Statistical Society: Series B (Methodological), vol. 58, no. 1, 1996, pp. 267–288., doi:10.1111/j.2517-6161.1996.tb02080.x.

Zou, Hui, and Trevor Hastie. "Regularization and Variable Selection via the Elastic Net." Journal of the Royal Statistical Society: Series B (Statistical Methodology), vol. 67, no. 2, 2005, pp. 301–320., doi:10.1111/j.1467-9868.2005.00503.x.