Regularization in Linear Regression

Applied Machine Learning for Educational Data Science

true

10/20/2021

Contents

[Updated: Mon, Oct 25, 2021 - 00:44:46]

Regularization

Regularization is a general strategy to incorporate additional penalty terms into the model fitting process and used not just for regression but a variety of other types of models. The idea behind the regularization is to constrain the size of regression coefficients with the purpose of reducing their sampling variation and, hence, reducing the variance of model predictions. These constrains are typically incorporated into the loss function to be optimized. There are two commonly used regularization strategy: **ridge penalty** and **lasso penalty**. In addition, there is also **elastic net**, a mixture of these two strategies.

Ridge Regression

Ridge Penalty

Remember that we formulated the loss function for the linear regression as the sum of squared residuals across all observations. For ridge regression, we add a penalty term to this loss function and this penalty term is a function of all the regression coefficients in the model. Assuming that there are P regression coefficients in the model, the penalty term for the ridge regression would be

$$\lambda \sum_{i=1}^{P} \beta_p^2,$$

where λ is a parameter that penalizes the regression coefficients when they get larger. Therefore, when we fit a regression model with ridge penalty, the loss function to minimize becomes

$$Loss = \sum_{i=1}^{N} \epsilon_{(i)}^2 + \lambda \sum_{i=1}^{P} \beta_p^2,$$

$$Loss = SSR + \lambda \sum_{i=1}^{P} \beta_p^2.$$

Let's consider the same example from the previous class. Suppose we fit a simple linear regression model such that the readability score is the outcome (Y) and average word length is the predictor(X). Our regression model is

$$Y = \beta_0 + \beta_1 X + \epsilon,$$

and let's assume the set of coefficients are $\{\beta_0, \beta_1\} = \{7.5,\text{-}2\}$, so my model is

$$Y = 7.5 - 2X + \epsilon.$$

Then, the value of the loss function when $\lambda = 0.2$ would be equal to 27.433.

```
readability_sub <- read.csv('https://raw.githubusercontent.com/uo-datasci-specialization/c4-ml-fall-202

d <-  readability_sub[,c('mean.wl','target')]

b0 = 7.5
b1 = -2

d$predicted <- b0 + b1*d$mean.wl
d$error     <- d$target - d$predicted

d
```

```
    mean.wl      target  predicted       error
1  4.603659 -2.58590836 -1.7073171 -0.87859129
2  3.830688  0.45993224 -0.1613757  0.62130790
3  4.180851 -1.07470758 -0.8617021 -0.21300545
4  4.015544 -1.81700402 -0.5310881 -1.28591594
5  4.686047 -1.81491744 -1.8720930  0.05717559
6  4.211340 -0.94968236 -0.9226804 -0.02700194
7  4.025000 -0.12103065 -0.5500000  0.42896935
```

```
8  4.443182 -2.82200582 -1.3863636 -1.43564218
9  4.089385 -0.74845172 -0.6787709 -0.06968077
10 4.156757  0.73948755 -0.8135135  1.55300107
11 4.463277 -0.96218937 -1.4265537  0.46436430
12 5.478261 -2.21514888 -3.4565217  1.24137286
13 4.770492 -1.21845136 -2.0409836  0.82253224
14 4.568966 -1.89544351 -1.6379310 -0.25751247
15 4.735751 -0.04101056 -1.9715026  1.93049203
16 4.372340 -1.83716516 -1.2446809 -0.59248431
17 4.103448 -0.18818586 -0.7068966  0.51871069
18 4.042857 -0.81739314 -0.5857143 -0.23167886
19 4.202703 -1.86307557 -0.9054054 -0.95767016
20 3.853535 -0.41630158 -0.2070707 -0.20923088
```

```
lambda = 0.2

loss <- sum((d$error)^2) + lambda*(b0^2 + b1^2)

loss
```

```
[1] 27.43364
```

Notice that when λ is equal to zero, the loss function is identical to SSR; therefore, it becomes a linear regression with no regularization. As the value of λ increases, the degree of penalty linearly increases. Technically, the λ can take any positive value between 0 and ∞.

As we did in the previous lecture, imagine that we computed the loss function with the ridge penalty term for every possible combination of the intercept (β_0) and the slope (β_1). Let's say the plausible range for the intercept is from -10 to 10 and the plausible range for the slope is from -2 to 2. Now, we also have to think different values of λ because the surface we try to minimize is dependent on the value λ and different values of λ yield different estimates of β_0 and and β_1.

You can try a number of different values for λ using the shiny app at this link and explore how the loss function value and coefficient estimates change for different values of λ. Note that when λ is equal to zero, it should be equivalent of what we have seen in the earlier lecture. Try values of 1, 5, 10, 50, and 100.

Below is also a demonstration of what happens to loss function and the regression coefficients for increasing levels of ridge penalty (λ).

Model Estimation

Matrix Solution The matrix solution we learned before for regression without regularization can also be applied to estimate the coeffients from ridge regression given the λ value. Given that

- $\mathbf{Y}$ is an N x 1 column vector of observed values for the outcome variable,
- $\mathbf{X}$ is an N x (P+1) **design matrix* for the set of predictor variables including an intercept term,
- $\boldsymbol{\beta}$ is an (P+1) x 1 column vector of regression coefficients,
- $\mathbf{I}$ is a (P+1) x (P+1) identity matrix,
- and λ is positive real-valued number,

the set of ridge regression coefficients can be estimated using the following matrix operation.

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\mathbf{T}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\mathbf{T}\mathbf{Y}$$

Now, suppose we want to predict the readability score by using the two predictors, the average word length (X_1) and number of sentences (X_2). Our model will be

$$Y_{(i)} = \beta_0 + \beta_1 X_{1(i)} + \beta_2 X_{2(i)} + \epsilon_{(i)}.$$

If we estimate the ridge regression coefficients by using $\lambda = .5$, the estimates would be $\{\beta_0, \beta_1, \beta_2\} = \{0.277,-.593,0.097\}$.

```
Y <-  as.matrix(readability_sub$target)
X <-  as.matrix(cbind(1,readability_sub$mean.wl,readability_sub$sents))

lambda <- 0.5

beta <- solve(t(X)%*%X + lambda*diag(ncol(X)))%*%t(X)%*%Y

beta
```

```
            [,1]
[1,]  0.27693153
[2,] -0.59327091
[3,]  0.09692781
```

If we change the value of λ to 2, then we will get a different set of estimates for the regression coefficients.

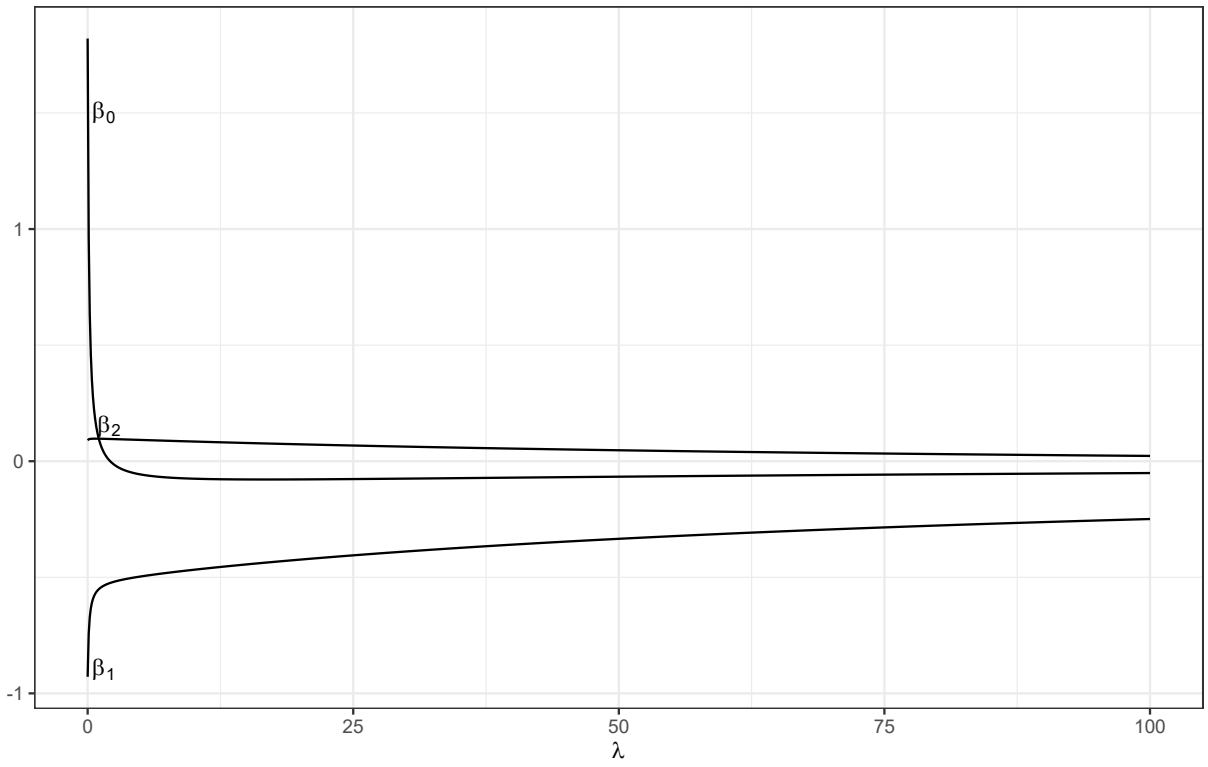
```
Y <-  as.matrix(readability_sub$target)
X <-  as.matrix(cbind(1,readability_sub$mean.wl,readability_sub$sents))

lambda <- 2

beta <- solve(t(X)%*%X + lambda*diag(ncol(X)))%*%t(X)%*%Y

beta
```

```
             [,1]
[1,]  0.006012867
[2,] -0.526374942
[3,]  0.095845692
```

We can manipulate the value of λ from 0 to 100 with increments of .1 and calculate the regression coefficients for every possible value of λ. Note the regression coefficients will shrink towards zero, but will never be exactly equal to zero in ridge regression.

```
Y <-  as.matrix(readability_sub$target)
X <-  as.matrix(cbind(1,readability_sub$mean.wl,readability_sub$sents))

lambda <- seq(0,100,.1)

beta     <- data.frame(matrix(nrow=length(lambda),ncol=4))
beta[,1] <- lambda

for(i in 1:length(lambda)){
  beta[i,2:4] <- t(solve(t(X)%*%X + lambda[i]*diag(ncol(X)))%*%t(X)%*%Y)
```

```
}

ggplot(data = beta)+
  geom_line(aes(x=X1,y=X2))+
  geom_line(aes(x=X1,y=X3))+
  geom_line(aes(x=X1,y=X4))+
  xlab(expression(lambda))+
  ylab('')+
  theme_bw()+
  annotate(geom='text',x=1.5,y=1.5,label=expression(beta[0]))+
  annotate(geom='text',x=2,y=.15,label=expression(beta[2]))+
  annotate(geom='text',x=1.5,y=-.9,label=expression(beta[1]))
```



Standardized Variables We haven't considered a very important issue for the model estimation. This issue is not necessarily important if you have only one predictor. However, it is critical whenever you have more than one predictor. Different variables have different scales and therefore the magnitude of the regression coefficients for different variables will be dependent on the scales of the variables. A regression coefficient for a predictor with a range from 0 to 100 will be very different than a regression coefficient for a predictor with a range from 0 to 1. Therefore, if we work with the unstandardized variables, ridge penalty will be amplified for the coefficients of those variables with a larger range of values.

Therefore, it is critical that we standardize variables before we use ridge regression. Let's do the example in the previous section, but we now first standardize the variables in our model.

```
Y <-  as.matrix(readability_sub$target)

X <-  as.matrix(cbind(readability_sub$mean.wl,readability_sub$sents))
```

```
# Standardize Y

  Y <- scale(Y)

  Y
```

```
             [,1]
 [1,] -1.49010043
 [2,]  1.58384679
 [3,]  0.03504552
 [4,] -0.71410074
 [5,] -0.71199490
 [6,]  0.16122446
 [7,]  0.99752285
 [8,] -1.72837656
 [9,]  0.36431202
[10,]  1.86598181
[11,]  0.14860203
[12,] -1.11591963
[13,] -0.11002472
[14,] -0.79326406
[15,]  1.07828135
[16,] -0.73444792
[17,]  0.92974794
[18,]  0.29473442
[19,] -0.76059743
[20,]  0.69952720
attr(,"scaled:center")
[1] -1.109433
attr(,"scaled:scale")
[1] 0.9908565
```

```
# Standardized X

  X <- scale(X)
  X
```

```
            [,1]       [,2]
 [1,]  0.6695829 -0.7833675
 [2,] -1.3062112  1.6269940
 [3,] -0.4111573  0.7231084
 [4,] -0.8336993 -0.7833675
 [5,]  0.8801752 -0.9340151
 [6,] -0.3332238  0.8737560
 [7,] -0.8095289 -0.3314247
 [8,]  0.2593876 -1.2353102
 [9,] -0.6449529 -0.4820723
[10,] -0.4727448  2.3802319
[11,]  0.3107526  0.4218133
[12,]  2.9051581 -0.3314247
[13,]  1.0960262 -0.3314247
[14,]  0.5809039 -0.6327199
```

```
[15,]  1.0072258  1.0244036
[16,]  0.0783096  0.4218133
[17,] -0.6090069 -0.9340151
[18,] -0.7638842 -0.9340151
[19,] -0.3553022 -0.7833675
[20,] -1.2478105  1.0244036
attr(,"scaled:center")
[1]  4.341704 12.200000
attr(,"scaled:scale")
[1] 0.3912203 6.6380086
```

When we standardize the variables, the mean all variables become zero. So, the intercept estimate for any regression model with standardized variables is guaranteed to be zero. Note that our design matrix doesn't have a column of ones anymore because it is unnecessary (it would be a column of zeros if we had).

First, let's check the coefficients of the regression model with standardized variables when there is no ridge penalty.

```
lambda <- 0

beta.s <- solve(t(X)%*%X + lambda*diag(ncol(X)))%*%t(X)%*%Y

beta.s
```

```
           [,1]
[1,] -0.3666326
[2,]  0.6049359
```

Now, let's increase the ridge penalty to 0.5.

