ADVANCED DATA MINING AND PREDICTIVE ANALYTICS

(BA-64037-003)

SPRING 2024

ASSIGNMENT – 1

DIVYA CHANDRASEKARAN

ID: 811284790

PART A

QUESTION -1

What is the main purpose of regularization when training predictive models?

Answer:

When training a machine learning model, it is typical to face the challenges of overfitting or underfitting. To combat this, we utilize a regression method that imposes constraints, known as regularization, on the model's coefficient estimates, pushing them towards zero. Essentially, this technique discourages the model from becoming overly intricate or excessively flexible, thereby reducing the risk of overfitting and aiding in achieving an optimal and well-balanced model.

There are three primary regularization techniques:

L1 (Lasso) regularization,

L2 (Ridge) regularization,

Dropout regularization.

L1/Lasso Regularization: L1 regularization introduces a penalty term based on the absolute values of the model's coefficients, incentivizing certain coefficients to reach zero, thus acting as a feature selection mechanism. It is particularly useful when the goal is to identify and retain only the most significant features.

L2/Ridge Regularization: L2 regularization introduces a penalty term based on the square of the coefficients, encouraging all coefficients to be small without being reduced to zero. Ridge regularization is effective in diminishing the influence of less important features.

Dropout Regularization: This technique is employed in neural networks by randomly deactivating or "turning off" neurons during the training phase. It aids in averting overfitting by introducing randomness and decreasing reliance on any particular neuron.

Example:

Trying to build a model to predict housing prices based on various features such as size, number of bedrooms, and location. Gathering a dataset consisting of 100 houses with their corresponding features and prices.

Initially, when we train a regression model without any regularization techniques. However, when we evaluate its performance, we can notice that the model's predictions are highly sensitive to small fluctuations in the training data. It seems to be capturing noise and outliers rather than the underlying patterns in the data.

To address this issue, we need decide to incorporate L2 (Ridge) regularization into your model training process. With L2 regularization, the model is penalized for having large

coefficient values, encouraging them to stay close to zero. This helps prevent the model from becoming overly complex and reduces the risk of overfitting.

After retraining the model with L2 regularization, we will observe that its predictions are more stable and consistent, even when presented with new data. By imposing constraints on the model's coefficients, regularization has helped create a more robust and well-balanced predictive model, ultimately improving its performance and reliability.

QUESTION 2

What is the role of a loss function in a predictive model? Name two common loss functions for regression models and two common loss functions for classification models.

Answer:

First, let's delve into the definition of loss function;

The loss function, also referred to as the cost function or objective function, plays a pivotal role in predictive modeling by assessing the disparity between predicted values and actual target values. It quantifies the level of error, and the objective of model training is to minimize this loss function.

In regression models, there are common loss functions:

- Mean Square Error (MSE): MSE is extensively utilized in regression, calculating the average of squared variances between predicted and actual values. It gives heavier penalties to larger errors, thus being sensitive to outliers.

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

- Mean Absolute Error (MAE): MAE, another regression loss function, computes the average of absolute differences between predicted and actual values. It demonstrates lower sensitivity to outliers compared to MSE.

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

For classification models, the commonly used loss functions include:

- ***Binary Cross-Entropy (Log Loss):*** This loss function is popular for binary classification tasks, evaluating the difference between predicted probabilities and actual binary labels. It increases logarithmically as predictions deviate from true labels.

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

- Categorical Cross-Entropy (Log Loss): Employed in multiclass classification scenarios, this loss function measures the disparity between predicted class probabilities and actual class labels. It extends binary cross-entropy to multiple classes.

These loss functions are vital in model training as they offer a quantitative assessment of model performance. During training, adjustments are made to the model's parameters to minimize the selected loss function, thereby enhancing its ability to make accurate predictions on new, unseen data.

Example:

Trying to predict the price of a house based on its size. Collecting a small dataset consisting of only three houses:

House 1: Size = 1000 sq. ft., Price = $200,000

House 2: Size = 1500 sq. ft., Price = $250,000

House 3: Size = 2000 sq. ft., Price = $300,000

Now, we have to decide to build a simple linear regression model to predict house prices. The loss function you choose will play a crucial role in determining how well your model performs.

