Assignment 1

Krishna Kumar Tavva

2023-10-22

Part-A

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

Ans:- When training a machine learning model, it's common to encounter the challenges of overfitting or underfitting. To address this, we employ a regression technique that imposes constraints, regularization, or shrinkage on the model's coefficient estimates, pushing them closer to zero. In essence, this approach discourages the model from becoming overly complex or overly flexible, which in turn mitigates the risk of overfitting and guides us in achieving an optimal and well-balanced model.

There are three main regularization techniques: L1(Lasso) regularization, L2(Ridge) regularization, and Dropout regularization.

L1/Lasso Regularization: L1 regularization introduces a penalty term that depends on the absolute values of the model's coefficients. This penalty encourages certain coefficients to reach precisely zero, essentially acting as a feature selection mechanism. L1 regularization is particularly useful when we want to identify and retain only the most important features.

L2/Ridge Regularization: L2 regularization introduces a penalty term that depends on the square of the coefficients. This penalty encourages all coefficients to be small, though not reduced to zero. Ridge regularization is effective at reducing the impact of less important features.

Dropout Regularization: It is a technique employed in neural networks where random neurons are deactivated or "turned off" during the training process. It helps to prevent overfitting by introducing randomness and reducing the reliance on any specific neuron

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.

Ans:-The loss function, also known as the cost function or objective function, plays a crucial role in predictive modeling. It evaluates the distinction between the predicted values and the actual target values. The loss function quantifies the extent of error. The goal of the model training is to minimize the loss function.

Common loss functions for regression models:

Mean Square Error (MSE): MSE is a widely used loss function for regression. It measures the average of the squared differences between predicted and actual values. It penalizes larger errors more heavily, making it sensitive to outliers.

Mean Absolute Error (MAE): MAE is another loss function for regression. It measures the average of the absolute differences between predicted and actual values. MAE exhibits lower sensitivity to outliers in contrast to MSE.

Common loss functions for classification models:

Binary Cross-Entropy (Log Loss): Binary cross-entropy is a prevalent choice for binary classification tasks. It measures the disparity between the predicted probabilities and the actual binary labels. It is a logarithmic loss function that increases as predictions diverge from true labels.

Categorical Cross-Entropy (Log Loss): Categorical cross-entropy is employed in multiclass classification scenarios. It computes the difference between predicted class probabilities and the actual class labels. It is a generalization of binary cross-entropy to multiple classes.

These loss functions are essential for model training because they provide a quantitative measure of how well the model is performing. During training, the model's parameters are adjusted to minimize the chosen loss function, which, in turn, improves its ability to make accurate predictions on new, unseen data.

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

Ans:-When the training error is extremely small, it may indicate that the model has overfit the training data. Overfitting occurs when the model learns the noise and idiosyncrasies in the training dataset, rather than general patterns. This results in a model that performs exceptionally well on the training data but poorly on unseen data.

Small datasets can be insufficient to capture the full diversity of patterns in the data. The model may not be able to generalize well to new, unseen data because it has essentially memorized the training data. This lack of generalization can lead to poor performance in real-world applications.

In summary, while an extremely low training error can be a positive sign, it should be interpreted cautiously in the context of overfitting, limited generalization, and other considerations. Trust in the model should be based on its performance on validation or test data and an understanding of the dataset and model complexity.

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

Ans:-In regularized linear models like Lasso and Ridge regression, the lambda parameter, plays a critical role in controlling the level of regularization applied to the model. These models aim to strike a balance between fitting the data well and preventing overfitting. Here's the role of the lambda parameter in these models:

Controlling Model Complexity:
Lasso (L1 Regularization): In Lasso regression, lambda controls the amount of L1 regularization applied to the model. L1 regularization encourages sparsity in the model by adding a penalty term based on the absolute values of the coefficients. As lambda increases, more coefficients are pushed towards zero, effectively performing feature selection. Larger lambda values lead to simpler models with fewer non-zero coefficients.

Ridge (L2 Regularization): In Ridge regression, lambda controls the amount of L2 regularization applied to the model. This includes a penalty term based on the square of the coefficients. It promotes coefficients to be modest in size while ensuring they remain non-zero. Increasing lambda in Ridge regression results in smaller coefficient values, effectively reducing the impact of less important features.

Trade-Off between Bias and Variance:
By adjusting the lambda parameter, We have the ability to manage the balance between bias and variance in the model. A smaller lambda enables the model to closely fit the training data, reducing bias while potentially raising variance, which may result in overfitting. A larger lambda increases bias but reduces variance, promoting model generalization and preventing overfitting.

