Assign1_ADMPA

Janakisrija Maddukuri

03/13/2022

PART A:

QA1. What is the main purpose of regularization when training predictive models? 10 points

Purpose of Regularization:

Any machine learning model's goal is to learn patterns from training data and generalize them to efficiently predict unobserved data. As a result, generalization refers to a model's ability to respond to new inputs. The model's lack of generalization might be attributed to several factors. One of them is the model's overfitting. When a model becomes too intricate, it learns the detail and noise in the training data to the point where it severely affects the model's performance on fresh data. This means that the model picks up on noise or random fluctuations in the training data and learns them as ideas. As a result, the estimator's variance has grown.

Regularization is an approach for increasing the generalization capacity of a model by reducing its complexity. Regularization aims to improve the performance of a model by simplifying it. Regularization penalizes the model to simplify it and minimize its complexity. Machine learning models are calibrated via regularization to reduce the adjusted loss function and avoid overfitting or underfitting. We may train our machine learning model suitably on a particular test set and thereby reduce the mistakes in it using Regularization.

Example of Regularization techniques:

- Dropout is used for regularizing the deep learning models.
- Lasso & Ridge Regression is used for the regularization of regression models.

QA4. What is the role of the lambda parameter in regularized linear models such as Lasso or Ridge regression models? 10 points

Lambda is a hyperparameter in linear models like Lasso or Ridge Regression that is used for regularization or reducing the model's complexity.

In Lasso & Ridge Regression, Lambda is used to regularize the model.

To arrive at the best weights for each attribute, linear regression models strive to reduce the loss function (Ordinary Least Squares). When a model is overfitted and unable to generalize adequately, regularization is used to penalize it. This is accomplished by including an additional term for the loss function, whose value is proportionate to the Lambda value. As a result, increasing the Lambda value raises the overall loss value, and the model tries to reduce the coefficients of the parameters to find the best answer. The notion is that by decreasing or regularizing the coefficients, one can enhance prediction accuracy, reduce variance, and improve model interpretability.

We add a penalty to ridge regression by employing a tuning parameter lambda that is selected using cross validation. The goal is to reduce the size of the fit by reducing the residual sum of squares and imposing a shrinkage penalty. The shrinkage penalty is equal to lambda times the sum of squares of the coefficients, therefore big coefficients are punished. The bias remains constant as lambda increases, but the variance decreases. The disadvantage of ridge is that it does not allow you to pick variables. It includes all the variables that will be used in the final model.

The lasso regression’s cost function is as follows:

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

$$= RSS + \lambda \sum_{j=1}^{p} \beta_j^2$$

The penalty in lasso is equivalent to the total of the coefficients' absolute values. When lambda is large enough, Lasso decreases the coefficient estimates towards zero, and it has the effect of setting variables exactly equal to zero, whereas ridge does not. As a result, lasso accomplishes variable selection in the same way that the best subset selection method does. Cross validation is used to select the tuning parameter lambda. The lambda of the tuning parameter that is used in the lasso regression is chosen by cross validation.

The cost function of the lasso regression is as follows:

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

$$= RSS + \lambda \sum_{j=1}^{p} \beta_j^2$$

QA2. What is the role of a loss function in a predictive model? And name two common loss functions for regression models and two common loss functions for classification models. 10 points

Loss functions are used to determine how far a predictive model's estimated value differs from its true value. They specify a goal against which the model's performance is measured, and the parameters learned by the model are selected by minimizing a loss function. As a result, the loss function is the function that calculates the distance between the algorithm's current output and the expected output. It's a way of determining how well your algorithm

models the data. Changing the model's parameters until the lowest possible loss is attained continues the process of minimizing the loss. Some examples of Loss functions and how they work are shown below.

Common loss functions for classification models:

Binary Cross Entropy:

For classification issues with two classes, this is the most common loss function. The seemingly disparate word "entropy" has a statistical meaning. Entropy is a measure of the difference in randomness between two random variables, while cross entropy is a measure of the difference in randomness between two random variables.

The cross-entropy loss increases as the difference between the predicted probability and the actual label grows. Predicting a chance of.011 when the actual observation label is 1 would result in a huge loss value, according to this. A "perfect" model would have a log loss of 0 in an ideal circumstance.

$$J = -\sum_{i=1}^{N} y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i))$$

Where, hθ(xi) is the predicted value after hypothesis.