And, we have decided to use Mean Absolute Error (MAE) as your loss function. After training your model, you find that the average absolute difference between the predicted and actual prices is $20,000.

However, upon further examination, you realize that using Mean Square Error (MSE) might have been a better choice. Upon further calculation, the average of squared differences between predicted and actual prices, you find that it's $40,000.

This demonstrates how the choice of loss function can significantly impact the performance of the predictive model. By selecting the appropriate loss function, we can better quantify and minimize the errors in your predictions, ultimately improving the model's accuracy and reliability.

QUESTION 3

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

Answer:

When the training error is exceptionally low, it might suggest that the model has overfit the training data. Overfitting happens when the model grasps the noise and peculiarities within the training dataset rather than the broader patterns. Consequently, the model excels on the training data but falters when presented with unseen data.

Small datasets may lack the breadth to encapsulate the full range of patterns present in the data. Consequently, the model might struggle to generalize effectively to new, unseen data as it essentially memorizes the training data. This failure to generalize can result in subpar performance in real-world applications.

To summarize, while an exceedingly low training error can seem positive, it should be approached cautiously due to concerns regarding overfitting, limited generalization, and other factors. Confidence in the model should be grounded in its performance on validation or test data, as well as a thorough understanding of the dataset and model complexity.

Let's consider a case where you're supposed with a task by building a spam email classifier for a small company's email system. You gather a dataset consisting of 1000 emails, half of which are spam and the other half are legitimate. You decide to use a sophisticated machine-learning model with various hyperparameters to build the classifier.

During training, you notice that the model's training error is exceptionally low, almost reaching 100% accuracy. Initially, this might seem like a positive sign – your model is performing remarkably well on the training data.

However, upon closer inspection, we will realize that your dataset is relatively small compared to the complexity of the problem. Because of this limited amount of data, the model might be memorizing specific characteristics of the training set, including noise and outliers, rather than learning general patterns that would help it accurately classify new, unseen emails.

This phenomenon is known as overfitting. Essentially, the model becomes too specialized for the training data, making it less effective at classifying emails it hasn't seen before.

In this case, we can't fully trust the model's performance based solely on the training error. Despite achieving an exceptionally low error rate during training, there's a risk that the model won't generalize well to real-world email data. To ensure the reliability of the model, it's crucial to evaluate its performance on separate validation or test data. This will provide a more accurate assessment of its ability to classify emails accurately in a real-world setting, taking into account the challenges of overfitting and the limitations of a small dataset.

So, we cannot trust the model only if it makes good prediction on the train data. Instead, it should be applied to the unseen data and check if variance is found between the train error and validation error. If there is high variance, the model is to be retrained by reducing the complexity of the model or use some regularization techniques for the model to generalize rather than just learning the patterns in the train dataset.

QUESTION 4

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

Answer:

When the training error is exceptionally low, it might suggest that the model has overfit the training data. Overfitting happens when the model grasps the noise and peculiarities within the training dataset rather than the broader patterns. Consequently, the model excels on the training data but falters when presented with unseen data.

Small datasets may lack the breadth to encapsulate the full range of patterns present in the data. Consequently, the model might struggle to generalize effectively to new, unseen data as it essentially memorizes the training data. This failure to generalize can result in subpar performance in real-world applications.

To summarize, while an exceedingly low training error can seem positive, it should be approached cautiously due to concerns regarding overfitting, limited generalization, and other factors. Confidence in the model should be grounded in its performance on validation or test data, as well as a thorough understanding of the dataset and model complexity.

For example, let's consider a small example related to the lambda parameter in regularized linear models like Lasso or Ridge regression.

Suppose we have a dataset with features representing housing prices, and we want to predict the price based on these features. We decided to use Ridge regression, which adds a penalty term to the ordinary least squares (OLS) regression to prevent overfitting.

The lambda parameter in Ridge regression controls the strength of this penalty term. A higher lambda value means a stronger penalty, which shrinks the coefficients of the features towards zero more aggressively. On the other hand, a lower lambda value means a weaker penalty, allowing the model to fit the training data more closely.