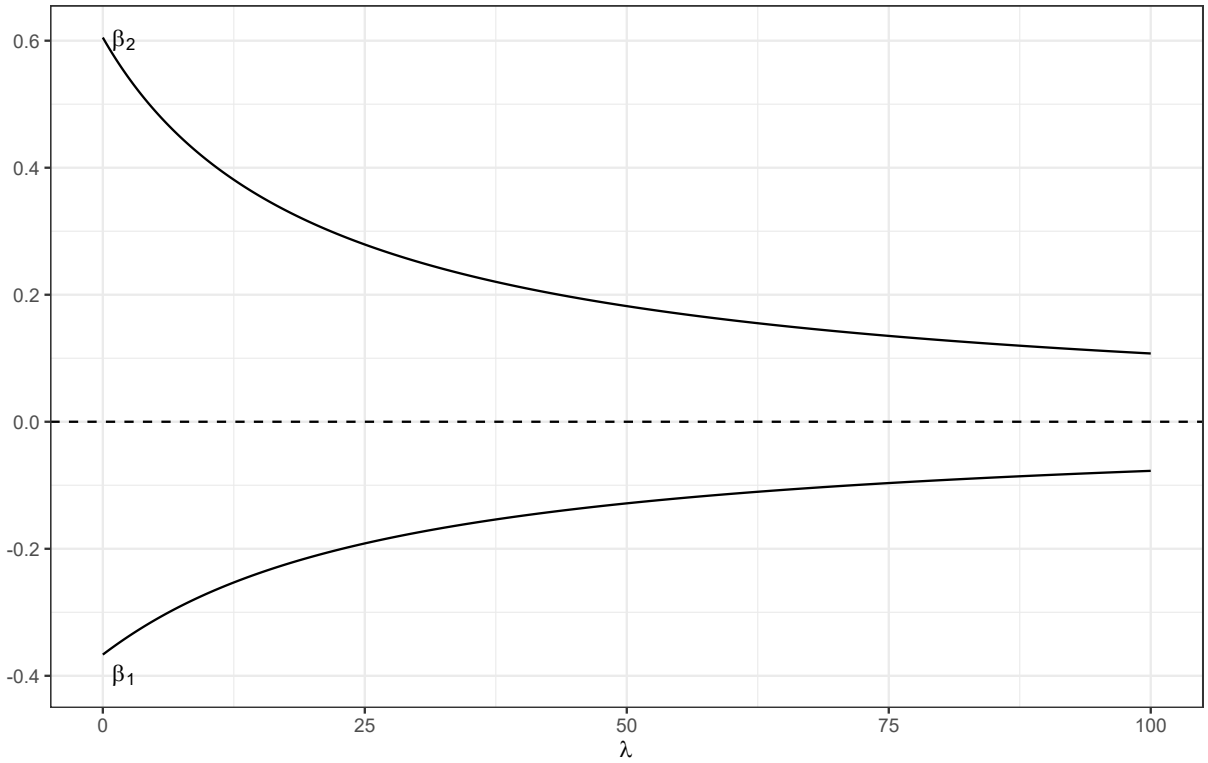
```
lambda <- 0.5

beta.s <- solve(t(X)%*%X + lambda*diag(ncol(X)))%*%t(X)%*%Y

beta.s
```

```
           [,1]
[1,] -0.3604763
[2,]  0.5908420
```

Below, we can manipulate the value of λ from 0 to 100 with increments of .1 as we did before and calculate the standardized regression coefficients for every possible value of λ.

```
Y <-  as.matrix(readability_sub$target)
X <-  as.matrix(cbind(readability_sub$mean.wl,readability_sub$sents))

Y <- scale(Y)
X <- scale(X)

lambda <- seq(0,100,.1)

beta     <- data.frame(matrix(nrow=length(lambda),ncol=3))
beta[,1] <- lambda
```

```r
for(i in 1:length(lambda)){
  beta[i,2:3] <- t(solve(t(X)%*%X + lambda[i]*diag(ncol(X)))%*%t(X)%*%Y)
}

ggplot(data = beta)+
  geom_line(aes(x=X1,y=X2))+
  geom_line(aes(x=X1,y=X3))+
  xlab(expression(lambda))+
  ylab('')+
  theme_bw()+
  geom_hline(yintercept=0,lty=2) +
  annotate(geom='text',x=2,y=.60,label=expression(beta[2]))+
  annotate(geom='text',x=2,y=-.4,label=expression(beta[1]))
```



`glmnet()` function Similar to `lm` function, we can use `glmnet()` function from the `glmnet` package to run a regression model with ridge penalty. There are many arguments of the `glmnet()` function. For now, the arguments we need to know are

- `x`: an N x P input matrix, where N is the number of observations and P is the number of predictor
- `y`: an N x 1 input matrix for the outcome variable
- `alpha`: a mixing constant for lasso and ridge penalty. When it is zero, the ridge regression is conducted
- `lambda`: penalty term
- `intercept`: set FALSE to avoid intercept for standardized variables

If you want to fit the linear regression without any regularization, you can specify `alpha = 0` and `lambda = 0`.

```r
#install.packages('glmnet')

require(glmnet)

Y <-  as.matrix(readability_sub$target)
X <-  as.matrix(cbind(readability_sub$mean.wl,readability_sub$sents))
Y <- scale(Y)
X <- scale(X)

mod <- glmnet(x = X,
              y = Y,
              family = 'gaussian',
              alpha = 0,
              lambda = 0,
              intercept=FALSE)


coef(mod)
```

```
3 x 1 sparse Matrix of class "dgCMatrix"
                    s0
(Intercept)  .
V1          -0.3666327
V2           0.6049359
```

We can also increase the penalty term (λ).

```r
#install.packages('glmnet')

require(glmnet)

Y <-  as.matrix(readability_sub$target)
X <-  as.matrix(cbind(readability_sub$mean.wl,readability_sub$sents))
Y <- scale(Y)
X <- scale(X)

mod <- glmnet(x = X,
              y = Y,
              family = 'gaussian',
              alpha = 0,
              lambda = 0.5,
              intercept=FALSE)


coef(mod)
```

```
3 x 1 sparse Matrix of class "dgCMatrix"
                    s0
(Intercept)  .
V1          -0.2720458
V2           0.4145987
```

NOTE

A careful eye should catch the fact that the coefficient estimates we obtained from `glmnet()` function for the the two standardized variables (average word length and number of sentences) are different when the penalty term (λ) is 0.5. When we apply the matrix solution above for the ridge regression, we obtained the estimates of -0.360 and 0.591 for the two predictors, respectively, at $\lambda = 0.5$. When we enter the same value in `glmnet()`, we obtained the estimates of -0.27 and 0.414. So, what is wrong? Where does this discrepancy come from?

In fact, there is nothing wrong. It appears that what `lambda` argument in `glmnet` indicates is $\frac{\lambda}{N}$. In most statistics textbook, the penalty term for the ridge regression is specified as

$$\lambda \sum_{i=1}^{P} \beta_p^2.$$

On the other hand, if we examine Equation 1-3 in this paper written by the developers of the `glmnet` package, we can see that the penalty term applied is equivalent of

$$\lambda N \sum_{i=1}^{P} \beta_p^2.$$

Therefore, if we want to get the identical results, then we should use $\lambda = 0.5/20$.

```
N = 20

mod <- glmnet(x = X,
              y = Y,
              family = 'gaussian',
              alpha = 0,
              lambda = 0.5/N,
              intercept=FALSE)


coef(mod)
```

```
3 x 1 sparse Matrix of class "dgCMatrix"
                    s0
(Intercept)  .
V1          -0.3606303
V2           0.5911903
```

Note that these numbers are still slightly different. We can attribute this difference to numerical approximation `glmnet` is using when optimizing the loss function. `glmnet` doesn't use the closed form matrix solution for ridge regression. This is a good thing because there is not always a closed form solution for different types of regularization approaches (e.g., lasso). Therefore, the computational approximation in `glmnet` is very needed moving forward.

Tuning the Hyperparameter λ The λ parameter in ridge regression is called a **hyperparameter**. In the context of machine learning, the parameters in a model can be classified into two types: parameters and hyperparameters. The parameters of the model are typically estimated from data and not set by users. In

the context of ridge regression, regression coefficients, $\{\beta_0, \beta_1, ..., \beta_P\}$, are parameters to be estimated from data. On the other hand, the hyperparameters are not estimable, most of the time due to the fact that there is no first order or second order derivatives for these hyperparameters. Therefore, they must be set by the users. In the context of ridge regression, penalty term, $\{\lambda\}$, is a hyperparameter.

The process of deciding what value to use for a hyperparameter is called **tuning**, and it is most of the time a trial-error process. The idea is simple. We try many different values of a hyperparameter and check how well the model performs based on a certain criteria (e.g., MAE, MSE, RMSE) using a k-fold cross validation. Then, we pick the value of a hyperparameter that provides the best performance.

Using Ridge Regression to Predict Readability Scores

In this section, we will apply ridge regression to predict the readability scores from all predictors in the dataset. We will use the `caret` package and use 10-fold cross validation to evaluate the model performance for different levels of penalty term (λ).

```
# Load the packages

  require(caret)
  require(recipes)
  require(finalfit)
  require(glmnet)

# Import the dataset

  readability <- read.csv('https://raw.githubusercontent.com/uo-datasci-specialization/c4-ml-fall-2021/m

# Initial preparation (remove variables with large amount of missingness)

  require(finalfit)

  missing_ <- ff_glimpse(readability)$Continuous
  flag_na <- which(as.numeric(missing_$missing_percent) > 80)
  readability <- readability[,-flag_na]

# Set the random seed for reproducibility

  set.seed(10152021)

# Train/Test Split

  loc      <- sample(1:nrow(readability), round(nrow(readability) * 0.9))
  read_tr  <- readability[loc, ]
  read_te  <- readability[-loc, ]

# Blueprint

  blueprint <- recipe(x     = readability,
                      vars  = colnames(readability),
                      roles = c(rep('predictor',990),'outcome')) %>%
    step_zv(all_numeric()) %>%
    step_nzv(all_numeric()) %>%
    step_impute_mean(all_numeric()) %>%
    step_normalize(all_numeric_predictors()) %>%
```

```
    step_corr(all_numeric(),threshold=0.9)

# Cross validation settings

    # Randomly shuffle the data

      read_tr = read_tr[sample(nrow(read_tr)),]

    # Create 10 folds with equal size

      folds = cut(seq(1,nrow(read_tr)),breaks=10,labels=FALSE)

    # Create the list for each fold

      my.indices <- vector('list',10)
      for(i in 1:10){
        my.indices[[i]] <- which(folds!=i)
      }

  cv <- trainControl(method = "cv",
                     index  = my.indices)

# Tune Grid

  # Here, we have to specify different values of lambda we want to try
  # This should be a dataframe with columns named are the same as
  # the tuning parameters available for the engine we are using

  # In order to get which parameters are available to tune for glmnet
  # run the following code

  caret::getModelInfo()$glmnet$parameters

  # This indicates there are two hyperparameters available to tune for the glmnet
  # For ridge regression, we will fix the value of alpha to 0
  # For lambda, we will consider all values from 0.01 to 3 with increments of 0.01


    # Remember how glmnet multiplies the lambda by sample size (N)
    # In this case, the sample size is 2834
    # So, for instance a lambda value of 1 would be 2834
    # You can try larger values and explore, but in this case a max value of 3
    # for lambda would be more than enough. I don't think it will improve performance
    # beyond this value

    # Also, note that there are 100 values, and for every lambda value we will do
    # 10-fold cross validation, so it can take a very long time to search this
    # grid

  grid <- data.frame(alpha = 0, lambda = seq(0.01,3,.01))
  grid
```

```r
# Train the model

  ridge <- caret::train(blueprint,
                        data      = read_tr,
                        method    = "glmnet",
                        trControl = cv,
                        tuneGrid  = grid)

    # This training took about 3 minutes in my computer

  ridge$results

  ridge$bestTune

  plot(ridge)
```

```
    alpha lambda      RMSE  Rsquared       MAE     RMSESD RsquaredSD      MAESD
1       0   0.01 0.5693556 0.7026113 0.4536980 0.03339565 0.02717370 0.02661641
2       0   0.02 0.5693556 0.7026113 0.4536980 0.03339565 0.02717370 0.02661641
3       0   0.03 0.5693556 0.7026113 0.4536980 0.03339565 0.02717370 0.02661641
4       0   0.04 0.5693556 0.7026113 0.4536980 0.03339565 0.02717370 0.02661641
5       0   0.05 0.5693556 0.7026113 0.4536980 0.03339565 0.02717370 0.02661641
6       0   0.06 0.5688754 0.7030377 0.4533584 0.03330730 0.02717044 0.02650685
7       0   0.07 0.5657611 0.7058185 0.4511540 0.03272822 0.02679653 0.02581546
8       0   0.08 0.5631828 0.7081239 0.4492663 0.03226788 0.02649567 0.02522399
9       0   0.09 0.5610528 0.7100320 0.4477183 0.03187503 0.02624041 0.02470547
10      0   0.10 0.5591914 0.7117015 0.4463558 0.03153475 0.02602184 0.02424918
11      0   0.11 0.5576213 0.7131114 0.4451710 0.03127429 0.02584887 0.02387395
12      0   0.12 0.5562448 0.7143491 0.4441044 0.03101785 0.02567019 0.02353168
13      0   0.13 0.5550365 0.7154358 0.4431402 0.03080676 0.02552802 0.02322289
14      0   0.14 0.5539643 0.7164019 0.4422796 0.03062323 0.02540385 0.02294966
15      0   0.15 0.5530031 0.7172711 0.4415044 0.03044362 0.02527209 0.02268268
16      0   0.16 0.5521598 0.7180311 0.4408316 0.03031078 0.02518412 0.02244827
17      0   0.17 0.5513942 0.7187235 0.4402053 0.03018070 0.02509060 0.02223840
18      0   0.18 0.5506878 0.7193647 0.4396279 0.03005825 0.02499882 0.02205255
19      0   0.19 0.5500690 0.7199261 0.4391323 0.02995488 0.02491941 0.02190026
20      0   0.20 0.5494992 0.7204441 0.4386782 0.02985668 0.02484288 0.02176385
21      0   0.21 0.5489883 0.7209085 0.4382769 0.02976456 0.02477502 0.02164215
22      0   0.22 0.5485159 0.7213392 0.4379251 0.02968047 0.02471071 0.02153178
23      0   0.23 0.5480744 0.7217443 0.4376007 0.02961214 0.02465235 0.02143595
24      0   0.24 0.5476653 0.7221207 0.4372942 0.02954832 0.02459480 0.02134776
25      0   0.25 0.5473050 0.7224502 0.4370215 0.02948976 0.02455136 0.02125709
26      0   0.26 0.5469657 0.7227614 0.4367622 0.02943205 0.02450823 0.02117104
27      0   0.27 0.5466651 0.7230391 0.4365273 0.02937885 0.02446151 0.02109564
28      0   0.28 0.5463768 0.7233070 0.4363168 0.02932953 0.02441624 0.02103216
29      0   0.29 0.5461136 0.7235523 0.4361310 0.02928742 0.02437426 0.02099181
30      0   0.30 0.5458758 0.7237734 0.4359530 0.02925059 0.02434132 0.02095470
31      0   0.31 0.5456496 0.7239846 0.4357794 0.02921446 0.02430940 0.02091412
32      0   0.32 0.5454469 0.7241748 0.4356244 0.02918048 0.02427726 0.02087462
33      0   0.33 0.5452573 0.7243537 0.4354800 0.02915090 0.02424848 0.02083861
34      0   0.34 0.5450774 0.7245243 0.4353450 0.02912206 0.02422045 0.02080469
35      0   0.35 0.5449195 0.7246752 0.4352259 0.02909655 0.02419370 0.02077949
36      0   0.36 0.5447745 0.7248145 0.4351171 0.02907454 0.02417087 0.02075610
```