Categorical Cross Entropy Loss:

Categorical Cross Entropy Loss is just Binary Cross Entropy Loss multiplied by the number of classes. When using a categorical cross entropy loss function, one condition is that the labels be one-hot encoded. As the other members in the vector are multiplied by zero, only one element will be non-zero. This trait is extended to a SoftMax activation function.

Common loss functions for regression models:

Mean Squared Error (MSE):

The average of the squared differences between the actual and anticipated values is the Mean Squared Error. The mean squared error is defined for a data point Yi and its predicted value I where n is the total number of data points in the collection.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Mean Absolute Error (MAE):

The Mean Absolute Error (also known as L1 loss) is a simple yet reliable loss function used in regression models. Due to the occurrence of outliers, regression problems may have variables that are not precisely Gaussian in nature (values that are very different from the rest of the data). In such instances, Mean Absolute Error would be an excellent choice because it ignores the direction of the outliers (unrealistically high positive or negative values). MAE calculates the average of the absolute discrepancies between the actual and anticipated values, as the name implies. The mean absolute error for a data point xi and its predicted value yi, where n is the total number of data points in the collection, is defined as follows:

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

QA3. Consider the following scenario. You are building a classification model with many hyper parameters on a relatively small dataset. You will see that the training error is extremely small. Can you fully trust this model? Discuss the reason. 10 points

When building a prediction model with a training set, it should be tested on both the training and validation sets. Only if both the train error and the validation error (error on data not observed by the model) are low can the model be regarded to be a good prediction model. There are two reasons why a prediction model is regarded to be dumb, as listed below.

When a model is underfitted, it signifies that it is unable to accurately reflect the relationship between the input and output variables. As a result, the model's performance on both the training and validation sets is poor. As a result, we must enhance the model's complexity for it to accurately capture the link between the input and output variables.

When a model is overfitted, it performs well on the training set with high train accuracy, but when the same model is applied to the validation set, the error is very high and the variance between the train and validation error is quite large. This indicates that the model has become overly complex, as it now learns the train data with all the noise and outliers and is sensitive to even minor changes in the input variables. As a result, the goal of any machine learning model, Generalization, is not met, causing the model to perform poorly on unobserved data. Overfitting occurs most frequently when the train data is small, and the model's complexity is large.

The model with multiple hyperparameters in the offered example is developed using a limited dataset. As a result, there's a good likelihood the model is overfitted. As previously stated, a model is only regarded to be a good prediction model if it performs well on both train and unseen data. Because the given model is prone to overfitting and has a high variance, it performs well on the training set but is likely to perform poorly on unseen data due to overfitting.

As a result, we can't put our faith in the model just based on how well it predicts train data. Instead, it should be applied to unknown data to examine if there is any difference between the train and validation errors. If the variance is excessive, the model should be retrained by