Now, let's say we start with a very low lambda value. The model might achieve an exceptionally low training error because it's essentially memorizing the training data, capturing all the noise and idiosyncrasies. However, when we test this model on new data, it might perform poorly because it hasn't learned the broader patterns but instead fits the training data too closely.

In contrast, if we choose a higher lambda value, the model might have a slightly higher training error because it penalizes the coefficients more, leading to a smoother and more generalized model. Although the training error might not be as impressively low, this model is more likely to perform better on unseen data because it avoids overfitting and captures the underlying patterns more effectively.

PART B

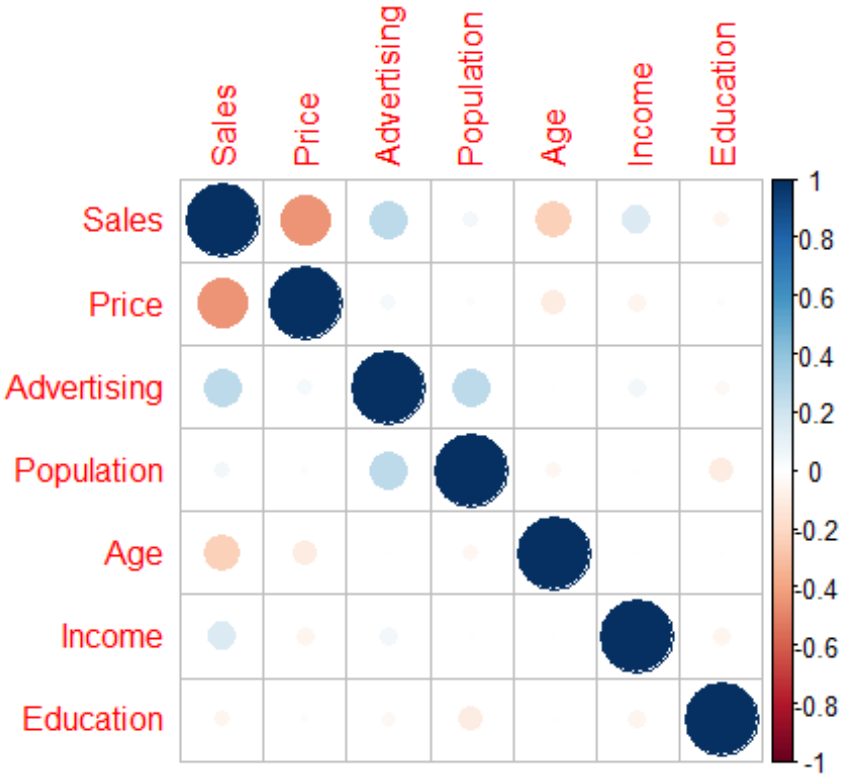
ADVANCED DATA MINING ASSIGNMENT 1

DIVYA CHANDRASEKARAN_811284790

2024-03-11

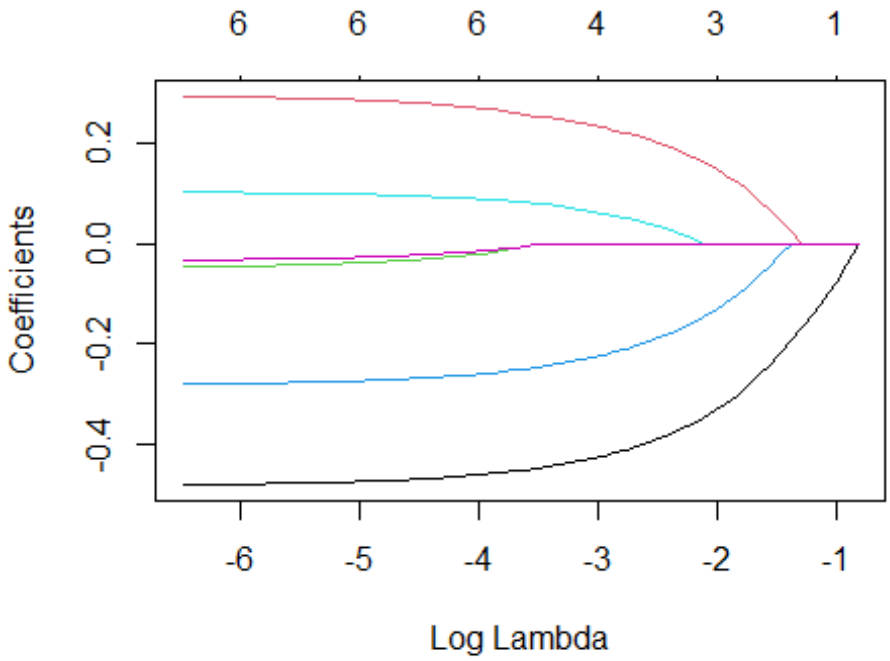
```
#PART - B

#Loading the required packages
library(class)

library(caret)

## Loading required package: ggplot2

## Loading required package: lattice

library(ggplot2)
library(ISLR)
library(corrplot)

## corrplot 0.92 loaded

library(tidyverse)

## ── Attaching core tidyverse packages ──────────────────────── tidyverse
2.0.0 ──
## ✓ dplyr     1.1.3     ✓ readr     2.1.4
## ✓ forcats   1.0.0     ✓ stringr   1.5.0
## ✓ lubridate 1.9.2     ✓ tibble    3.2.1
## ✓ purrr     1.0.2     ✓ tidyr     1.3.0
## ── Conflicts ────────────────────────────────────────
tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ✖ purrr::lift()   masks caret::lift()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all