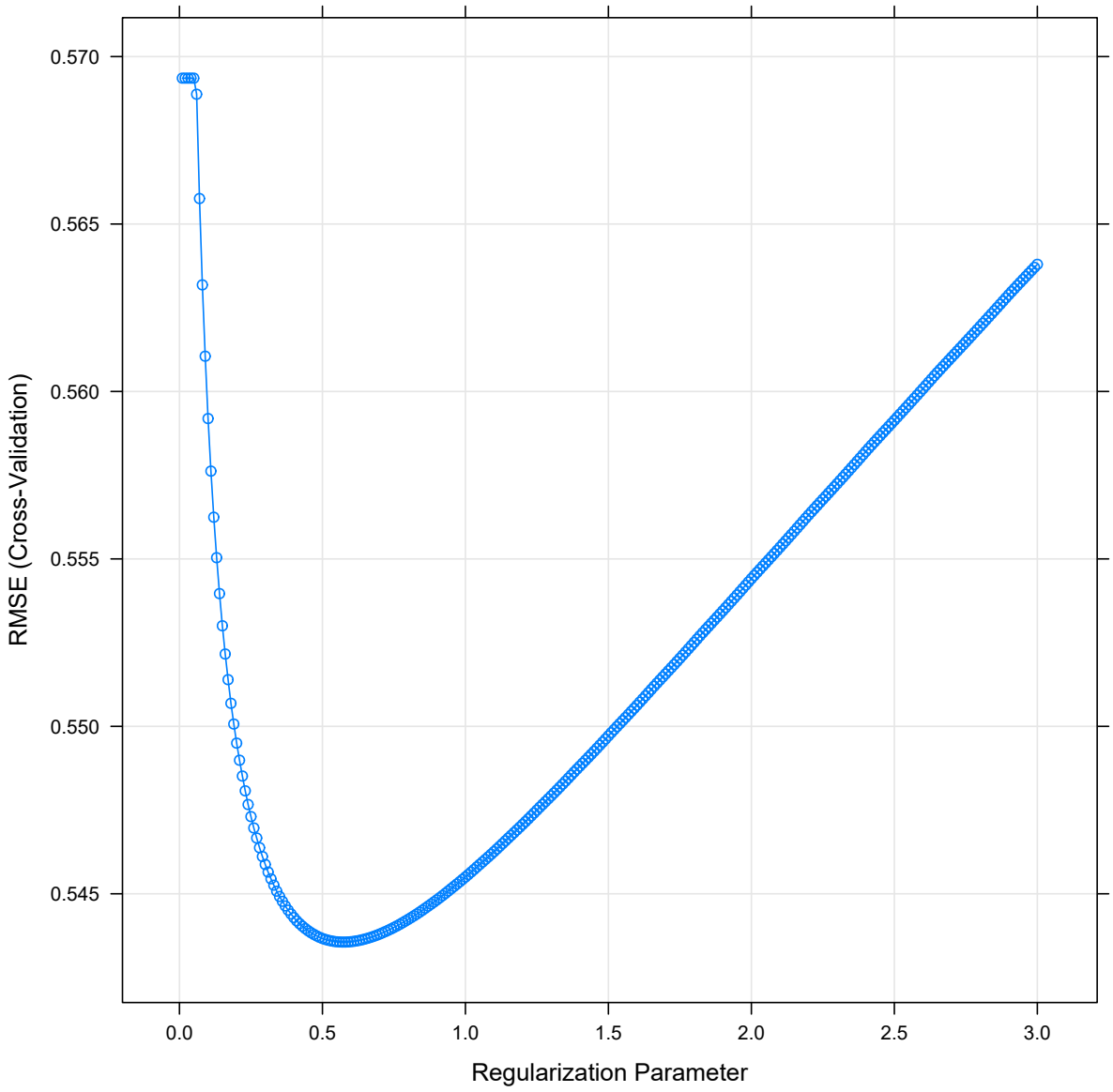
37	0	0.37	0.5446375	0.7249470	0.4350127	0.02905305	0.02414866	0.02073096
38	0	0.38	0.5445115	0.7250697	0.4349155	0.02903249	0.02412638	0.02070213
39	0	0.39	0.5443980	0.7251812	0.4348299	0.02901527	0.02410720	0.02067546
40	0	0.40	0.5442914	0.7252866	0.4347481	0.02899893	0.02408895	0.02065331
41	0	0.41	0.5441915	0.7253863	0.4346723	0.02898299	0.02407118	0.02063458
42	0	0.42	0.5441071	0.7254719	0.4346090	0.02896830	0.02405263	0.02062142
43	0	0.43	0.5440311	0.7255499	0.4345480	0.02895637	0.02403690	0.02061039
44	0	0.44	0.5439607	0.7256231	0.4344909	0.02894471	0.02402151	0.02059457
45	0	0.45	0.5438959	0.7256915	0.4344438	0.02893333	0.02400647	0.02058428
46	0	0.46	0.5438386	0.7257535	0.4344026	0.02892312	0.02399296	0.02057695
47	0	0.47	0.5437877	0.7258097	0.4343669	0.02891530	0.02398191	0.02057203
48	0	0.48	0.5437415	0.7258618	0.4343355	0.02890783	0.02397123	0.02056864
49	0	0.49	0.5437000	0.7259100	0.4343060	0.02890058	0.02396082	0.02056634
50	0	0.50	0.5436649	0.7259524	0.4342776	0.02889361	0.02395026	0.02056550
51	0	0.51	0.5436373	0.7259880	0.4342523	0.02888833	0.02394128	0.02056859
52	0	0.52	0.5436137	0.7260202	0.4342297	0.02888407	0.02393313	0.02057354
53	0	0.53	0.5435940	0.7260489	0.4342092	0.02887997	0.02392517	0.02057901
54	0	0.54	0.5435783	0.7260743	0.4341918	0.02887603	0.02391741	0.02058883
55	0	0.55	0.5435673	0.7260955	0.4341755	0.02887237	0.02390992	0.02059964
56	0	0.56	0.5435613	0.7261122	0.4341640	0.02887024	0.02390415	0.02061132
57	0	0.57	0.5435584	0.7261262	0.4341562	0.02886885	0.02389883	0.02062659
58	0	0.58	0.5435588	0.7261373	0.4341516	0.02886758	0.02389367	0.02064124
59	0	0.59	0.5435626	0.7261456	0.4341511	0.02886644	0.02388865	0.02065391
60	0	0.60	0.5435698	0.7261509	0.4341520	0.02886545	0.02388397	0.02066757
61	0	0.61	0.5435807	0.7261530	0.4341551	0.02886510	0.02388023	0.02068117
62	0	0.62	0.5435943	0.7261526	0.4341627	0.02886606	0.02387738	0.02069601
63	0	0.63	0.5436107	0.7261498	0.4341741	0.02886711	0.02387464	0.02071149
64	0	0.64	0.5436298	0.7261447	0.4341867	0.02886826	0.02387201	0.02072773
65	0	0.65	0.5436518	0.7261371	0.4342002	0.02886950	0.02386949	0.02074362
66	0	0.66	0.5436762	0.7261274	0.4342133	0.02887083	0.02386744	0.02075992
67	0	0.67	0.5437032	0.7261155	0.4342260	0.02887276	0.02386633	0.02077620
68	0	0.68	0.5437321	0.7261019	0.4342400	0.02887570	0.02386571	0.02079339
69	0	0.69	0.5437634	0.7260862	0.4342598	0.02887872	0.02386517	0.02081031
70	0	0.70	0.5437971	0.7260685	0.4342826	0.02888182	0.02386472	0.02082774
71	0	0.71	0.5438331	0.7260487	0.4343094	0.02888499	0.02386434	0.02084767
72	0	0.72	0.5438713	0.7260271	0.4343398	0.02888827	0.02386435	0.02086836
73	0	0.73	0.5439106	0.7260045	0.4343688	0.02889156	0.02386495	0.02088825
74	0	0.74	0.5439517	0.7259804	0.4343977	0.02889577	0.02386622	0.02090889
75	0	0.75	0.5439945	0.7259548	0.4344294	0.02890037	0.02386758	0.02092863
76	0	0.76	0.5440393	0.7259275	0.4344629	0.02890503	0.02386900	0.02094759
77	0	0.77	0.5440861	0.7258984	0.4344973	0.02890974	0.02387048	0.02096660
78	0	0.78	0.5441350	0.7258677	0.4345327	0.02891452	0.02387203	0.02098752
79	0	0.79	0.5441855	0.7258354	0.4345680	0.02891939	0.02387398	0.02100869
80	0	0.80	0.5442362	0.7258031	0.4346016	0.02892412	0.02387650	0.02102883
81	0	0.81	0.5442883	0.7257696	0.4346363	0.02892955	0.02387951	0.02104877
82	0	0.82	0.5443416	0.7257350	0.4346708	0.02893544	0.02388255	0.02106931
83	0	0.83	0.5443966	0.7256990	0.4347063	0.02894138	0.02388564	0.02108971
84	0	0.84	0.5444533	0.7256614	0.4347426	0.02894737	0.02388877	0.02111002
85	0	0.85	0.5445118	0.7256224	0.4347799	0.02895340	0.02389195	0.02113014
86	0	0.86	0.5445719	0.7255820	0.4348172	0.02895948	0.02389518	0.02115040
87	0	0.87	0.5446326	0.7255410	0.4348536	0.02896557	0.02389913	0.02117067
88	0	0.88	0.5446933	0.7255003	0.4348893	0.02897155	0.02390345	0.02118976
89	0	0.89	0.5447548	0.7254588	0.4349253	0.02897817	0.02390797	0.02120854
90	0	0.90	0.5448174	0.7254163	0.4349614	0.02898509	0.02391246	0.02122809

91	0	0.91	0.5448815	0.7253725	0.4349989	0.02899205	0.02391700	0.02124794
92	0	0.92	0.5449470	0.7253276	0.4350370	0.02899906	0.02392157	0.02126840
93	0	0.93	0.5450139	0.7252813	0.4350752	0.02900610	0.02392617	0.02128898
94	0	0.94	0.5450824	0.7252339	0.4351146	0.02901318	0.02393081	0.02130943
95	0	0.95	0.5451520	0.7251853	0.4351540	0.02902036	0.02393570	0.02133000
96	0	0.96	0.5452204	0.7251379	0.4351922	0.02902718	0.02394143	0.02134943
97	0	0.97	0.5452896	0.7250901	0.4352325	0.02903418	0.02394735	0.02137023
98	0	0.98	0.5453589	0.7250419	0.4352737	0.02904187	0.02395306	0.02139158
99	0	0.99	0.5454295	0.7249927	0.4353154	0.02904959	0.02395880	0.02141317
100	0	1.00	0.5455014	0.7249425	0.4353580	0.02905734	0.02396457	0.02143623
101	0	1.01	0.5455744	0.7248911	0.4354019	0.02906513	0.02397036	0.02145984
102	0	1.02	0.5456487	0.7248387	0.4354475	0.02907295	0.02397618	0.02148387
103	0	1.03	0.5457243	0.7247853	0.4354940	0.02908080	0.02398203	0.02150808
104	0	1.04	0.5458011	0.7247307	0.4355436	0.02908868	0.02398791	0.02153219
105	0	1.05	0.5458766	0.7246773	0.4355910	0.02909639	0.02399493	0.02155541
106	0	1.06	0.5459519	0.7246243	0.4356380	0.02910396	0.02400189	0.02157829
107	0	1.07	0.5460275	0.7245710	0.4356850	0.02911194	0.02400875	0.02160175
108	0	1.08	0.5461038	0.7245171	0.4357320	0.02912022	0.02401550	0.02162560
109	0	1.09	0.5461810	0.7244623	0.4357793	0.02912852	0.02402228	0.02164957
110	0	1.10	0.5462594	0.7244067	0.4358266	0.02913684	0.02402907	0.02167362
111	0	1.11	0.5463388	0.7243501	0.4358740	0.02914520	0.02403589	0.02169775
112	0	1.12	0.5464192	0.7242926	0.4359214	0.02915358	0.02404272	0.02172195
113	0	1.13	0.5465007	0.7242342	0.4359687	0.02916199	0.02404958	0.02174623
114	0	1.14	0.5465833	0.7241748	0.4360166	0.02917043	0.02405646	0.02177067
115	0	1.15	0.5466648	0.7241164	0.4360629	0.02917875	0.02406442	0.02179445
116	0	1.16	0.5467453	0.7240590	0.4361076	0.02918676	0.02407230	0.02181758
117	0	1.17	0.5468263	0.7240014	0.4361527	0.02919492	0.02408017	0.02184048
118	0	1.18	0.5469071	0.7239438	0.4361982	0.02920356	0.02408775	0.02186425
119	0	1.19	0.5469889	0.7238853	0.4362442	0.02921223	0.02409534	0.02188819
120	0	1.20	0.5470716	0.7238261	0.4362917	0.02922092	0.02410294	0.02191181
121	0	1.21	0.5471552	0.7237661	0.4363401	0.02922963	0.02411057	0.02193496
122	0	1.22	0.5472397	0.7237054	0.4363884	0.02923836	0.02411821	0.02195814
123	0	1.23	0.5473251	0.7236438	0.4364369	0.02924712	0.02412587	0.02198132
124	0	1.24	0.5474114	0.7235815	0.4364864	0.02925590	0.02413355	0.02200434
125	0	1.25	0.5474986	0.7235183	0.4365365	0.02926470	0.02414124	0.02202733
126	0	1.26	0.5475851	0.7234557	0.4365868	0.02927346	0.02414993	0.02204976
127	0	1.27	0.5476699	0.7233949	0.4366356	0.02928176	0.02415863	0.02207119
128	0	1.28	0.5477549	0.7233340	0.4366878	0.02929005	0.02416727	0.02208985
129	0	1.29	0.5478397	0.7232732	0.4367405	0.02929879	0.02417557	0.02210897
130	0	1.30	0.5479251	0.7232119	0.4367938	0.02930766	0.02418381	0.02212915
131	0	1.31	0.5480112	0.7231500	0.4368476	0.02931654	0.02419205	0.02214943
132	0	1.32	0.5480981	0.7230874	0.4369017	0.02932544	0.02420031	0.02216982
133	0	1.33	0.5481858	0.7230241	0.4369560	0.02933435	0.02420859	0.02219026
134	0	1.34	0.5482743	0.7229601	0.4370106	0.02934329	0.02421688	0.02221095
135	0	1.35	0.5483635	0.7228955	0.4370651	0.02935225	0.02422518	0.02223170
136	0	1.36	0.5484534	0.7228302	0.4371197	0.02936123	0.02423350	0.02225248
137	0	1.37	0.5485442	0.7227642	0.4371743	0.02937023	0.02424183	0.02227332
138	0	1.38	0.5486349	0.7226983	0.4372290	0.02937927	0.02425080	0.02229394
139	0	1.39	0.5487232	0.7226346	0.4372830	0.02938774	0.02426029	0.02231389
140	0	1.40	0.5488113	0.7225712	0.4373365	0.02939616	0.02426953	0.02233362
141	0	1.41	0.5488995	0.7225078	0.4373905	0.02940472	0.02427856	0.02235391
142	0	1.42	0.5489876	0.7224444	0.4374439	0.02941362	0.02428727	0.02237486
143	0	1.43	0.5490763	0.7223805	0.4374973	0.02942255	0.02429599	0.02239585
144	0	1.44	0.5491656	0.7223161	0.4375511	0.02943148	0.02430472	0.02241695