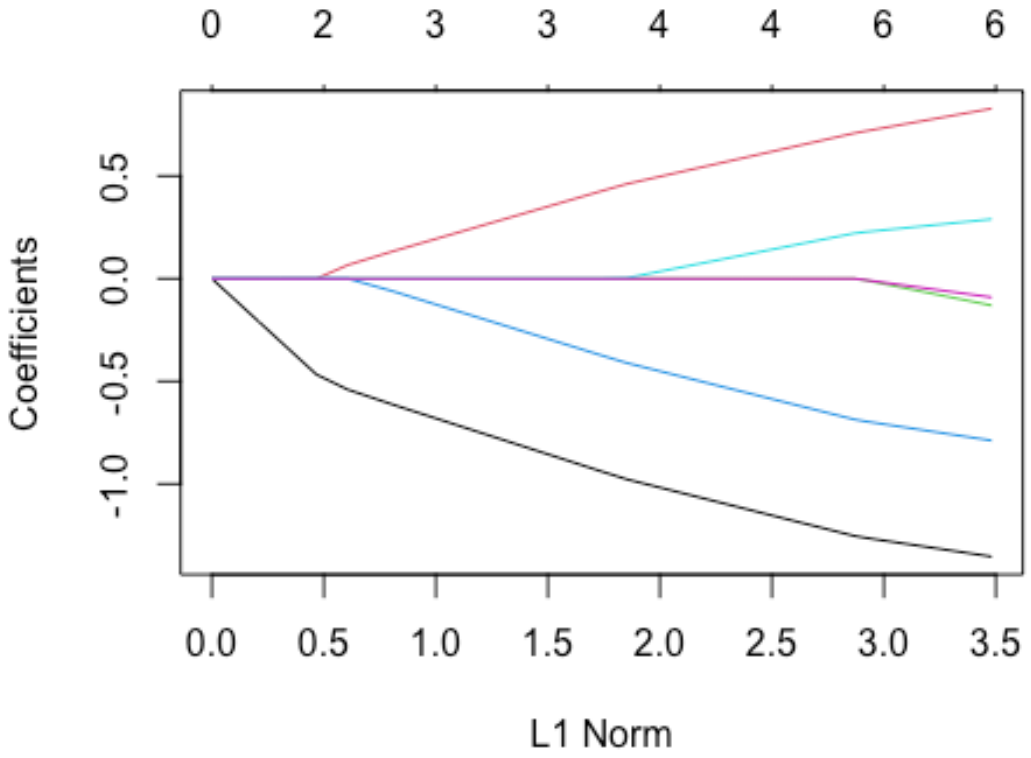
lowering the model's complexity or using regularization techniques to help the model generalize rather than only learning the patterns in the train dataset.

```
# Loading the required libraries
library(ISLR)
library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(glmnet)

## Loading required package: Matrix

## Loaded glmnet 4.1-3

library(caret)

## Loading required package: ggplot2

## Loading required package: lattice

attach(Carseats)
summary(Carseats)

##      Sales          CompPrice       Income        Advertising
##  Min.   : 0.000   Min.   : 77   Min.   : 21.00   Min.   : 0.000
##  1st Qu.: 5.390   1st Qu.:115   1st Qu.: 42.75   1st Qu.: 0.000
##  Median : 7.490   Median :125   Median : 69.00   Median : 5.000
##  Mean   : 7.496   Mean   :125   Mean   : 68.66   Mean   : 6.635
##  3rd Qu.: 9.320   3rd Qu.:135   3rd Qu.: 91.00   3rd Qu.:12.000
##  Max.   :16.270   Max.   :175   Max.   :120.00   Max.   :29.000
##    Population        Price        ShelveLoc        Age          Education
##  Min.   : 10.0   Min.   : 24.0   Bad   : 96   Min.   :25.00   Min.   :10.0
##  1st Qu.:139.0   1st Qu.:100.0   Good  : 85   1st Qu.:39.75   1st Qu.:12.0
##  Median :272.0   Median :117.0   Medium:219   Median :54.50   Median :14.0
##  Mean   :264.8   Mean   :115.8                Mean   :53.32   Mean   :13.9
##  3rd Qu.:398.5   3rd Qu.:131.0                3rd Qu.:66.00   3rd Qu.:16.0
##  Max.   :509.0   Max.   :191.0                Max.   :80.00   Max.   :18.0
##  Urban       US
##  No :118   No :142
```

```
##  Yes:282   Yes:258
##
##
##
##
```

QB1) Build a Lasso regression model to predict Sales based on all other attributes ("Price", "Advertising", "Population", "Age", "Income" and "Education"). What is the best value of lambda for such a lasso model?

```
# Taking all the input attributes into Carseats_Filtered and then scaling the
input attributes.
Carseats_Filtered <- Carseats %>% select( "Price", "Advertising", "Population
", "Age", "Income", "Education") %>% scale(center = TRUE, scale = TRUE) %>% a
s.matrix()

# using glmnet library to convert the input attributes to matrix format.
x <- Carseats_Filtered

# storing the response variable into y in matrix format
y <- Carseats %>% select("Sales") %>% as.matrix()

## building the model
fit = glmnet(x, y)
summary(fit)

##           Length Class     Mode
## a0         62    -none-    numeric
## beta      372    dgCMatrix S4
## df         62    -none-    numeric
## dim         2    -none-    numeric
## lambda     62    -none-    numeric
## dev.ratio  62    -none-    numeric
## nulldev     1    -none-    numeric
## npasses     1    -none-    numeric
## jerr        1    -none-    numeric
## offset      1    -none-    logical
## call        3    -none-    call
## nobs        1    -none-    numeric

plot(fit)
```