conflicts to become errors

library(dplyr)
library(tinytex)

## Warning: package 'tinytex' was built under R version 4.3.2

library(glmnet)

## Warning: package 'glmnet' was built under R version 4.3.3

## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 4.1-8

#Loading the car seats data
Carseats <- Carseats
View(Carseats)

#Checking for null values in the dataset
print(is.null(Carseats))

## [1] FALSE

#Subsetting the data
Carseats_Filtered <- Carseats %>% select(Sales, Price, Advertising,
Population, Age, Income, Education)
summary(Carseats_Filtered)

##      Sales            Price        Advertising       Population
##  Min.   : 0.000   Min.   : 24.0   Min.   : 0.000   Min.   : 10.0
##  1st Qu.: 5.390   1st Qu.:100.0   1st Qu.: 0.000   1st Qu.:139.0
##  Median : 7.490   Median :117.0   Median : 5.000   Median :272.0
##  Mean   : 7.496   Mean   :115.8   Mean   : 6.635   Mean   :264.8
##  3rd Qu.: 9.320   3rd Qu.:131.0   3rd Qu.:12.000   3rd Qu.:398.5
##  Max.   :16.270   Max.   :191.0   Max.   :29.000   Max.   :509.0
##       Age            Income         Education
##  Min.   :25.00   Min.   : 21.00   Min.   :10.0
##  1st Qu.:39.75   1st Qu.: 42.75   1st Qu.:12.0
##  Median :54.50   Median : 69.00   Median :14.0
##  Mean   :53.32   Mean   : 68.66   Mean   :13.9
##  3rd Qu.:66.00   3rd Qu.: 91.00   3rd Qu.:16.0
##  Max.   :80.00   Max.   :120.00   Max.   :18.0

#Now, let's construct a correlation plot for the Carseats_Filtered data
corrplot(cor(Carseats_Filtered))
```