Preventing Overfitting:
lambda is a regularization hyperparameter that helps prevent overfitting. Overfitting occurs when a model becomes too complex and fits the training data's noise, resulting in poor performance on new, unseen data. By setting an appropriate value for lambda, we can constrain the model's complexity, making it more likely to generalize well.

Cross-Validation and Hyperparameter Tuning:
Determining the optimal lambda value typically involves cross-validation. we can try different values of lambda and evaluate the model's performance using techniques like k-fold cross-validation. The lambda value that results in the best performance on the validation set is selected as the model's hyperparameter.

In summary, the lambda parameter in regularized linear models allows we to control the amount of regularization applied to the model. Adjusting lambda is a crucial step in finding the right balance between fitting the training data and preventing overfitting. The selection of lambda relies on the particular dataset, feature attributes, and the performance criteria of the model. Regularization, with the help of lambda, is a powerful tool for building more robust and generalizable linear models.

Reference:-
https://www.analyticsvidhya.com/blog/2022/08/regularization-in-machine-learning/
https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a
https://www.geeksforgeeks.org/regularization-in-machine-learning/
https://www.simplilearn.com/tutorials/machine-learning-tutorial/regularization-in-machine-learning

Part-B

#Loading Required Packages

```
library(class)
```

```
## Warning: package 'class' was built under R version 4.2.3
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.2.3
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.2.3
```

```
## Loading required package: lattice
```

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

```
## corrplot 0.92 loaded
```

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.2.3
```

```
## Warning: package 'tibble' was built under R version 4.2.3
```

```
## Warning: package 'readr' was built under R version 4.2.3
```

```
## Warning: package 'dplyr' was built under R version 4.2.3
```

```
## Warning: package 'lubridate' was built under R version 4.2.3
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.1     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v lubridate 1.9.2     v tibble    3.2.1
## v purrr     1.0.1     v tidyr     1.3.0
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x purrr::lift()   masks caret::lift()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```
library(dplyr)
library(tinytex)
```

```
## Warning: package 'tinytex' was built under R version 4.2.3
```

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.2.3
```

```
## Loading required package: Matrix
```

```
## Warning: package 'Matrix' was built under R version 4.2.3
```

```
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
## Loaded glmnet 4.1-7
```

#Loading Carseats data

```
data <- Carseats
view(data)
```

#checking for null values

```
print(is.null(data))
```

```
## [1] FALSE
```

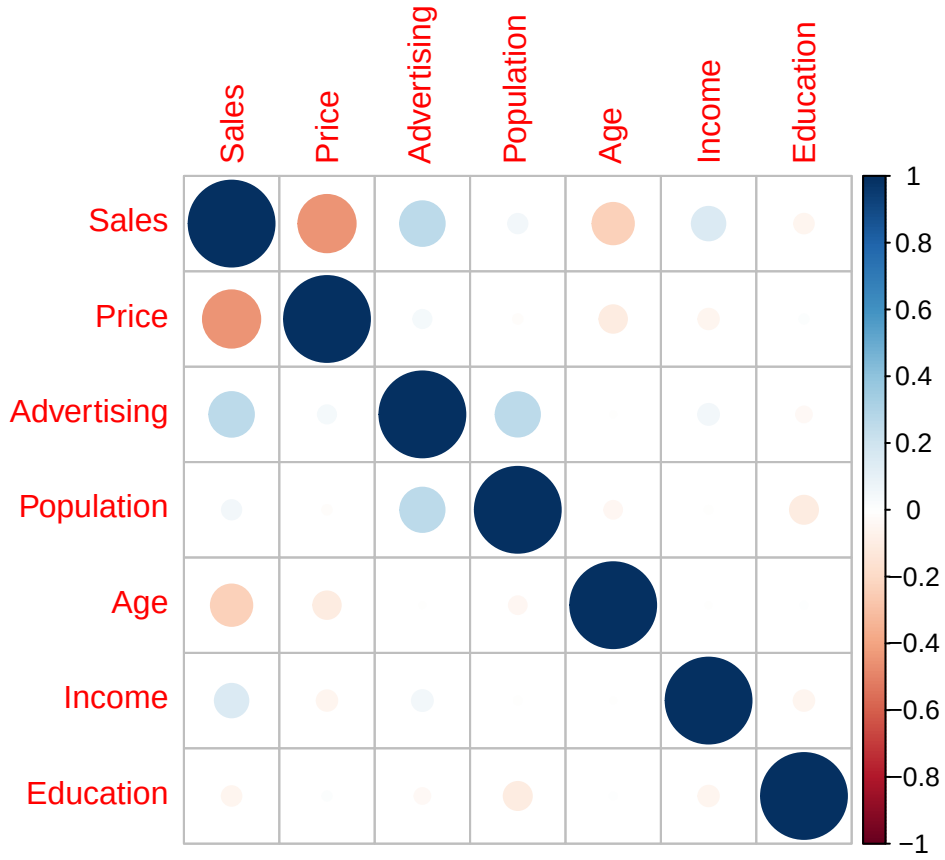
#Subsetting data

```
new_data <- data %>% select(Sales, Price, Advertising, Population, Age, Income, Education)
summary(new_data)
```

```
##      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
```