145	0	1.45	0.5492556	0.7222510	0.4376059	0.02944044	0.02431346	0.02243795
146	0	1.46	0.5493463	0.7221854	0.4376611	0.02944941	0.02432222	0.02245887
147	0	1.47	0.5494377	0.7221192	0.4377166	0.02945840	0.02433098	0.02247985
148	0	1.48	0.5495297	0.7220524	0.4377722	0.02946741	0.02433976	0.02250089
149	0	1.49	0.5496223	0.7219850	0.4378277	0.02947643	0.02434855	0.02252196
150	0	1.50	0.5497156	0.7219170	0.4378841	0.02948547	0.02435734	0.02254169
151	0	1.51	0.5498096	0.7218485	0.4379425	0.02949452	0.02436615	0.02255895
152	0	1.52	0.5499012	0.7217821	0.4379990	0.02950331	0.02437639	0.02257534
153	0	1.53	0.5499918	0.7217167	0.4380543	0.02951174	0.02438618	0.02259142
154	0	1.54	0.5500824	0.7216515	0.4381097	0.02952009	0.02439569	0.02260754
155	0	1.55	0.5501728	0.7215864	0.4381661	0.02952873	0.02440494	0.02262394
156	0	1.56	0.5502632	0.7215212	0.4382237	0.02953756	0.02441398	0.02264096
157	0	1.57	0.5503542	0.7214556	0.4382815	0.02954640	0.02442302	0.02265780
158	0	1.58	0.5504458	0.7213895	0.4383394	0.02955526	0.02443208	0.02267468
159	0	1.59	0.5505379	0.7213228	0.4383973	0.02956413	0.02444115	0.02269158
160	0	1.60	0.5506306	0.7212557	0.4384552	0.02957302	0.02445022	0.02270854
161	0	1.61	0.5507239	0.7211881	0.4385136	0.02958192	0.02445931	0.02272545
162	0	1.62	0.5508177	0.7211200	0.4385724	0.02959084	0.02446840	0.02274162
163	0	1.63	0.5509121	0.7210514	0.4386312	0.02959976	0.02447751	0.02275774
164	0	1.64	0.5510071	0.7209822	0.4386904	0.02960871	0.02448662	0.02277405
165	0	1.65	0.5511026	0.7209126	0.4387500	0.02961766	0.02449575	0.02279063
166	0	1.66	0.5511982	0.7208429	0.4388095	0.02962671	0.02450524	0.02280716
167	0	1.67	0.5512911	0.7207756	0.4388674	0.02963514	0.02451572	0.02282251
168	0	1.68	0.5513835	0.7207090	0.4389244	0.02964344	0.02452578	0.02283768
169	0	1.69	0.5514757	0.7206426	0.4389810	0.02965162	0.02453551	0.02285234
170	0	1.70	0.5515677	0.7205764	0.4390375	0.02966005	0.02454498	0.02286739
171	0	1.71	0.5516596	0.7205103	0.4390941	0.02966868	0.02455420	0.02288320
172	0	1.72	0.5517520	0.7204437	0.4391507	0.02967733	0.02456343	0.02289902
173	0	1.73	0.5518449	0.7203767	0.4392076	0.02968599	0.02457267	0.02291482
174	0	1.74	0.5519382	0.7203093	0.4392644	0.02969465	0.02458192	0.02293064
175	0	1.75	0.5520320	0.7202414	0.4393213	0.02970334	0.02459118	0.02294650
176	0	1.76	0.5521264	0.7201731	0.4393790	0.02971203	0.02460044	0.02296429
177	0	1.77	0.5522212	0.7201044	0.4394382	0.02972073	0.02460971	0.02298417
178	0	1.78	0.5523165	0.7200353	0.4394979	0.02972945	0.02461899	0.02300419
179	0	1.79	0.5524122	0.7199657	0.4395585	0.02973818	0.02462828	0.02302573
180	0	1.80	0.5525085	0.7198957	0.4396195	0.02974692	0.02463757	0.02304779
181	0	1.81	0.5526053	0.7198253	0.4396805	0.02975567	0.02464688	0.02306987
182	0	1.82	0.5527024	0.7197544	0.4397420	0.02976451	0.02465626	0.02309284
183	0	1.83	0.5527969	0.7196861	0.4398014	0.02977285	0.02466709	0.02311464
184	0	1.84	0.5528903	0.7196188	0.4398606	0.02978088	0.02467732	0.02313565
185	0	1.85	0.5529834	0.7195518	0.4399202	0.02978885	0.02468725	0.02315571
186	0	1.86	0.5530768	0.7194846	0.4399805	0.02979677	0.02469706	0.02317472
187	0	1.87	0.5531694	0.7194182	0.4400404	0.02980511	0.02470634	0.02319417
188	0	1.88	0.5532623	0.7193514	0.4401002	0.02981346	0.02471563	0.02321364
189	0	1.89	0.5533557	0.7192843	0.4401607	0.02982181	0.02472492	0.02323319
190	0	1.90	0.5534495	0.7192167	0.4402211	0.02983017	0.02473422	0.02325276
191	0	1.91	0.5535437	0.7191488	0.4402817	0.02983854	0.02474353	0.02327237
192	0	1.92	0.5536383	0.7190806	0.4403430	0.02984692	0.02475284	0.02329207
193	0	1.93	0.5537333	0.7190119	0.4404043	0.02985531	0.02476216	0.02331178
194	0	1.94	0.5538288	0.7189429	0.4404657	0.02986371	0.02477149	0.02333152
195	0	1.95	0.5539247	0.7188735	0.4405277	0.02987213	0.02478082	0.02335079
196	0	1.96	0.5540209	0.7188038	0.4405900	0.02988055	0.02479016	0.02337061
197	0	1.97	0.5541176	0.7187336	0.4406530	0.02988898	0.02479951	0.02339094
198	0	1.98	0.5542148	0.7186631	0.4407168	0.02989742	0.02480886	0.02341136

199	0	1.99	0.5543123	0.7185922	0.4407811	0.02990587	0.02481822	0.02343152
200	0	2.00	0.5544095	0.7185216	0.4408449	0.02991438	0.02482809	0.02345154
201	0	2.01	0.5545042	0.7184534	0.4409070	0.02992224	0.02483880	0.02347028
202	0	2.02	0.5545981	0.7183859	0.4409686	0.02992994	0.02484905	0.02348880
203	0	2.03	0.5546916	0.7183189	0.4410301	0.02993756	0.02485893	0.02350694
204	0	2.04	0.5547853	0.7182516	0.4410920	0.02994512	0.02486870	0.02352502
205	0	2.05	0.5548783	0.7181851	0.4411541	0.02995303	0.02487800	0.02354405
206	0	2.06	0.5549715	0.7181184	0.4412174	0.02996099	0.02488724	0.02356299
207	0	2.07	0.5550650	0.7180513	0.4412822	0.02996896	0.02489648	0.02358241
208	0	2.08	0.5551589	0.7179840	0.4413477	0.02997693	0.02490572	0.02360271
209	0	2.09	0.5552532	0.7179163	0.4414140	0.02998492	0.02491497	0.02362283
210	0	2.10	0.5553478	0.7178483	0.4414806	0.02999291	0.02492422	0.02364268
211	0	2.11	0.5554428	0.7177800	0.4415482	0.03000091	0.02493348	0.02366226
212	0	2.12	0.5555381	0.7177114	0.4416168	0.03000891	0.02494275	0.02368227
213	0	2.13	0.5556337	0.7176424	0.4416853	0.03001693	0.02495202	0.02370228
214	0	2.14	0.5557298	0.7175731	0.4417538	0.03002496	0.02496130	0.02372231
215	0	2.15	0.5558262	0.7175035	0.4418223	0.03003299	0.02497058	0.02374236
216	0	2.16	0.5559229	0.7174336	0.4418909	0.03004103	0.02497986	0.02376242
217	0	2.17	0.5560200	0.7173634	0.4419597	0.03004908	0.02498916	0.02378254
218	0	2.18	0.5561175	0.7172928	0.4420289	0.03005714	0.02499845	0.02380259
219	0	2.19	0.5562153	0.7172219	0.4420985	0.03006520	0.02500776	0.02382275
220	0	2.20	0.5563113	0.7171526	0.4421675	0.03007311	0.02501845	0.02384273
221	0	2.21	0.5564052	0.7170854	0.4422352	0.03008039	0.02502869	0.02386110
222	0	2.22	0.5564990	0.7170183	0.4423027	0.03008769	0.02503883	0.02387950
223	0	2.23	0.5565923	0.7169517	0.4423700	0.03009482	0.02504843	0.02389765
224	0	2.24	0.5566860	0.7168848	0.4424377	0.03010196	0.02505805	0.02391551
225	0	2.25	0.5567788	0.7168187	0.4425047	0.03010946	0.02506716	0.02393404
226	0	2.26	0.5568718	0.7167524	0.4425715	0.03011697	0.02507624	0.02395262
227	0	2.27	0.5569651	0.7166858	0.4426384	0.03012448	0.02508532	0.02397121
228	0	2.28	0.5570587	0.7166190	0.4427055	0.03013201	0.02509441	0.02398939
229	0	2.29	0.5571526	0.7165519	0.4427734	0.03013954	0.02510351	0.02400629
230	0	2.30	0.5572468	0.7164845	0.4428412	0.03014707	0.02511261	0.02402321
231	0	2.31	0.5573414	0.7164168	0.4429091	0.03015462	0.02512171	0.02404014
232	0	2.32	0.5574362	0.7163489	0.4429770	0.03016217	0.02513082	0.02405726
233	0	2.33	0.5575313	0.7162806	0.4430455	0.03016973	0.02513993	0.02407486
234	0	2.34	0.5576268	0.7162121	0.4431139	0.03017729	0.02514905	0.02409246
235	0	2.35	0.5577225	0.7161433	0.4431827	0.03018486	0.02515817	0.02410944
236	0	2.36	0.5578186	0.7160743	0.4432516	0.03019244	0.02516729	0.02412624
237	0	2.37	0.5579150	0.7160049	0.4433206	0.03020003	0.02517642	0.02414306
238	0	2.38	0.5580116	0.7159353	0.4433897	0.03020762	0.02518556	0.02416000
239	0	2.39	0.5581086	0.7158654	0.4434589	0.03021522	0.02519470	0.02417702
240	0	2.40	0.5582059	0.7157952	0.4435282	0.03022282	0.02520384	0.02419406
241	0	2.41	0.5583024	0.7157257	0.4435966	0.03023044	0.02521372	0.02421091
242	0	2.42	0.5583966	0.7156584	0.4436640	0.03023740	0.02522416	0.02422575
243	0	2.43	0.5584899	0.7155920	0.4437306	0.03024419	0.02523412	0.02424038
244	0	2.44	0.5585830	0.7155258	0.4437972	0.03025096	0.02524384	0.02425436
245	0	2.45	0.5586758	0.7154599	0.4438637	0.03025761	0.02525321	0.02426801
246	0	2.46	0.5587689	0.7153938	0.4439304	0.03026432	0.02526253	0.02428176
247	0	2.47	0.5588610	0.7153285	0.4439963	0.03027132	0.02527138	0.02429621
248	0	2.48	0.5589534	0.7152630	0.4440622	0.03027833	0.02528022	0.02431066
249	0	2.49	0.5590460	0.7151973	0.4441287	0.03028534	0.02528907	0.02432468
250	0	2.50	0.5591390	0.7151313	0.4441961	0.03029236	0.02529792	0.02433754
251	0	2.51	0.5592321	0.7150651	0.4442640	0.03029939	0.02530678	0.02435019
252	0	2.52	0.5593256	0.7149987	0.4443320	0.03030642	0.02531564	0.02436278

```
253     0   2.53 0.5594193 0.7149320 0.4444002 0.03031345 0.02532450 0.02437527
254     0   2.54 0.5595133 0.7148651 0.4444685 0.03032050 0.02533337 0.02438756
255     0   2.55 0.5596075 0.7147979 0.4445371 0.03032754 0.02534224 0.02439973
256     0   2.56 0.5597020 0.7147305 0.4446069 0.03033460 0.02535111 0.02441324
257     0   2.57 0.5597968 0.7146628 0.4446768 0.03034166 0.02535999 0.02442700
258     0   2.58 0.5598918 0.7145949 0.4447468 0.03034872 0.02536887 0.02444056
259     0   2.59 0.5599871 0.7145268 0.4448171 0.03035579 0.02537775 0.02445360
260     0   2.60 0.5600826 0.7144584 0.4448875 0.03036287 0.02538664 0.02446647
261     0   2.61 0.5601785 0.7143898 0.4449581 0.03036995 0.02539553 0.02447935
262     0   2.62 0.5602745 0.7143209 0.4450288 0.03037704 0.02540442 0.02449226
263     0   2.63 0.5603709 0.7142518 0.4450996 0.03038413 0.02541332 0.02450524
264     0   2.64 0.5604675 0.7141825 0.4451708 0.03039127 0.02542225 0.02451931
265     0   2.65 0.5605617 0.7141153 0.4452401 0.03039808 0.02543273 0.02453261
266     0   2.66 0.5606542 0.7140497 0.4453078 0.03040435 0.02544250 0.02454521
267     0   2.67 0.5607468 0.7139842 0.4453755 0.03041062 0.02545218 0.02455767
268     0   2.68 0.5608387 0.7139193 0.4454428 0.03041684 0.02546145 0.02456977
269     0   2.69 0.5609307 0.7138544 0.4455105 0.03042296 0.02547051 0.02458143
270     0   2.70 0.5610227 0.7137895 0.4455784 0.03042916 0.02547950 0.02459341
271     0   2.71 0.5611139 0.7137253 0.4456460 0.03043562 0.02548803 0.02460574
272     0   2.72 0.5612052 0.7136610 0.4457136 0.03044209 0.02549657 0.02461806
273     0   2.73 0.5612968 0.7135965 0.4457812 0.03044856 0.02550511 0.02463039
274     0   2.74 0.5613887 0.7135317 0.4458497 0.03045503 0.02551365 0.02464329
275     0   2.75 0.5614807 0.7134668 0.4459193 0.03046152 0.02552220 0.02465705
276     0   2.76 0.5615730 0.7134016 0.4459890 0.03046800 0.02553074 0.02467086
277     0   2.77 0.5616655 0.7133362 0.4460592 0.03047449 0.02553930 0.02468399
278     0   2.78 0.5617583 0.7132706 0.4461296 0.03048099 0.02554785 0.02469695
279     0   2.79 0.5618512 0.7132049 0.4462003 0.03048749 0.02555641 0.02471001
280     0   2.80 0.5619444 0.7131389 0.4462710 0.03049399 0.02556497 0.02472306
281     0   2.81 0.5620378 0.7130727 0.4463418 0.03050050 0.02557353 0.02473597
282     0   2.82 0.5621315 0.7130062 0.4464127 0.03050702 0.02558209 0.02474890
283     0   2.83 0.5622253 0.7129396 0.4464846 0.03051354 0.02559066 0.02476175
284     0   2.84 0.5623194 0.7128728 0.4465578 0.03052006 0.02559923 0.02477493
285     0   2.85 0.5624137 0.7128057 0.4466310 0.03052659 0.02560780 0.02478818
286     0   2.86 0.5625083 0.7127385 0.4467045 0.03053313 0.02561638 0.02480172
287     0   2.87 0.5626031 0.7126710 0.4467783 0.03053967 0.02562496 0.02481517
288     0   2.88 0.5626981 0.7126034 0.4468523 0.03054621 0.02563354 0.02482861
289     0   2.89 0.5627933 0.7125355 0.4469266 0.03055276 0.02564213 0.02484216
290     0   2.90 0.5628882 0.7124679 0.4470006 0.03055943 0.02565113 0.02485571
291     0   2.91 0.5629809 0.7124023 0.4470726 0.03056543 0.02566119 0.02486834
292     0   2.92 0.5630721 0.7123382 0.4471433 0.03057115 0.02567056 0.02488059
293     0   2.93 0.5631634 0.7122740 0.4472139 0.03057686 0.02567989 0.02489287
294     0   2.94 0.5632541 0.7122104 0.4472838 0.03058254 0.02568883 0.02490489
295     0   2.95 0.5633448 0.7121470 0.4473534 0.03058812 0.02569751 0.02491679
296     0   2.96 0.5634356 0.7120833 0.4474231 0.03059370 0.02570619 0.02492869
297     0   2.97 0.5635257 0.7120204 0.4474921 0.03059956 0.02571446 0.02494117
298     0   2.98 0.5636156 0.7119576 0.4475609 0.03060546 0.02572262 0.02495379
299     0   2.99 0.5637058 0.7118946 0.4476298 0.03061137 0.02573080 0.02496672
300     0   3.00 0.5637962 0.7118314 0.4476988 0.03061728 0.02573897 0.02497987

   alpha lambda
57     0   0.57
```