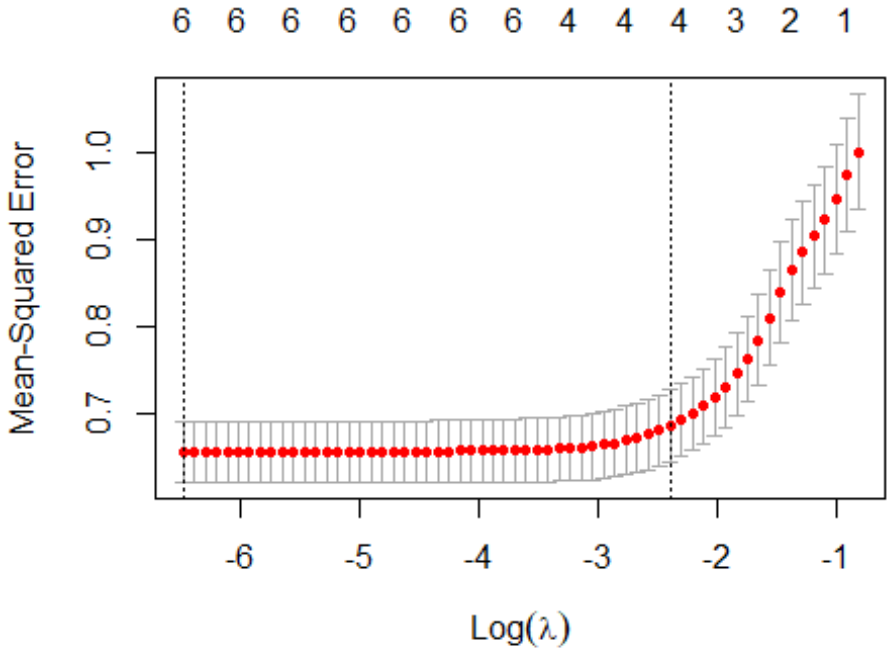


#From this we can conclude that the sales has negative correlation with Price and Age, whereas positive correlation with Advertising and Income.

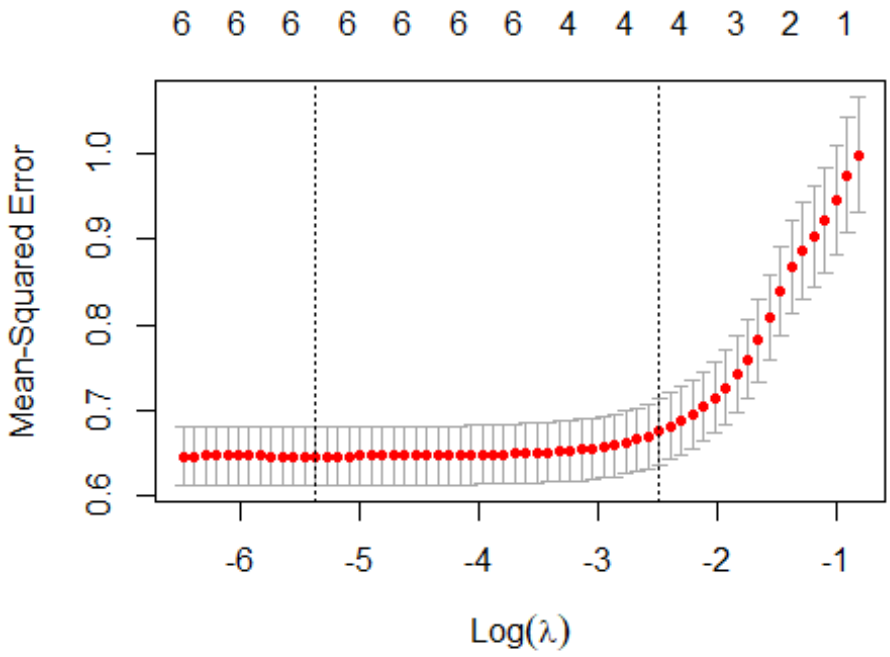
```
#Let's do normalization for this model
set.seed(1)
normalize_new_car <- scale(Carseats_Filtered)
X <- as.matrix(normalize_new_car[, c('Price', 'Advertising', 'Population',
'Age', 'Income', 'Education')])
Y <- normalize_new_car[, 'Sales']
fit.lasso <- glmnet(X, Y, alpha = 1)
plot(fit.lasso,xvar = "lambda")
```




```
plot(cv.glmnet(X, Y, alpha = 1))
```



```
#QUESTION 1
#BUILDING A LASSO REGULARIZATION 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?

cv_fit <- cv.glmnet(X, Y, alpha = 1)
plot(cv_fit)
```




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

## [1] 0.004655544
```

#Therefore, the best value of lambda is 0.00465

```
#QUESTION2
#WHAT IS THE COEFFICIENT FOR THE PRICE (NORMALIZED) ATTRIBUTE IN THE BEST
MODEL (I.E., MODEL WITH THE OPTIMAL LAMBDA)?

coef(fit.lasso, s = lambdaa)