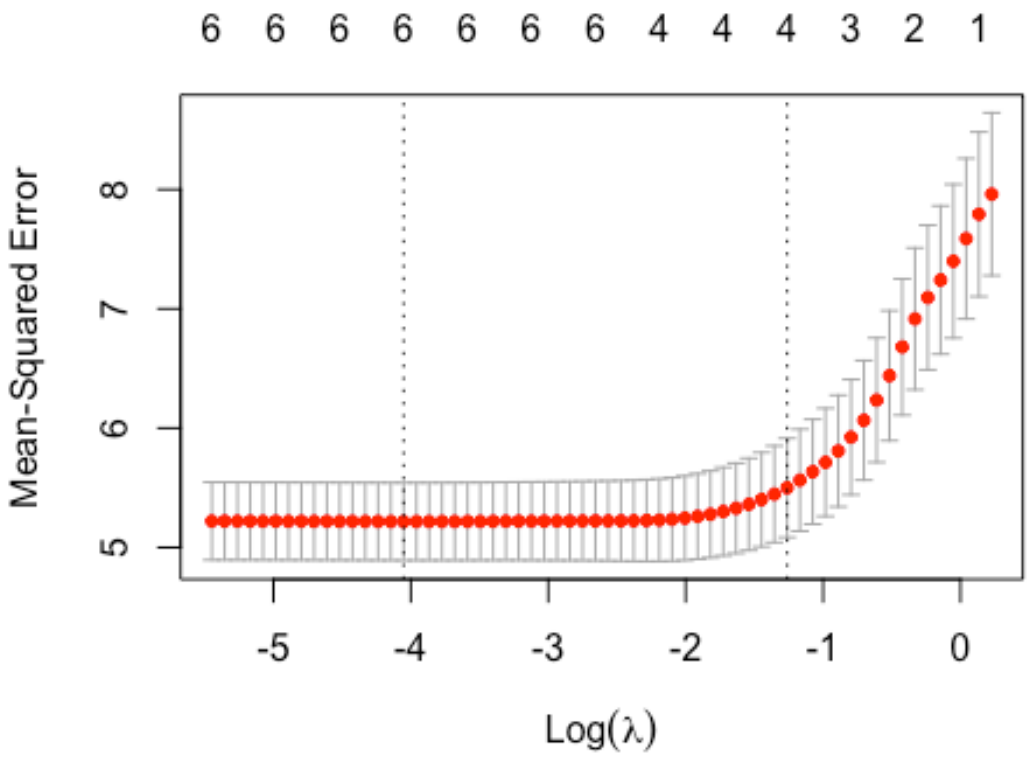

```
print(fit)
```

```
##
## Call:  glmnet(x = x, y = y)
##
##    Df  %Dev  Lambda
## 1   0  0.00 1.25500
## 2   1  3.36 1.14400
## 3   1  6.15 1.04200
## 4   1  8.47 0.94940
## 5   1 10.39 0.86500
## 6   1 11.99 0.78820
## 7   2 14.62 0.71820
## 8   3 18.08 0.65440
## 9   3 21.12 0.59620
## 10  3 23.64 0.54330
## 11  3 25.73 0.49500
## 12  3 27.46 0.45100
## 13  3 28.91 0.41100
## 14  3 30.10 0.37450
## 15  4 31.12 0.34120
## 16  4 32.13 0.31090
## 17  4 32.97 0.28330
```

```
## 18  4 33.67 0.25810
## 19  4 34.25 0.23520
## 20  4 34.73 0.21430
## 21  4 35.13 0.19520
## 22  4 35.46 0.17790
## 23  4 35.74 0.16210
## 24  4 35.97 0.14770
## 25  4 36.16 0.13460
## 26  4 36.31 0.12260
## 27  4 36.45 0.11170
## 28  4 36.55 0.10180
## 29  4 36.64 0.09276
## 30  6 36.75 0.08451
## 31  6 36.86 0.07701
## 32  6 36.95 0.07017
## 33  6 37.02 0.06393
## 34  6 37.09 0.05825
## 35  6 37.14 0.05308
## 36  6 37.18 0.04836
## 37  6 37.21 0.04407
## 38  6 37.24 0.04015
## 39  6 37.27 0.03658
## 40  6 37.29 0.03333
## 41  6 37.30 0.03037
## 42  6 37.32 0.02767
## 43  6 37.33 0.02522
## 44  6 37.34 0.02298
## 45  6 37.35 0.02094
## 46  6 37.35 0.01908
## 47  6 37.36 0.01738
## 48  6 37.36 0.01584
## 49  6 37.37 0.01443
## 50  6 37.37 0.01315
## 51  6 37.37 0.01198
## 52  6 37.38 0.01092
## 53  6 37.38 0.00995
## 54  6 37.38 0.00906
## 55  6 37.38 0.00826
## 56  6 37.38 0.00752
## 57  6 37.38 0.00686
## 58  6 37.38 0.00625
## 59  6 37.38 0.00569
## 60  6 37.38 0.00519
## 61  6 37.38 0.00472
## 62  6 37.38 0.00430
```

```
cv_fit <- cv.glmnet(x, y, alpha = 1)

# finding the minimum lambda value
```

```
best_lambda <- cv_fit$lambda.min
best_lambda

## [1] 0.01738061

plot(cv_fit)
```