The 10-fold cross-validation results on the training dataset indicate that a λ value of 0.57 provides the best performance (minimum RMSE). Let's use this model to predict the outcome in the hold-out test dataset.

```
predict_te_ridge <- predict(ridge, read_te)

rsq_te <- cor(read_te$target,predict_te_ridge)^2
rsq_te
```

```
[1] 0.7274006
```

```
mae_te <- mean(abs(read_te$target - predict_te_ridge))
mae_te
```

```
[1] 0.4347177
```

```
rmse_te <- sqrt(mean((read_te$target - predict_te_ridge)^2))
rmse_te
```

```
[1] 0.5355444
```

Below is a table to compare the performance of ridge regression and linear regression (from earlier lecture) on the test dataset.

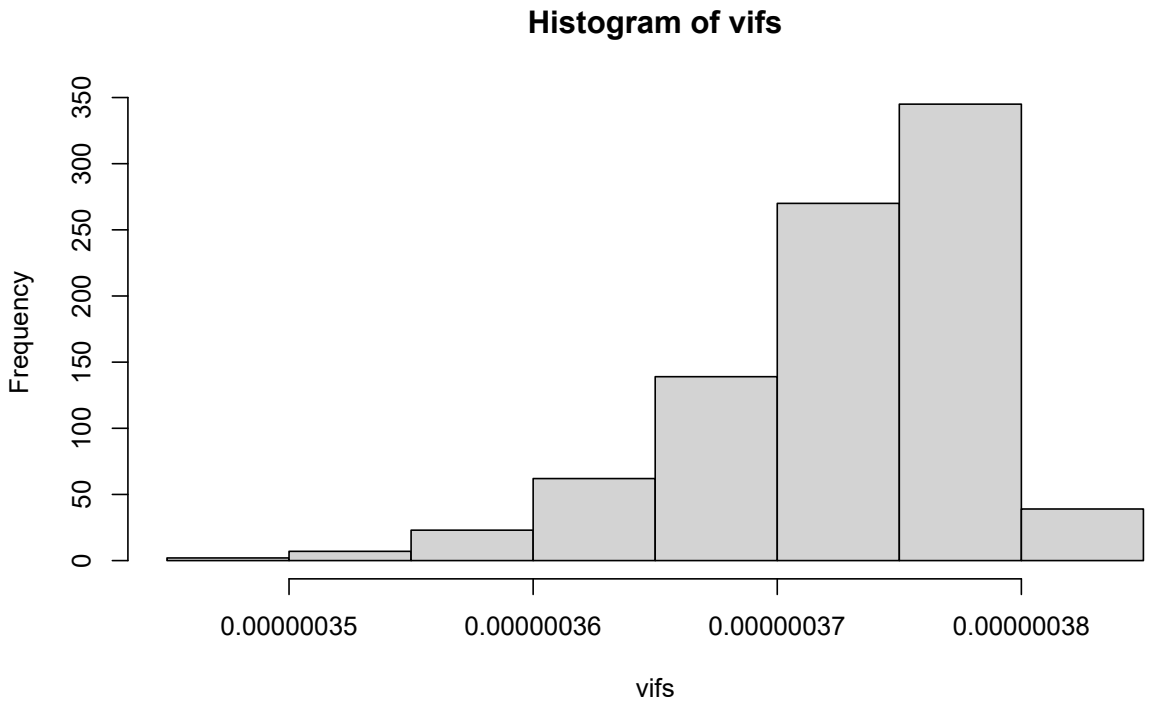
	R-square	MAE	RMSE
Linear Regression	0.644	0.522	0.644
Ridge Regression	0.727	0.435	0.536

Impact on VIFs

VIF values for the ridge regression can also be calculated using the following matrix operation,

$$(r_{\mathbf{XX}} + \lambda \mathbf{I})^{-1} r_{\mathbf{XX}} (r_{\mathbf{XX}} + \lambda \mathbf{I})^{-1}.$$

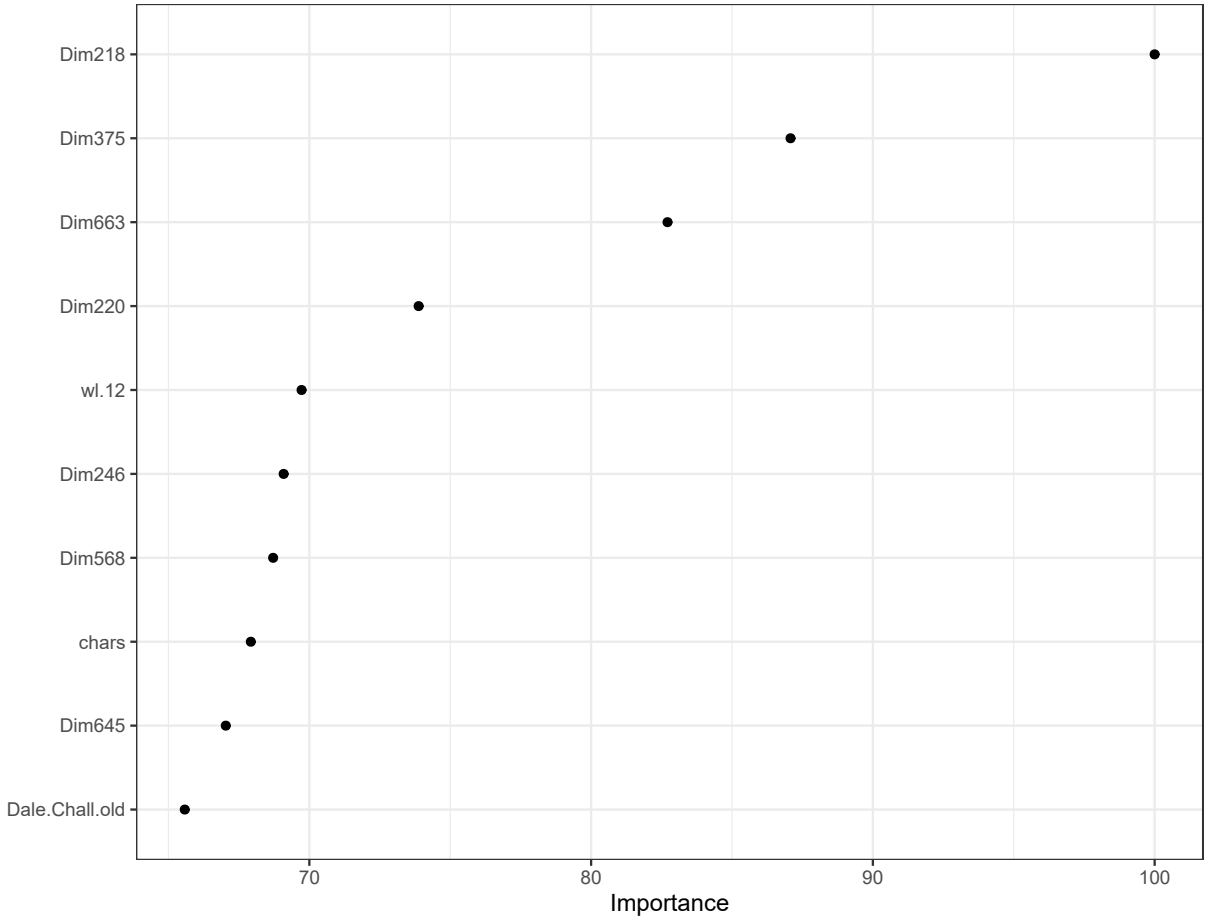
Note that the optimal value of λ yielded by our grid search using `glmnet` was 0.63; however, we know that the λ used by `glmnet` is reduced by a factor of N. Therefore, we should replace the λ in this formula by 2384*0.63 to find the new VIF values.



Variable Importance

Variable importance in ridge regression can be evaluated based on the magnitude of the standardized coefficients and be obtained using `vip()` function from the `vip` package. For instance, the plot below shows the most importan 10 predictors of readability scores in this dataset.

```
#install.packages('vip')

require(vip)

vip(ridge, num_features = 10, geom = "point") +
  theme_bw()
```



The Importance values on the X-axis are directly related to the magnitude of the standardized coefficients obtained from the final model. The standardized regression coefficients are rescaled such that the largest regression coefficient is 100 and the rescaled values are used on the X-axis. Below provides a list of the 10 largest standardized regression coefficients.

```
coefs <- coef(ridge$finalModel,ridge$bestTune$lambda)

ind   <- order(abs(coefs),decreasing=T)

head(as.matrix(coefs[ind[-1],]),10)
```

```
                    [,1]
Dim218          0.03535232
Dim375         -0.03078513
Dim663          0.02924184
Dim220          0.02611983
wl.12          -0.02465119
Dim246         -0.02442627
Dim568          0.02429442
chars          -0.02401463
Dim645         -0.02370056
Dale.Chall.old -0.02318597
```

Lasso Regression

Lasso regression is very similar to the Ridge regression. The only difference is that it applies a different penalty to the loss function. Assuming that there are P regression coefficients in the model, the penalty term for the ridge regression would be

$$\lambda \sum_{i=1}^{P} |\beta_p|,$$

where λ is again the penalty constant and $|\beta_p|$ is the absolute value of the regression coefficient for the P^{th} parameter. Lasso regression also penalizes the regression coefficients when they get larger, but in a different way. When we fit a regression model with lasso penalty, the loss function to minimize becomes

$$Loss = \sum_{i=1}^{N} \epsilon_{(i)}^2 + \lambda \sum_{i=1}^{P} |\beta_p|,$$

$$Loss = SSR + \lambda \sum_{i=1}^{P} |\beta_p|.$$

Let's consider again the same example where we fit a simple linear regression model such that the readability score is the outcome (Y) and average word length is the predictor(X). Our regression model is

$$Y = \beta_0 + \beta_1 X + \epsilon,$$

and let's assume the set of coefficients are $\{\beta_0, \beta_1\} = \{7.5,\text{-}2\}$, so my model is

$$Y = 7.5 - 2X + \epsilon.$$

Then, the value of the loss function when $\lambda = 0.2$ would be equal to 17.284.

```
readability_sub <- read.csv('https://raw.githubusercontent.com/uo-datasci-specialization/c4-ml-fall-202

d <-  readability_sub[,c('mean.wl','target')]

b0 = 7.5
b1 = -2

d$predicted <- b0 + b1*d$mean.wl
d$error     <- d$target - d$predicted

d
```

```
    mean.wl      target  predicted       error
1  4.603659 -2.58590836 -1.7073171 -0.87859129
2  3.830688  0.45993224 -0.1613757  0.62130790
3  4.180851 -1.07470758 -0.8617021 -0.21300545
4  4.015544 -1.81700402 -0.5310881 -1.28591594
5  4.686047 -1.81491744 -1.8720930  0.05717559
6  4.211340 -0.94968236 -0.9226804 -0.02700194
7  4.025000 -0.12103065 -0.5500000  0.42896935
8  4.443182 -2.82200582 -1.3863636 -1.43564218
9  4.089385 -0.74845172 -0.6787709 -0.06968077
10 4.156757  0.73948755 -0.8135135  1.55300107
11 4.463277 -0.96218937 -1.4265537  0.46436430
12 5.478261 -2.21514888 -3.4565217  1.24137286
13 4.770492 -1.21845136 -2.0409836  0.82253224
14 4.568966 -1.89544351 -1.6379310 -0.25751247
15 4.735751 -0.04101056 -1.9715026  1.93049203
16 4.372340 -1.83716516 -1.2446809 -0.59248431
17 4.103448 -0.18818586 -0.7068966  0.51871069
18 4.042857 -0.81739314 -0.5857143 -0.23167886
19 4.202703 -1.86307557 -0.9054054 -0.95767016
20 3.853535 -0.41630158 -0.2070707 -0.20923088
```

```
lambda = 0.2

loss <- sum((d$error)^2) + lambda*(abs(b0) + abs(b1))

loss
```

```
[1] 17.28364
```

When λ is equal to 0, the loss function is again identical to SSR; therefore, it becomes a linear regression with no regularization. You can similarly try a number of different values for λ using the shiny app at this link and explore how the loss function value and coefficient estimates change for different values of λ.

Below is also a demonstration of what happens to loss function and the regression coefficients for increasing levels of loss penalty (λ).