## 7 x 1 sparse Matrix of class "dgCMatrix"
##                         s1
## (Intercept)  9.841298e-17
## Price       -4.758044e-01
## Advertising  2.889156e-01
```

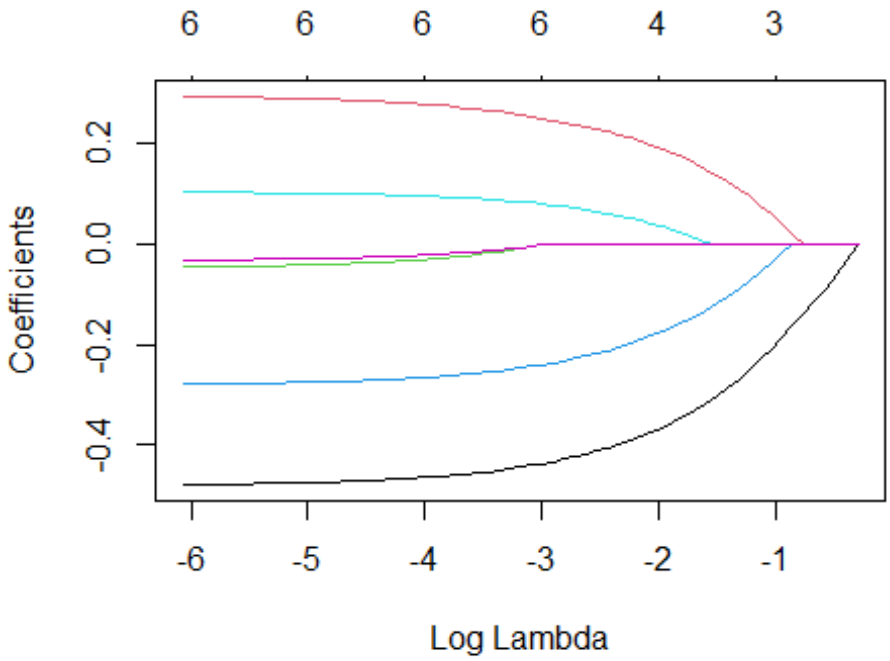
```
## Population  -4.144318e-02
## Age         -2.755668e-01
## Income       1.000095e-01
## Education   -2.893355e-02
```

#The coefficient for the "Price" (normalized) is -4.758044e-01

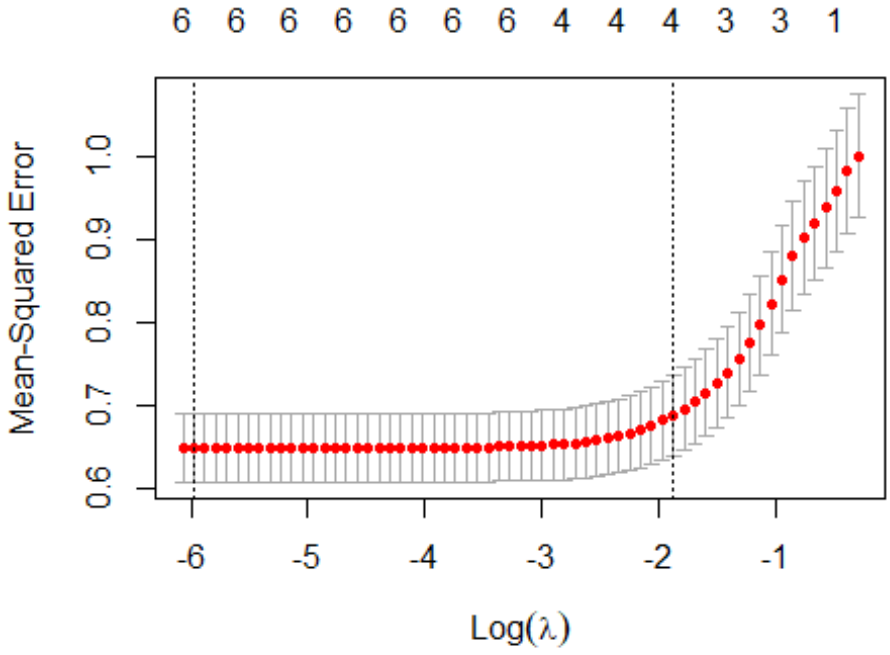
```
#QUESTION 3
#HOW MANY ATTRIBUTES REMAIN IN THE MODEL IF LAMBDA I SET TO 0.01?
#HOW IS THAT NUMBER CHANGES IF LAMBDA IS INCREASED TO 0.1?
#DO WE EXPECT MORE VARIABLES TO STAY IN THE MODEL AS WE INCREASE THE LAMBDA?