#Correlation Plot

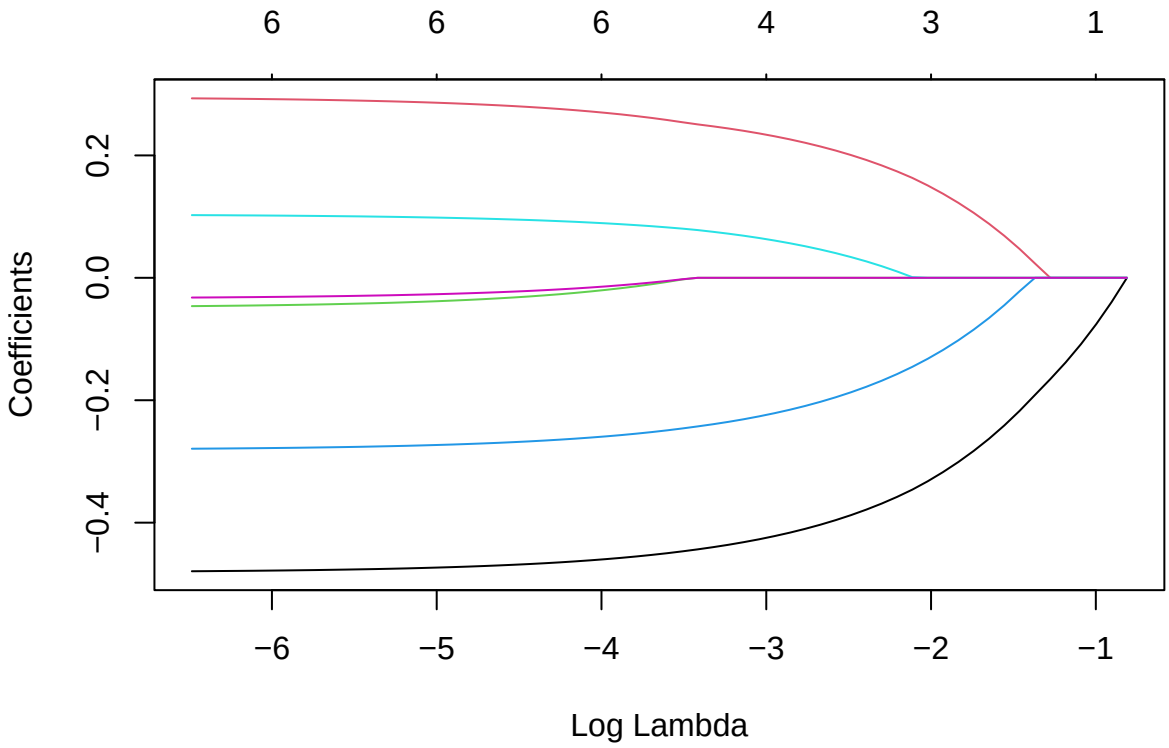
```
corrplot(cor(new_data))
```



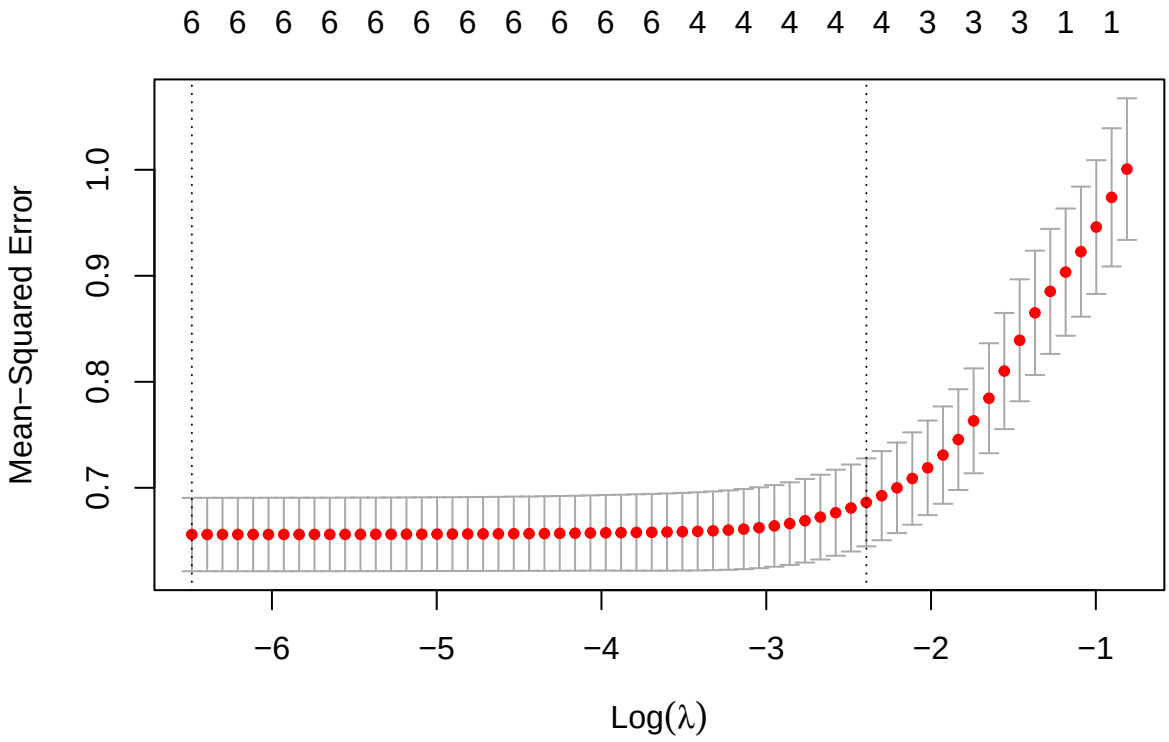
#Sales has negative correlation with Price and Age, whereas positive correlation with Advertising and Income.

#Normalization

```
set.seed(1)
Normalized_new_Data <- scale(new_data)