Model Estimation

Unfortunately, there is no closed form solution for lasso regression due to the absolute value terms in the loss function. The only way to estimate the coefficients of the lasso regression is to optimize the loss function using numerical techniques and obtain computational approximations of the regression coefficients. Similar to the ridge regression, `glmnet` is an engine we can use to estimate the coefficients of the lasso regression.

glmnet() function We can fit the lasso regression by setting the `alpha=` argument to 1 in `glmnet()` and specifying the penalty term (λ).

```
Y <-  as.matrix(readability_sub$target)
X <-  as.matrix(cbind(readability_sub$mean.wl,readability_sub$sents))
Y <- scale(Y)
X <- scale(X)
```

```
mod <- glmnet(x = X,
              y = Y,
              family = 'gaussian',
              alpha = 1,
              lambda = 0.5,
              intercept=FALSE)


coef(mod)
```

```
3 x 1 sparse Matrix of class "dgCMatrix"
                    s0
(Intercept) .
V1          .
V2          0.1785686
```

Notice that there is a `.` symbol for the coefficient of the first predictor. This indicates that it is equal to zero. While the regression coefficients in the ridge regression shrink to zero, they are not necessarily end up being exactly equal to zero. In contrast, lasso regression may yield a value of zero for some coefficients in the model. For this reason, lasso regression may be used as a variable selection algorithm. The variables with coefficients equal to zero may be discarded from future considerations as they are found to be not important for predicting the outcome.

Tuning λ We implement a similar strategy for finding the optimal value of λ. We try many different values of λ and check how well the model performs based on a certain criteria (e.g., MAE, MSE, RMSE) using a k-fold cross validation. Then, we pick the value of λ that provides the best performance.

Using Lasso Regression to Predict the Readability Scores

In this section, we will apply the lasso regression to predict the readability scores from all predictors in the dataset. We will use the `caret` package and use 10-fold cross validation to evaluate the model performance for different levels of penalty term (λ).

```
# Load the packages

  require(caret)
  require(recipes)
  require(finalfit)
  require(glmnet)

# Import the dataset

  readability <- read.csv('https://raw.githubusercontent.com/uo-datasci-specialization/c4-ml-fall-2021/m

# Initial preparation (remove variables with large amount of missingness)

  require(finalfit)

  missing_ <- ff_glimpse(readability)$Continuous
  flag_na <- which(as.numeric(missing_$missing_percent) > 80)
  readability <- readability[,-flag_na]
```

```r
# Set the random seed for reproducibility

  set.seed(10152021)

# Train/Test Split

  loc      <- sample(1:nrow(readability), round(nrow(readability) * 0.9))
  read_tr  <- readability[loc, ]
  read_te  <- readability[-loc, ]

# Blueprint

  blueprint <- recipe(x     = readability,
                      vars  = colnames(readability),
                      roles = c(rep('predictor',990),'outcome')) %>%
    step_zv(all_numeric()) %>%
    step_nzv(all_numeric()) %>%
    step_impute_mean(all_numeric()) %>%
    step_normalize(all_numeric_predictors()) %>%
    step_corr(all_numeric(),threshold=0.9)

# Cross validation settings

    # Randomly shuffle the data

      read_tr = read_tr[sample(nrow(read_tr)),]

    # Create 10 folds with equal size

      folds = cut(seq(1,nrow(read_tr)),breaks=10,labels=FALSE)

    # Create the list for each fold

      my.indices <- vector('list',10)
      for(i in 1:10){
        my.indices[[i]] <- which(folds!=i)
      }

  cv <- trainControl(method = "cv",
                     index  = my.indices)

# Tune Grid

  # Note that we set the value of alpha to 1 for lasso regression

  grid <- data.frame(alpha = 1, lambda = seq(0.01,3,.01))

# Train the model

  lasso <- caret::train(blueprint,
                        data      = read_tr,
                        method    = "glmnet",
                        trControl = cv,
```

```
                        tuneGrid  = grid)

  lasso$results
```

```
grid <- data.frame(alpha = 1, lambda = seq(0.001,0.015,.001))

grid

# Train the model

  lasso2 <- caret::train(blueprint,
                        data      = read_tr,
                        method    = "glmnet",
                        trControl = cv,
                        tuneGrid  = grid)

  lasso2$results
```

```
    alpha lambda      RMSE  Rsquared       MAE     RMSESD RsquaredSD      MAESD
1       1   0.01 0.5527021 0.7167313 0.4403822 0.03063071 0.02560409 0.02066813
2       1   0.02 0.5627915 0.7073538 0.4472971 0.02921962 0.02528707 0.02030593
3       1   0.03 0.5720316 0.6994504 0.4538987 0.02857803 0.02606661 0.02046186
4       1   0.04 0.5809996 0.6925455 0.4609607 0.02888263 0.02712518 0.02158991
5       1   0.05 0.5899363 0.6859944 0.4682094 0.02962202 0.02857089 0.02269408
6       1   0.06 0.5983950 0.6804278 0.4749360 0.03056496 0.03013211 0.02403291
7       1   0.07 0.6067620 0.6753616 0.4816176 0.03122209 0.03138664 0.02501375
8       1   0.08 0.6154053 0.6703696 0.4885315 0.03187616 0.03228234 0.02589924
9       1   0.09 0.6246012 0.6648980 0.4958061 0.03246466 0.03317177 0.02659769
10      1   0.10 0.6339658 0.6594438 0.5032602 0.03278424 0.03371821 0.02702865
11      1   0.11 0.6432597 0.6542425 0.5106759 0.03293731 0.03387951 0.02743164
12      1   0.12 0.6530183 0.6485795 0.5186997 0.03314533 0.03398951 0.02804703
13      1   0.13 0.6630736 0.6424338 0.5269805 0.03328544 0.03419125 0.02840416
14      1   0.14 0.6733798 0.6358758 0.5355602 0.03332334 0.03428356 0.02857934
15      1   0.15 0.6839339 0.6287857 0.5442988 0.03334627 0.03431994 0.02873817
16      1   0.16 0.6947581 0.6210412 0.5532677 0.03332968 0.03439348 0.02879910
17      1   0.17 0.7054986 0.6132777 0.5621888 0.03289089 0.03429294 0.02880663
18      1   0.18 0.7159612 0.6058828 0.5708931 0.03259265 0.03429948 0.02896959
19      1   0.19 0.7260092 0.5992464 0.5793802 0.03247260 0.03367761 0.02933742
20      1   0.20 0.7354862 0.5939249 0.5874709 0.03256865 0.03398105 0.02985925
21      1   0.21 0.7450280 0.5884470 0.5955982 0.03275205 0.03411181 0.03047628
22      1   0.22 0.7542913 0.5836651 0.6034861 0.03306855 0.03439364 0.03113245
23      1   0.23 0.7637802 0.5783553 0.6116313 0.03342458 0.03444597 0.03187720
24      1   0.24 0.7733223 0.5728547 0.6199231 0.03368250 0.03469948 0.03251607
25      1   0.25 0.7830962 0.5665993 0.6284125 0.03398241 0.03485172 0.03320708
26      1   0.26 0.7929600 0.5598681 0.6369056 0.03434747 0.03522679 0.03384372
27      1   0.27 0.8029632 0.5524125 0.6455101 0.03476707 0.03550203 0.03445929
28      1   0.28 0.8131476 0.5439342 0.6541977 0.03519655 0.03583389 0.03503651
29      1   0.29 0.8233295 0.5348910 0.6629340 0.03566621 0.03600798 0.03565359
30      1   0.30 0.8337204 0.5243750 0.6718814 0.03616824 0.03600612 0.03619208
31      1   0.31 0.8442680 0.5122781 0.6809006 0.03664997 0.03598503 0.03670537
32      1   0.32 0.8549094 0.4985543 0.6900067 0.03715020 0.03558948 0.03725145
33      1   0.33 0.8657502 0.4824705 0.6993388 0.03767417 0.03504059 0.03778672
34      1   0.34 0.8758868 0.4675867 0.7080425 0.03799133 0.03645114 0.03830921
```

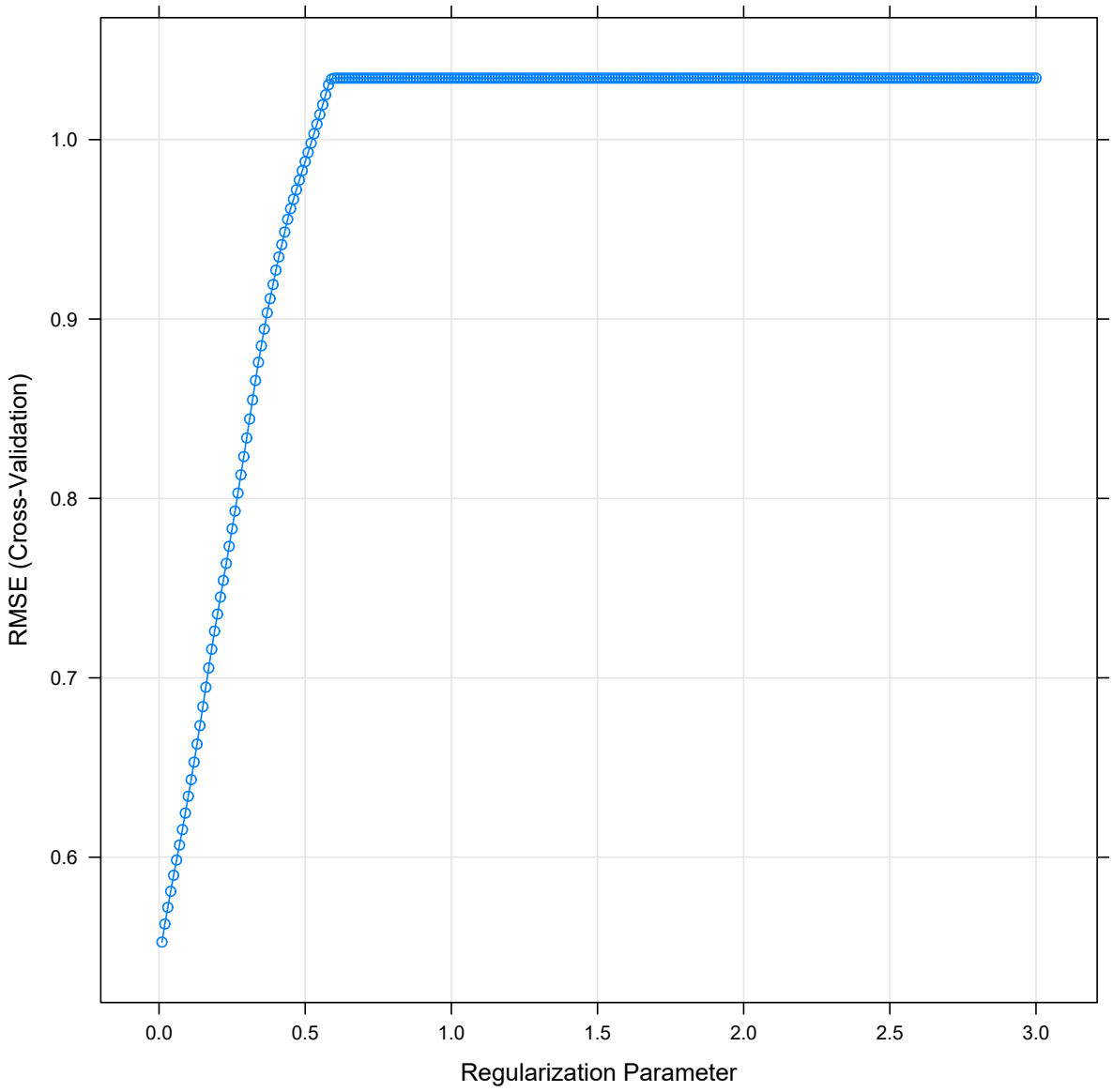
35	1	0.35	0.8850811	0.4554043	0.7159393	0.03840270	0.03670658	0.03891451
36	1	0.36	0.8944171	0.4411948	0.7239806	0.03883784	0.03679666	0.03946033
37	1	0.37	0.9034904	0.4265608	0.7317854	0.03909087	0.03821715	0.03979209
38	1	0.38	0.9113644	0.4164396	0.7385942	0.03935626	0.03963577	0.04006758
39	1	0.39	0.9192466	0.4051814	0.7454635	0.03971783	0.04018679	0.04042466
40	1	0.40	0.9272551	0.3918303	0.7524619	0.04008227	0.04076101	0.04076394
41	1	0.41	0.9346167	0.3809401	0.7589076	0.04018751	0.04379497	0.04088227
42	1	0.42	0.9414967	0.3717469	0.7649467	0.04047894	0.04469194	0.04116205
43	1	0.43	0.9484847	0.3604938	0.7710559	0.04077539	0.04551394	0.04143911
44	1	0.44	0.9555848	0.3465886	0.7772563	0.04107380	0.04629255	0.04170599
45	1	0.45	0.9615259	0.3388156	0.7824853	0.04065570	0.04937270	0.04136882
46	1	0.46	0.9667619	0.3358377	0.7871491	0.04048222	0.04771768	0.04120147
47	1	0.47	0.9720670	0.3323149	0.7918378	0.04032493	0.04553810	0.04102643
48	1	0.48	0.9774496	0.3279718	0.7966229	0.04018055	0.04284746	0.04084313
49	1	0.49	0.9826934	0.3250728	0.8013138	0.04009098	0.04206595	0.04069133
50	1	0.50	0.9877362	0.3249005	0.8058246	0.04009909	0.04227020	0.04060460
51	1	0.51	0.9928387	0.3249005	0.8103491	0.04013265	0.04227020	0.04053231
52	1	0.52	0.9980154	0.3249005	0.8149023	0.04016897	0.04227020	0.04046007
53	1	0.53	1.0032653	0.3249005	0.8194837	0.04020825	0.04227020	0.04038701
54	1	0.54	1.0085871	0.3249005	0.8240885	0.04025074	0.04227020	0.04033668
55	1	0.55	1.0139796	0.3249005	0.8287283	0.04029665	0.04227020	0.04030961
56	1	0.56	1.0194419	0.3249005	0.8334178	0.04034623	0.04227020	0.04026812
57	1	0.57	1.0249726	0.3249005	0.8381464	0.04039970	0.04227020	0.04020223
58	1	0.58	1.0305709	0.3249005	0.8429013	0.04045728	0.04227020	0.04013365
59	1	0.59	1.0337551	0.2629539	0.8456337	0.03941865	0.02289979	0.03927805
60	1	0.60	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
61	1	0.61	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
62	1	0.62	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
63	1	0.63	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
64	1	0.64	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
65	1	0.65	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
66	1	0.66	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
67	1	0.67	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
68	1	0.68	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
69	1	0.69	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
70	1	0.70	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
71	1	0.71	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
72	1	0.72	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
73	1	0.73	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
74	1	0.74	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
75	1	0.75	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
76	1	0.76	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
77	1	0.77	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
78	1	0.78	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
79	1	0.79	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
80	1	0.80	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
81	1	0.81	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
82	1	0.82	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
83	1	0.83	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
84	1	0.84	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
85	1	0.85	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
86	1	0.86	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
87	1	0.87	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700
88	1	0.88	1.0342776	NaN	0.8460946	0.03888238	NA	0.03880700