So, from the above results, we can see that only 37.38% variance in the target variable, sales with regularization and a best lambda value which is 0.0043.

QB2. What is the coefficient for the price (normalized) attribute in the best model (i.e. model with the optimal lambda)?

```
best_model <- glmnet(x, y, alpha = 1, lambda = best_lambda)
coef(best_model)

## 7 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept)  7.49632500
## Price       -1.33889011
## Advertising  0.81012314
## Population  -0.11054223
```

```
## Age         -0.77329377
## Income       0.27914466
## Education   -0.07725352
```

The coefficient of the Price attribute with the best lambda value is -1.35384596.

QB3. How many attributes remain in the model if lambda is set to 0.01? How that number changes if lambda is increased to 0.1? Do you expect more variables to stay in the model (i.e., to have non-zero coefficients) as we increase lambda?

```
# Let us see the coefficients of the attributes that are still remained if la
mbda is set to 0.01.

best_model <- glmnet(x, y, alpha = 1, lambda = 0.01)
coef(best_model)

## 7 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept)  7.49632500
## Price       -1.34733223
## Advertising  0.82026088
## Population  -0.12187685
## Age         -0.78190633
## Income       0.28488631
## Education   -0.08502707
```

Above are the coefficients of the independent attributes with the lambda value 0.01. No coefficients are eliminated here.

```
# Let us see the coefficients of the attributes that are still remained if la
mbda is set to 0.1.

best_model <- glmnet(x, y, alpha = 1, lambda = 0.1)
coef(best_model)

## 7 x 1 sparse Matrix of class "dgCMatrix"
##                      s0
## (Intercept)  7.4963250
## Price       -1.2447745
## Advertising  0.7007230
## Population   .
## Age         -0.6775428
## Income       0.2139222
## Education    .
```

We can say from the above results that the values of the independent attributes have shrinked to some extent and that two of the coefficients of the attributes are eliminated when the lambda is set to 0.1.

```
# Let us see the coefficients of the attributes that are still remained if la
mbda is set to 0.3.

best_model <- glmnet(x, y, alpha = 1, lambda = 0.3)
coef(best_model)
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept)  7.49632500
## Price       -1.02298693
## Advertising  0.50192192
## Population   .
## Age         -0.45635365
## Income       0.03900787
## Education    .
```

From the above results we can see that two of the coefficients of the attributes are eliminated and the independent attributes have shrinked further when lambda value is 0.3.