X <- as.matrix(Normalized_new_Data[ , c('Price','Advertising',
'Population','Age','Income','Education')])
Y <- Normalized_new_Data[,'Sales']
fit.lasso <- glmnet(X,Y,alpha = 1)
plot(fit.lasso,xvar = "lambda")
```

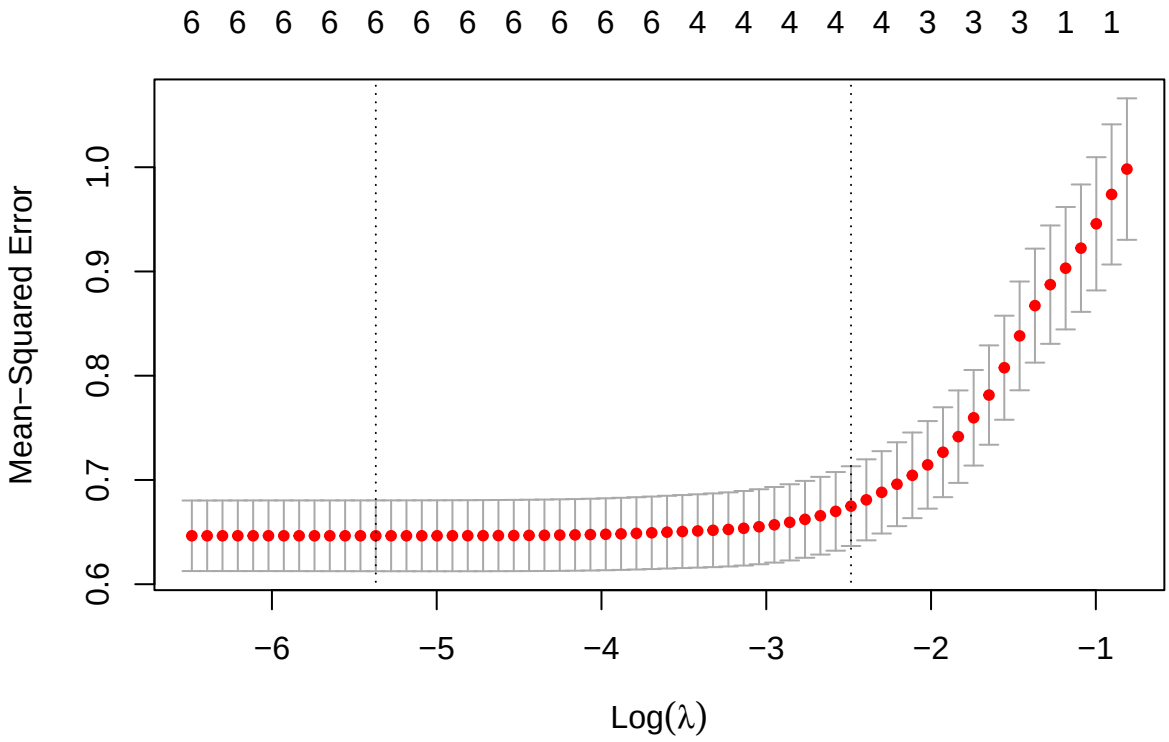



```
plot(cv.glmnet(X,Y,alpha = 1))
```