89	1	0.89	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
90	1	0.90	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
91	1	0.91	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
92	1	0.92	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
93	1	0.93	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
94	1	0.94	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
95	1	0.95	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
96	1	0.96	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
97	1	0.97	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
98	1	0.98	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
99	1	0.99	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
100	1	1.00	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
101	1	1.01	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
102	1	1.02	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
103	1	1.03	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
104	1	1.04	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
105	1	1.05	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
106	1	1.06	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
107	1	1.07	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
108	1	1.08	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
109	1	1.09	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
110	1	1.10	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
111	1	1.11	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
112	1	1.12	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
113	1	1.13	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
114	1	1.14	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
115	1	1.15	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
116	1	1.16	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
117	1	1.17	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
118	1	1.18	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
119	1	1.19	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
120	1	1.20	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
121	1	1.21	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
122	1	1.22	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
123	1	1.23	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
124	1	1.24	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
125	1	1.25	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
126	1	1.26	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
127	1	1.27	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
128	1	1.28	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
129	1	1.29	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
130	1	1.30	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
131	1	1.31	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
132	1	1.32	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
133	1	1.33	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
134	1	1.34	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
135	1	1.35	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
136	1	1.36	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
137	1	1.37	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
138	1	1.38	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
139	1	1.39	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
140	1	1.40	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
141	1	1.41	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
142	1	1.42	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700

143	1	1.43	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
144	1	1.44	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
145	1	1.45	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
146	1	1.46	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
147	1	1.47	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
148	1	1.48	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
149	1	1.49	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
150	1	1.50	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
151	1	1.51	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
152	1	1.52	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
153	1	1.53	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
154	1	1.54	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
155	1	1.55	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
156	1	1.56	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
157	1	1.57	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
158	1	1.58	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
159	1	1.59	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
160	1	1.60	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
161	1	1.61	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
162	1	1.62	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
163	1	1.63	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
164	1	1.64	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
165	1	1.65	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
166	1	1.66	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
167	1	1.67	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
168	1	1.68	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
169	1	1.69	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
170	1	1.70	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
171	1	1.71	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
172	1	1.72	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
173	1	1.73	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
174	1	1.74	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
175	1	1.75	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
176	1	1.76	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
177	1	1.77	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
178	1	1.78	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
179	1	1.79	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
180	1	1.80	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
181	1	1.81	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
182	1	1.82	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
183	1	1.83	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
184	1	1.84	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
185	1	1.85	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
186	1	1.86	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
187	1	1.87	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
188	1	1.88	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
189	1	1.89	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
190	1	1.90	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
191	1	1.91	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
192	1	1.92	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
193	1	1.93	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
194	1	1.94	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
195	1	1.95	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
196	1	1.96	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700

197	1	1.97	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
198	1	1.98	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
199	1	1.99	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
200	1	2.00	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
201	1	2.01	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
202	1	2.02	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
203	1	2.03	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
204	1	2.04	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
205	1	2.05	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
206	1	2.06	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
207	1	2.07	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
208	1	2.08	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
209	1	2.09	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
210	1	2.10	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
211	1	2.11	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
212	1	2.12	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
213	1	2.13	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
214	1	2.14	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
215	1	2.15	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
216	1	2.16	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
217	1	2.17	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
218	1	2.18	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
219	1	2.19	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
220	1	2.20	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
221	1	2.21	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
222	1	2.22	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
223	1	2.23	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
224	1	2.24	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
225	1	2.25	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
226	1	2.26	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
227	1	2.27	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
228	1	2.28	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
229	1	2.29	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
230	1	2.30	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
231	1	2.31	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
232	1	2.32	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
233	1	2.33	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
234	1	2.34	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
235	1	2.35	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
236	1	2.36	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
237	1	2.37	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
238	1	2.38	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
239	1	2.39	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
240	1	2.40	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
241	1	2.41	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
242	1	2.42	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
243	1	2.43	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
244	1	2.44	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
245	1	2.45	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
246	1	2.46	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
247	1	2.47	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
248	1	2.48	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
249	1	2.49	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700
250	1	2.50	1.0342776	NaN	0.8460946	0.03888238	NA 0.03880700

```
251     1   2.51 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
252     1   2.52 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
253     1   2.53 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
254     1   2.54 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
255     1   2.55 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
256     1   2.56 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
257     1   2.57 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
258     1   2.58 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
259     1   2.59 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
260     1   2.60 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
261     1   2.61 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
262     1   2.62 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
263     1   2.63 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
264     1   2.64 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
265     1   2.65 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
266     1   2.66 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
267     1   2.67 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
268     1   2.68 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
269     1   2.69 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
270     1   2.70 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
271     1   2.71 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
272     1   2.72 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
273     1   2.73 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
274     1   2.74 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
275     1   2.75 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
276     1   2.76 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
277     1   2.77 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
278     1   2.78 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
279     1   2.79 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
280     1   2.80 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
281     1   2.81 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
282     1   2.82 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
283     1   2.83 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
284     1   2.84 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
285     1   2.85 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
286     1   2.86 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
287     1   2.87 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
288     1   2.88 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
289     1   2.89 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
290     1   2.90 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
291     1   2.91 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
292     1   2.92 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
293     1   2.93 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
294     1   2.94 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
295     1   2.95 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
296     1   2.96 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
297     1   2.97 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
298     1   2.98 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
299     1   2.99 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
300     1   3.00 1.0342776       NaN 0.8460946 0.03888238         NA 0.03880700
```

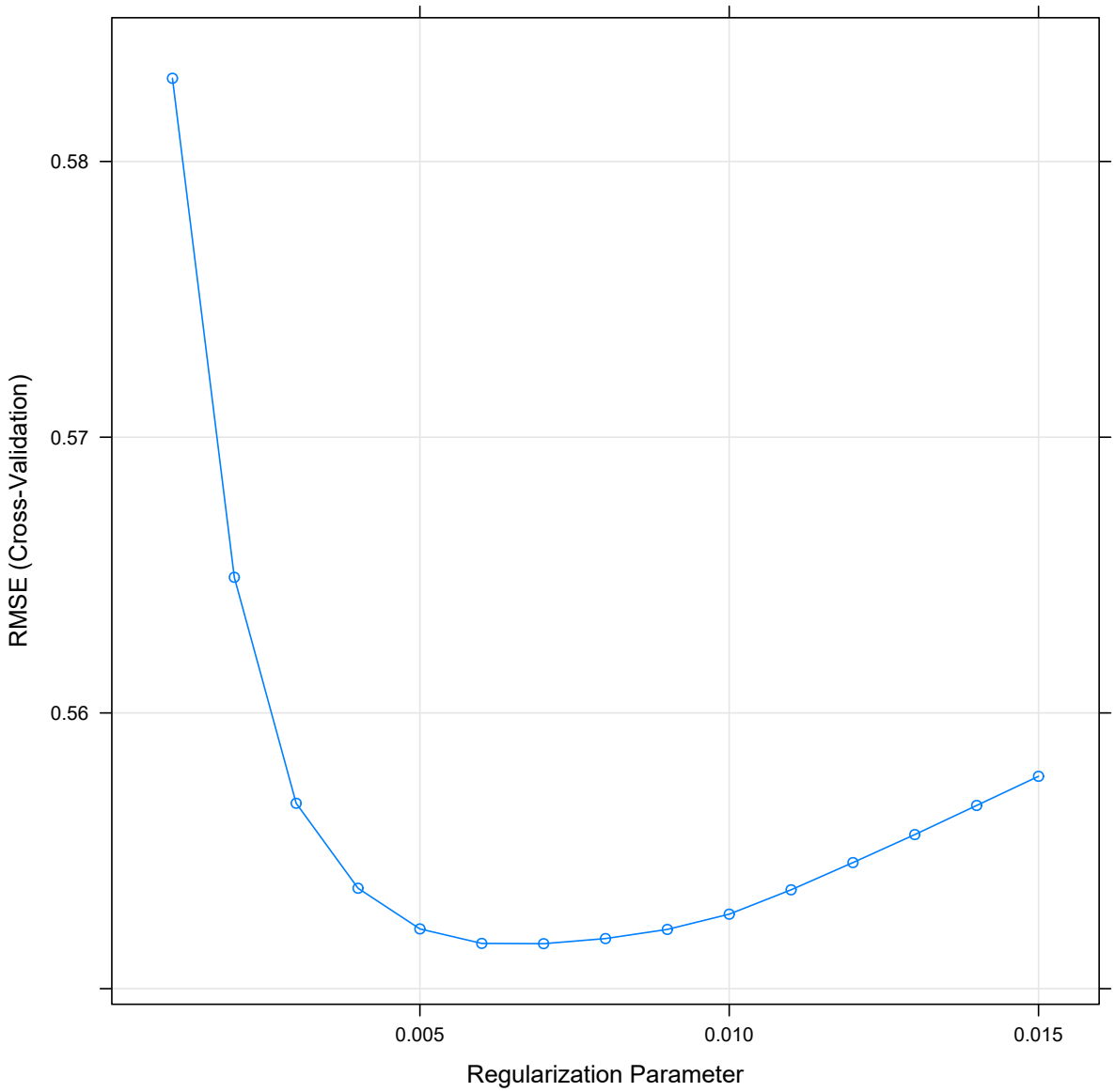


Compared to the ridge regression, we have a different result. It seems a very large λ value is not useful at all. In fact, any number larger than .01 made the predictions worse. In this case, we can do another search with really small numbers. We will update our grid, and we will look at the λ values from 0.001 to 0.015 with increments of 0.001.

```
grid <- data.frame(alpha = 1, lambda = seq(0.005,0.015,.001))

# Train the model

  lasso2 <- caret::train(blueprint,
                         data      = read_tr,
                         method    = "glmnet",
                         trControl = cv,
                         tuneGrid  = grid)
```

```
lasso2$results


lasso2$bestTune

plot(lasso2)
```

```
   alpha lambda      RMSE  Rsquared       MAE     RMSESD RsquaredSD      MAESD
1      1  0.001 0.5830163 0.6907522 0.4645742 0.03638327 0.02843938 0.03070448
2      1  0.002 0.5649188 0.7065728 0.4507361 0.03318174 0.02746046 0.02646456
3      1  0.003 0.5567239 0.7138610 0.4433280 0.03269790 0.02729168 0.02530233
4      1  0.004 0.5536425 0.7164928 0.4401351 0.03225470 0.02706128 0.02414390
5      1  0.005 0.5521671 0.7176980 0.4385788 0.03213741 0.02682849 0.02339077
6      1  0.006 0.5516420 0.7180015 0.4381638 0.03184394 0.02646033 0.02288465
7      1  0.007 0.5516315 0.7178569 0.4384046 0.03174775 0.02648458 0.02248970
8      1  0.008 0.5518174 0.7176045 0.4390774 0.03135930 0.02620771 0.02173153
9      1  0.009 0.5521487 0.7172609 0.4397817 0.03095173 0.02587008 0.02112498
10     1  0.010 0.5527021 0.7167313 0.4403822 0.03063071 0.02560409 0.02066813
11     1  0.011 0.5535851 0.7158751 0.4410780 0.03050288 0.02559054 0.02044073
12     1  0.012 0.5545729 0.7149311 0.4417915 0.03030850 0.02551712 0.02029668
13     1  0.013 0.5555884 0.7139690 0.4424747 0.03015095 0.02542215 0.02032593
14     1  0.014 0.5566442 0.7129784 0.4432138 0.02996679 0.02533131 0.02032013
15     1  0.015 0.5577015 0.7119926 0.4439236 0.02980517 0.02527210 0.02038409

  alpha lambda
7     1  0.007
```



It looks like a λ value of 0.007 is the best pick although any number between 0.005 and 0.010 probably works fine. Let's now use the lasso regression model to predict the outcome in the hold-out test dataset as we did before for the ridge regression.

```
predict_te_lasso<- predict(lasso2, read_te)

rsq_te <- cor(read_te$target,predict_te_lasso)^2
rsq_te
```

```
[1] 0.7247531
```

```
mae_te <- mean(abs(read_te$target - predict_te_lasso))
mae_te
```

```
[1] 0.4341808
```

```
rmse_te <- sqrt(mean((read_te$target - predict_te_lasso)^2))
rmse_te
```

```
[1] 0.5380492
```

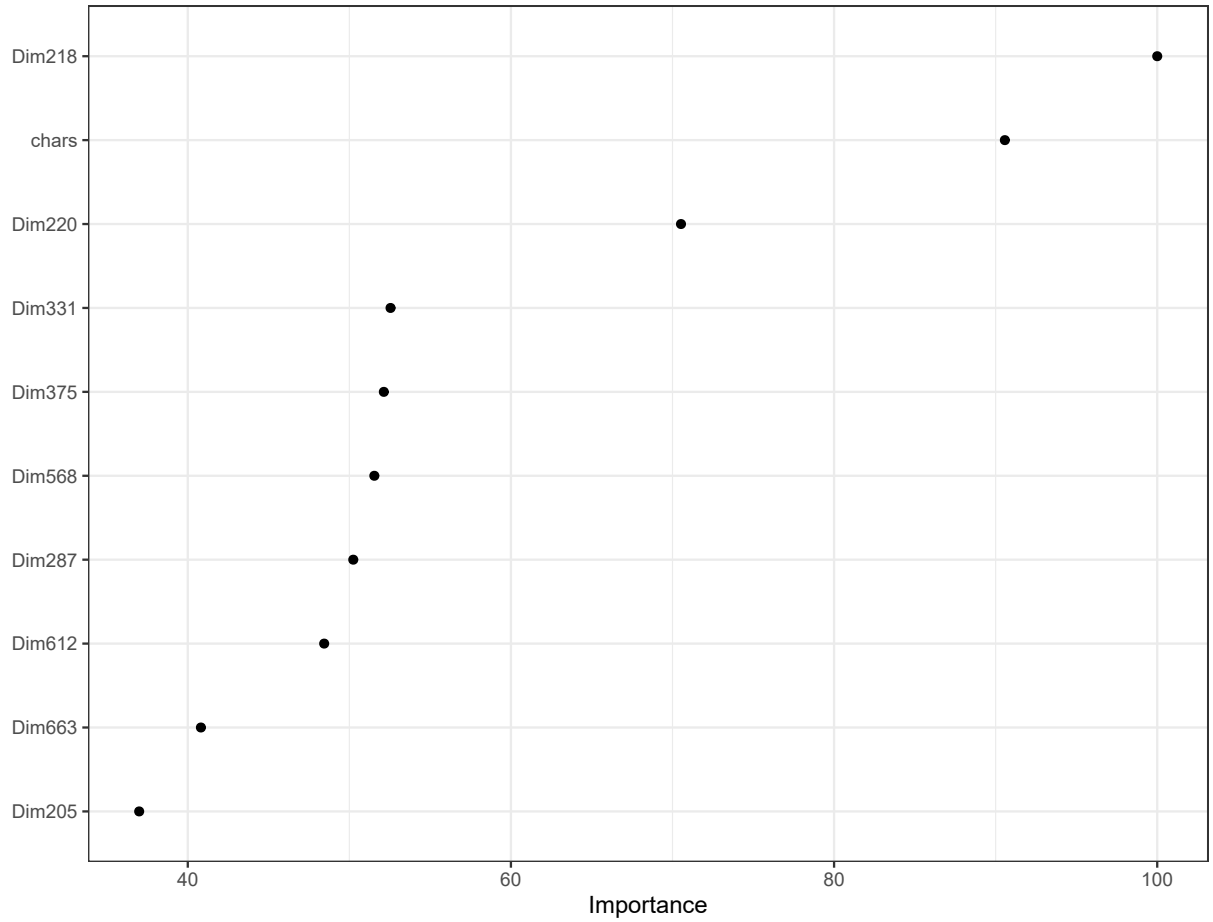
Below is a table to compare the performance of linear regression, ridge regression, and linear regression on the test dataset. The performance of the lasso regression was very close to the ridge regression but didn't provide any significant improvement over the ridge regression.