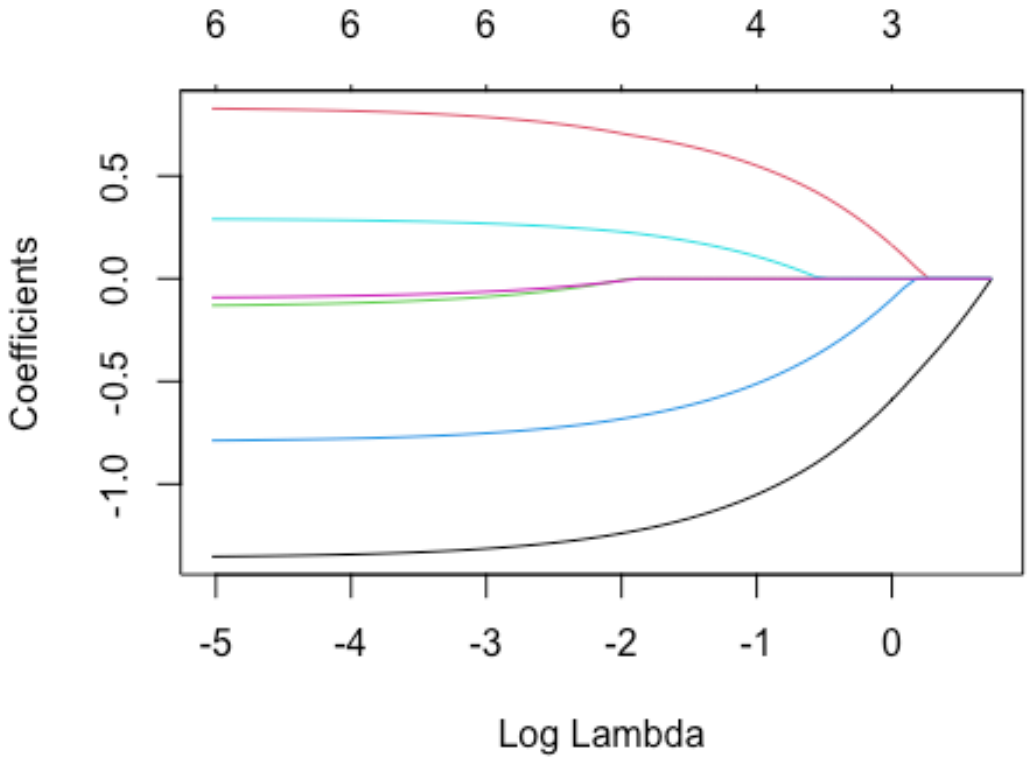
```
# Let us see the coefficients of the attributes that are still remained if la
mbda is set to 0.5.

best_model <- glmnet(x, y, alpha = 1, lambda = 0.5)
coef(best_model)
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                      s0
## (Intercept)  7.4963250
## Price       -0.7929743
## Advertising  0.2947434
## Population   .
## Age         -0.2337276
## Income       .
## Education    .
```

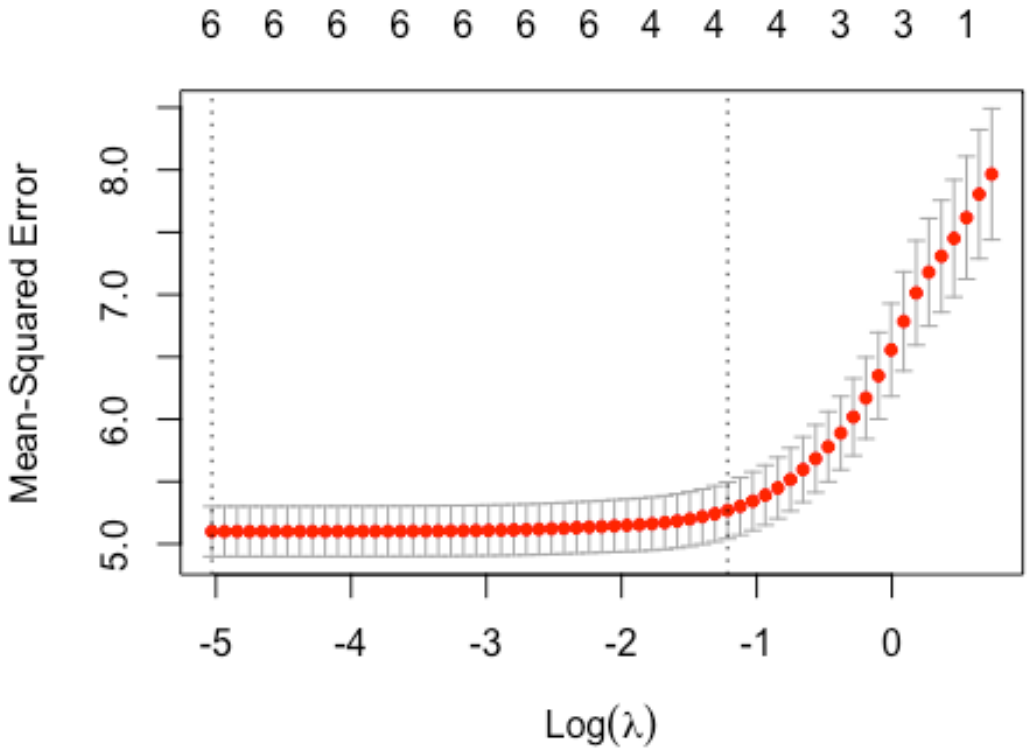
From the above results we can see that three of the coefficients of the attributes are eliminated and the independent attributes have shrinked further when lambda value is 0.5.