coef(fit.lasso, s = 0.01)

## 7 x 1 sparse Matrix of class "dgCMatrix"
##                         s1
## (Intercept)  9.798009e-17
## Price       -4.696889e-01
## Advertising  2.815718e-01
## Population  -3.323443e-02
## Age         -2.693300e-01
## Income       9.585212e-02
## Education   -2.330455e-02

coef(fit.lasso, s = 0.1)

## 7 x 1 sparse Matrix of class "dgCMatrix"
##                         s1
## (Intercept)  9.803050e-17
## Price       -3.691394e-01
## Advertising  1.839178e-01
## Population   .
## Age         -1.684796e-01
## Income       1.925921e-02
## Education    .
```

##As it can be seen from the above, we are not losing any attributes in the model When the Lambda value is decreased from 0.0015 to 0.01. Whereas, some of the attributes i.e. Population and Education have been lost when we increased the lambda value from 0.01 to 0.1. With an increase in the lambda value, it is likely that more properties will be lost.

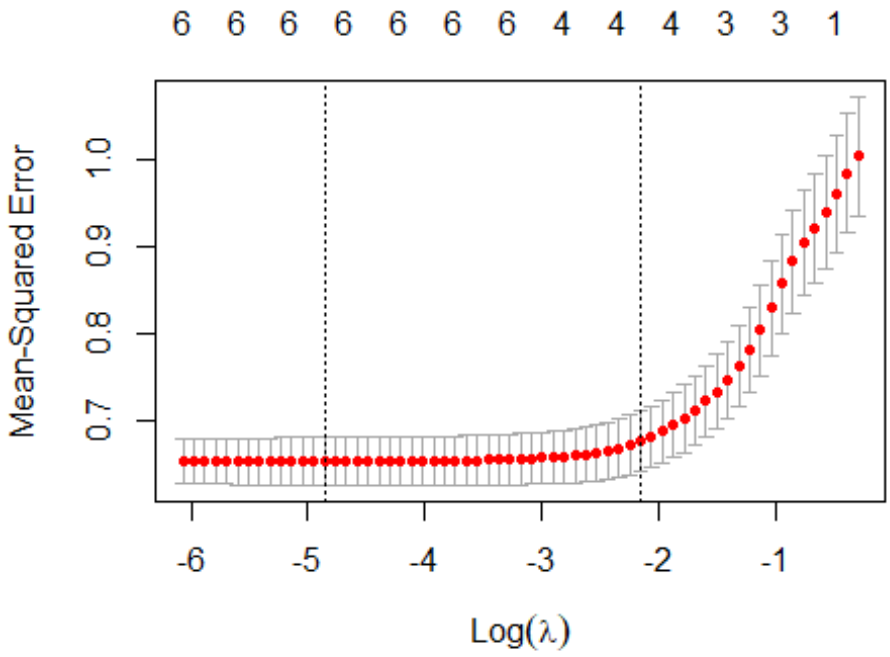
```
#QUESTION4
# BUILD AN ELASTIC NET-MODEL WITH ALPHA SET TO 0.6.
#WHAT IS THE BEST VALUE OF LAMBDA FOR SUCH A MODEL?

fit.elastic.model <- glmnet(X,Y,alpha = 0.6)
plot(fit.elastic.model, xvar = "lambda")
```




```
plot(cv.glmnet(X,Y,alpha = 0.6))
```



```
CV.Fit.elastic <- cv.glmnet(X,Y, alpha = 0.6)
plot(CV.Fit.elastic)
```




```
elastic <- CV.Fit.elastic$lambda.min
elastic

## [1] 0.007759239
```

#The best value of Lambda for an elastic model with alpha set to 0.6 is 0.00776