	R-square	MAE	RMSE
Linear Regression	0.644	0.522	0.644
Ridge Regression	0.727	0.435	0.536
Lasso Regression	0.725	0.434	0.538

Variable Importance

We can again look at the variable importance using the `vip` function. While there are similarities and common important variables, the lasso regression provided a different top 10 important variables.

```
vip(lasso2, num_features = 10, geom = "point") +
  theme_bw()
```



Below provides a list of the 10 largest standardized regression coefficients. Out of 887 predictors, lasso yielded a value of zero for the regression coefficient for 571 predictors.

```
coefs <- coef(lasso2$finalModel,lasso2$bestTune$lambda)

coefs.zero <- coefs[which(coefs[,1]==0),]
length(coefs.zero)
```

```
[1] 610
```

```
coefs.nonzero <- coefs[which(coefs[,1]!=0),]
length(coefs.nonzero)
```

```
[1] 278
```

```
ind   <- order(abs(coefs.nonzero),decreasing=T)
head(as.matrix(coefs.nonzero[ind[-1]]),10)
```

```
              [,1]
Dim218  0.12620547
chars  -0.11430373
```

```
Dim220  0.08900703
Dim331 -0.06631947
Dim375 -0.06579564
Dim568  0.06505346
Dim287  0.06340827
Dim612  0.06113590
Dim663  0.05150805
Dim205 -0.04668562
```

Elastic Net

Elastic net simply combines the two types of penalty into one by mixing them together with some kind of weighted average. The penalty term for the elastic net could be written as

$$\lambda \left[(1-\alpha) \sum_{i=1}^{P} \beta_p^2 + \alpha \sum_{i=1}^{P} |\beta_p|) \right].$$

Note that this term reduces to

$$\lambda \sum_{i=1}^{P} \beta_p^2$$

when α is equal to 1 and to

$$\lambda \sum_{i=1}^{P} |\beta_p|.$$

when α is equal to 0. When α is set to 1, this is equivalent of ridge regression. When α is equal to 0 this is equivalent of lasso regression. When α takes any value between 0 and 1, this term becomes a weighted average of ridge penalty and lasso penalty. In Elastic Net, there are two hyperparameters to be tuned, α and λ. We can consider all possible combinations of these two hyperparameters, and try to find the optimal combination using a 10-fold cross validation.

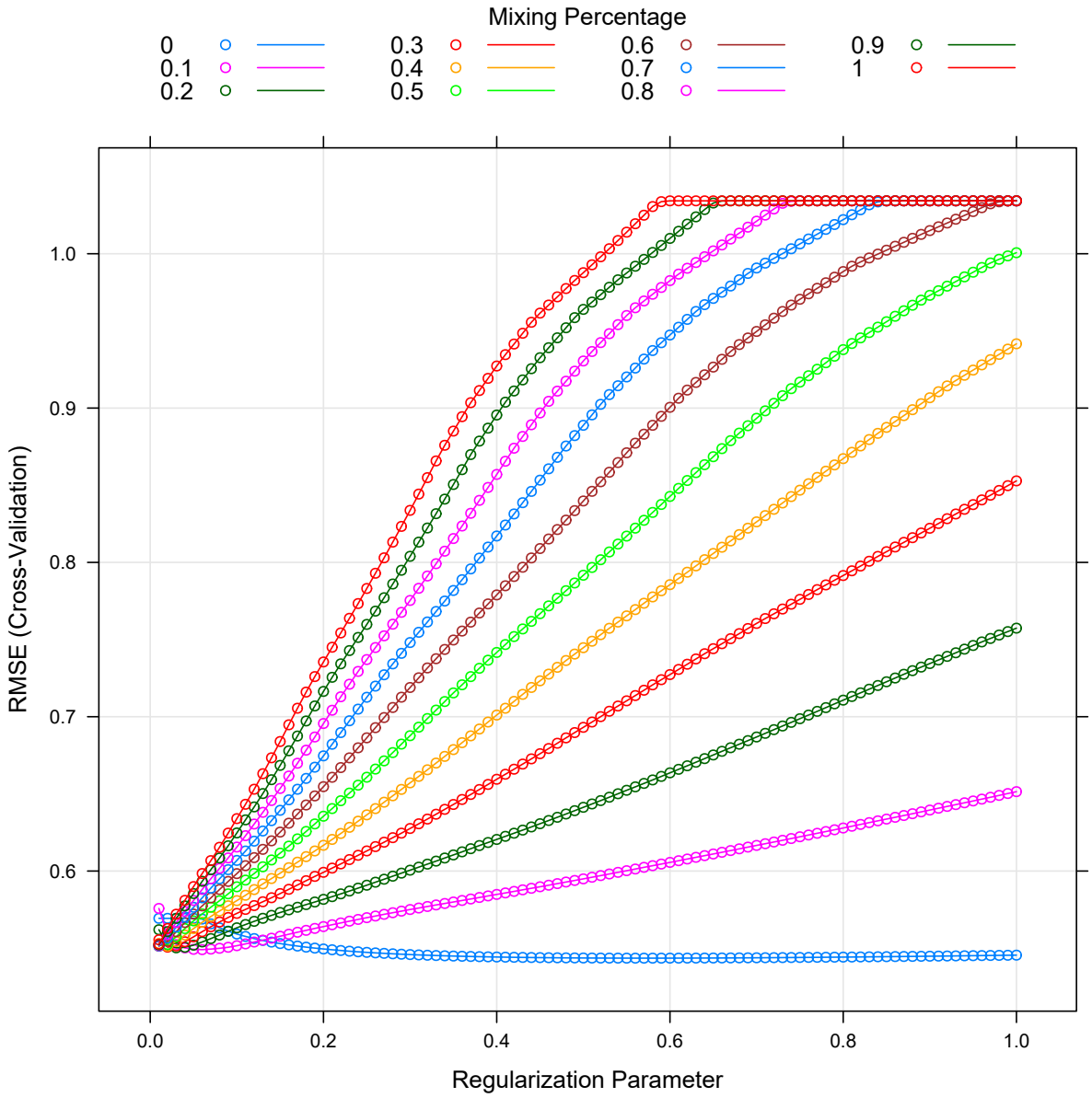
Below is a syntax we can use to predict the readability scores using elastic net. The elastic net results indicate that there is not much benefit of mixing, and the results are consistent that the best performance is still provided by the ridge regression.

```
# Load the packages

  require(caret)
  require(recipes)
  require(finalfit)
  require(glmnet)

# Import the dataset

  readability <- read.csv('https://raw.githubusercontent.com/uo-datasci-specialization/c4-ml-fall-2021/m

# Initial preparation (remove variables with large amount of missingness)

  require(finalfit)

  missing_ <- ff_glimpse(readability)$Continuous
```

```
  flag_na <- which(as.numeric(missing_$missing_percent) > 80)
  readability <- readability[,-flag_na]

# Set the random seed for reproducibility

  set.seed(10152021)

# Train/Test Split

  loc      <- sample(1:nrow(readability), round(nrow(readability) * 0.9))
  read_tr  <- readability[loc, ]
  read_te  <- readability[-loc, ]

# Blueprint

  blueprint <- recipe(x     = readability,
                      vars  = colnames(readability),
                      roles = c(rep('predictor',990),'outcome')) %>%
    step_zv(all_numeric()) %>%
    step_nzv(all_numeric()) %>%
    step_impute_mean(all_numeric()) %>%
    step_normalize(all_numeric_predictors()) %>%
    step_corr(all_numeric(),threshold=0.9)

# Cross validation settings

    # Randomly shuffle the data

      read_tr = read_tr[sample(nrow(read_tr)),]

    # Create 10 folds with equal size

      folds = cut(seq(1,nrow(read_tr)),breaks=10,labels=FALSE)

    # Create the list for each fold

      my.indices <- vector('list',10)
      for(i in 1:10){
        my.indices[[i]] <- which(folds!=i)
      }

  cv <- trainControl(method = "cv",
                     index  = my.indices)

# Tune Grid

  # Note that we set the value of alpha to 1 for lasso regression

  grid <- expand.grid(alpha = seq(0,1,.1), lambda = seq(0.01,1,.01))

# Train the model

  elastic <- caret::train(blueprint,
```

```
                        data       = read_tr,
                        method     = "glmnet",
                        trControl  = cv,
                        tuneGrid   = grid)

  # elastic$results

  elastic$bestTune

  plot(elastic)
```

```
   alpha lambda
57     0   0.57
```



Using the Prediction Model for a New Text

Compile the code to generate input features as a function. This function will require two inputs, a model object and a new text. The function will then return a a matrix of input features.

```
generate_feats <- function(my.model,new.text){

    # Tokenization and document-feature matrix

      tokenized <- tokens(new.text,
                          remove_punct = TRUE,
                          remove_numbers = TRUE,
                          remove_symbols = TRUE,
                          remove_separators = TRUE)

      dm <- dfm(tokenized)

    # basic text stats

      text_sm <- textstat_summary(dm)
      text_sm$sents <- nsentence(new.text)
      text_sm$chars <- nchar(new.text)

    # Word-length features

      wl <- nchar(tokenized[[1]])

      wl.tab <- table(wl)

      wl.features <- data.frame(matrix(0,nrow=1,nco=30))
      colnames(wl.features) <- paste0('wl.',1:30)

      ind <- colnames(wl.features)%in%paste0('wl.',names(wl.tab))

      wl.features[,ind] <- wl.tab

      wl.features$mean.wl  <-   mean(wl)
      wl.features$sd.wl    <-   sd(wl)
      wl.features$min.wl   <-   min(wl)
      wl.features$max.wl   <-   max(wl)

    # Text entropy/Max entropy ratio

      t.ent <- textstat_entropy(dm)
      n     <-  sum(featfreq(dm))
      p     <- rep(1/n,n)
      m.ent <- -sum(p*log(p,base=2))

      ent <- t.ent$entropy/m.ent

    # Lexical diversity

      text_lexdiv <- textstat_lexdiv(tokenized,
                                     remove_numbers = TRUE,
```

```
                                  remove_punct   = TRUE,
                                  remove_symbols = TRUE,
                                  measure        = 'all')

# Measures of readability

  text_readability <- textstat_readability(new.text,measure='all')

# POS tag frequency

  annotated <- udpipe_annotate(ud_eng, x = new.text)
  annotated <- as.data.frame(annotated)
  annotated <- cbind_morphological(annotated)

  pos_tags <- c(table(annotated$upos),table(annotated$xpos))

# Syntactic relations

  dep_rel <- table(annotated$dep_rel)

# morphological features

  feat_names <- c('morph_abbr','morph_animacy','morph_aspect','morph_case',
                  'morph_clusivity','morph_definite','morph_degree',
                  'morph_evident','morph_foreign','morph_gender','morph_mood',
                  'morph_nounclass','morph_number','morph_numtype',
                  'morph_person','morph_polarity','morph_polite','morph_poss',
                  'morph_prontype','morph_reflex','morph_tense','morph_typo',
                  'morph_verbform','morph_voice')

  feat_vec <- c()

  for(j in 1:length(feat_names)){

    if(feat_names[j]%in%colnames(annotated)){
      morph_tmp   <- table(annotated[,feat_names[j]])
      names_tmp   <- paste0(feat_names[j],'_',names(morph_tmp))
      morph_tmp   <- as.vector(morph_tmp)
      names(morph_tmp) <- names_tmp
      feat_vec  <- c(feat_vec,morph_tmp)
    }
  }

# Sentence Embeddings

  embeds <- textEmbed(x     = new.text,
                      model = 'roberta-base',
                      layers = 12,
                      context_aggregation_layers = 'concatenate')


# combine them all into one vector and store in the list object
```

```
      input <- cbind(text_sm[2:length(text_sm)],
                               wl.features,
                               as.data.frame(ent),
                               text_lexdiv[,2:ncol(text_lexdiv)],
                               text_readability[,2:ncol(text_readability)],
                               t(as.data.frame(pos_tags)),
                               t(as.data.frame(c(dep_rel))),
                               t(as.data.frame(feat_vec)),
                               as.data.frame(embeds$x)
                               )

    # feature names from the model

      my_feats <- my.model$recipe$var_info$variable

    # Find the features missing from the new text

      missing_feats <- ! my_feats %in% colnames(input)

    # Add the missing features (with assigned values of zeros)

      temp           <- data.frame(matrix(0,1,sum(missing_feats)))
      colnames(temp) <- my_feats[missing_feats]

      input <- cbind(input,temp)

      return(list(input=input))
}
```

For a given text, predict the scores using the ridge and lasso regression models we trained in this lectures

```
# For the next few lines of codes to work, you will need the following
# in your R environment
  # 1. R Libraries (quanteda, quanteda.textstats, text, udpipe, reticulate)
  # 2. Supplemental Python libraries (torch, tokenizers, nltk, transformers,numpy)
  # 3. Model object (caret_mod)
  # 4. A function to generate the features in the model (generate_feats)


##############################################################################

  require(quanteda)
  require(quanteda.textstats)
  require(udpipe)
  require(reticulate)
  require(text)

  ud_eng <- udpipe_load_model(here('english-ewt-ud-2.5-191206.udpipe'))

  reticulate::import('torch')
```

```
Module(torch)
```

```
  reticulate::import('numpy')
```

```
Module(numpy)
```

```
  reticulate::import('transformers')
```

```
Module(transformers)
```

```
  reticulate::import('nltk')
```

```
Module(nltk)
```

```
  reticulate::import('tokenizers')
```

```
Module(tokenizers)
```

```
################################################################################

# Sample texts

my.text1   <- 'Sora has a new kite. The kite is red. It is a good kite to fly.'

my.text2   <- 'Saguaros have roots underground that grow outward rather than downward.'

  # Ridge regression

  ridge.inputs1 <- generate_feats(my.model = ridge,
                                  new.text = my.text1)

  ridge.inputs2 <- generate_feats(my.model = ridge,
                                  new.text = my.text2)


  predict(ridge, ridge.inputs1$input)
```

```
[1] 1.067174
```

```
  predict(ridge, ridge.inputs2$input)
```

```
[1] 0.2562777
```

```
  # Lasso Regression

  lasso.inputs1 <- generate_feats(my.model = lasso2,
                                  new.text = my.text1)

  lasso.inputs2 <- generate_feats(my.model = lasso2,
                                  new.text = my.text2)


  predict(lasso2, lasso.inputs1$input)
```

```
[1] 1.079534
```

```
  predict(lasso2, lasso.inputs2$input)
```

```
[1] 0.2934944
```