QB4. Build an elastic-net model with alpha set to 0.6. What is the best value of lambda for such a model?

```
# Building an elastic_net model with alpha = 0.6
el_net = glmnet(x, y, alpha = 0.6)
plot(el_net, xvar = "lambda")
```




```
plot(cv.glmnet(x, y, alpha = 0.6))
```




```
summary(el_net)

##           Length Class     Mode
## a0         63    -none-    numeric
## beta      378    dgCMatrix S4
## df         63    -none-    numeric
## dim         2    -none-    numeric
## lambda     63    -none-    numeric
## dev.ratio  63    -none-    numeric
## nulldev     1    -none-    numeric
## npasses     1    -none-    numeric
## jerr        1    -none-    numeric
## offset      1    -none-    logical
## call        4    -none-    call
## nobs        1    -none-    numeric

print(el_net)

##
## Call:  glmnet(x = x, y = y, alpha = 0.6)
##
##    Df  %Dev  Lambda
## 1   0  0.00 2.09200
## 2   1  2.67 1.90600
```

```
## 3   1  5.03 1.73700
## 4   1  7.09 1.58200
## 5   1  8.90 1.44200
## 6   1 10.47 1.31400
## 7   2 12.89 1.19700
## 8   3 16.00 1.09100
## 9   3 18.95 0.99370
## 10  3 21.49 0.90540
## 11  3 23.67 0.82500
## 12  3 25.55 0.75170
## 13  3 27.15 0.68490
## 14  3 28.52 0.62410
## 15  4 29.75 0.56860
## 16  4 30.91 0.51810
## 17  4 31.89 0.47210
## 18  4 32.72 0.43020
## 19  4 33.43 0.39190
## 20  4 34.02 0.35710
## 21  4 34.52 0.32540
## 22  4 34.93 0.29650
## 23  4 35.29 0.27020
## 24  4 35.58 0.24620
## 25  4 35.83 0.22430
## 26  4 36.04 0.20440
## 27  4 36.21 0.18620
## 28  4 36.36 0.16970
## 29  4 36.48 0.15460
## 30  6 36.60 0.14090
## 31  6 36.73 0.12830
## 32  6 36.84 0.11690
## 33  6 36.93 0.10660
## 34  6 37.01 0.09709
## 35  6 37.07 0.08846
## 36  6 37.12 0.08060
## 37  6 37.17 0.07344
## 38  6 37.20 0.06692
## 39  6 37.23 0.06097
## 40  6 37.26 0.05556
## 41  6 37.28 0.05062
## 42  6 37.30 0.04612
## 43  6 37.31 0.04203
## 44  6 37.33 0.03829
## 45  6 37.34 0.03489
## 46  6 37.34 0.03179
## 47  6 37.35 0.02897
## 48  6 37.36 0.02639
## 49  6 37.36 0.02405
## 50  6 37.37 0.02191
## 51  6 37.37 0.01997
## 52  6 37.37 0.01819
```

```
## 53  6 37.37 0.01658
## 54  6 37.38 0.01510
## 55  6 37.38 0.01376
## 56  6 37.38 0.01254
## 57  6 37.38 0.01143
## 58  6 37.38 0.01041
## 59  6 37.38 0.00949
## 60  6 37.38 0.00864
## 61  6 37.38 0.00788
## 62  6 37.38 0.00718
## 63  6 37.38 0.00654
```

We can observe from the above results that the variance is 37.38 in the dependent variable (Sales) which is explained by the given attributes to apply the regularization by setting the alpha value to 0.6 and the best lambda value is 0.00654.