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?

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




```
lambda <- CV_Fit$lambda.min
lambda
```

```
## [1] 0.004655544
```

#The best value of lambda is 0.00465

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

```
coef(fit.lasso, s = lambda)
```

```
## 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

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 we expect more variables to stay in the model (i.e., to have non-zero coefficients) as we increase 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
```

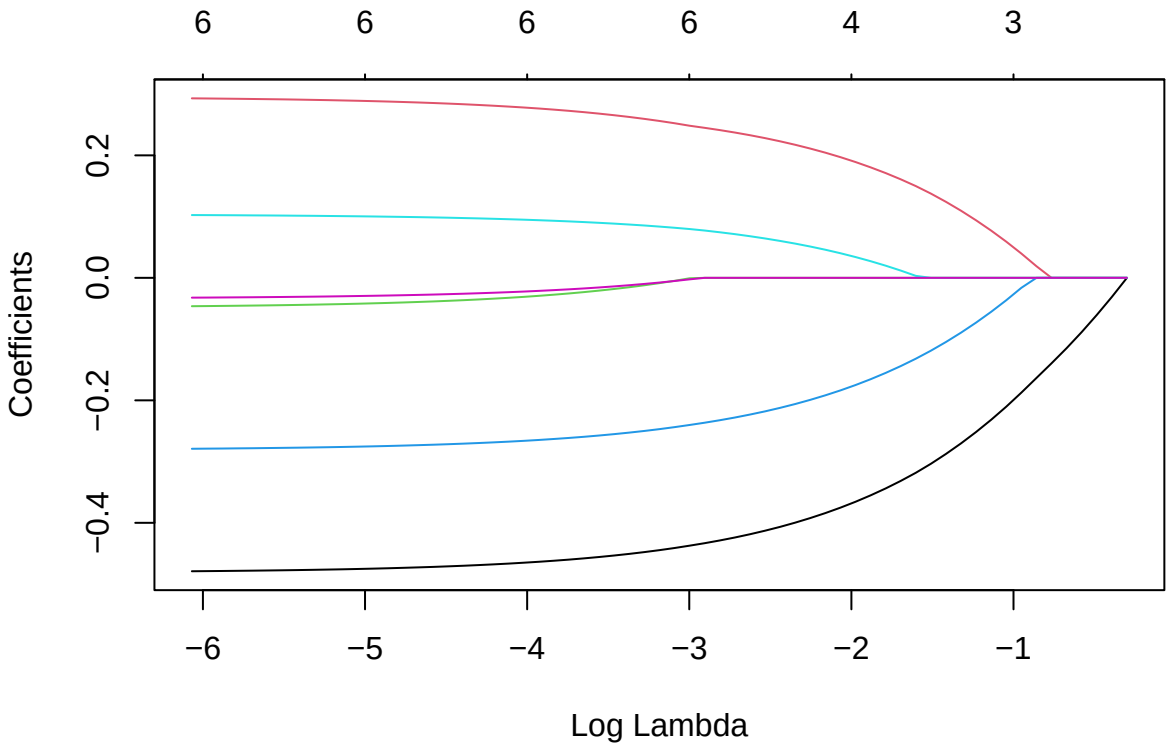
```
coef(fit.lasso, s = 0.1)
```

```
## 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    .
```

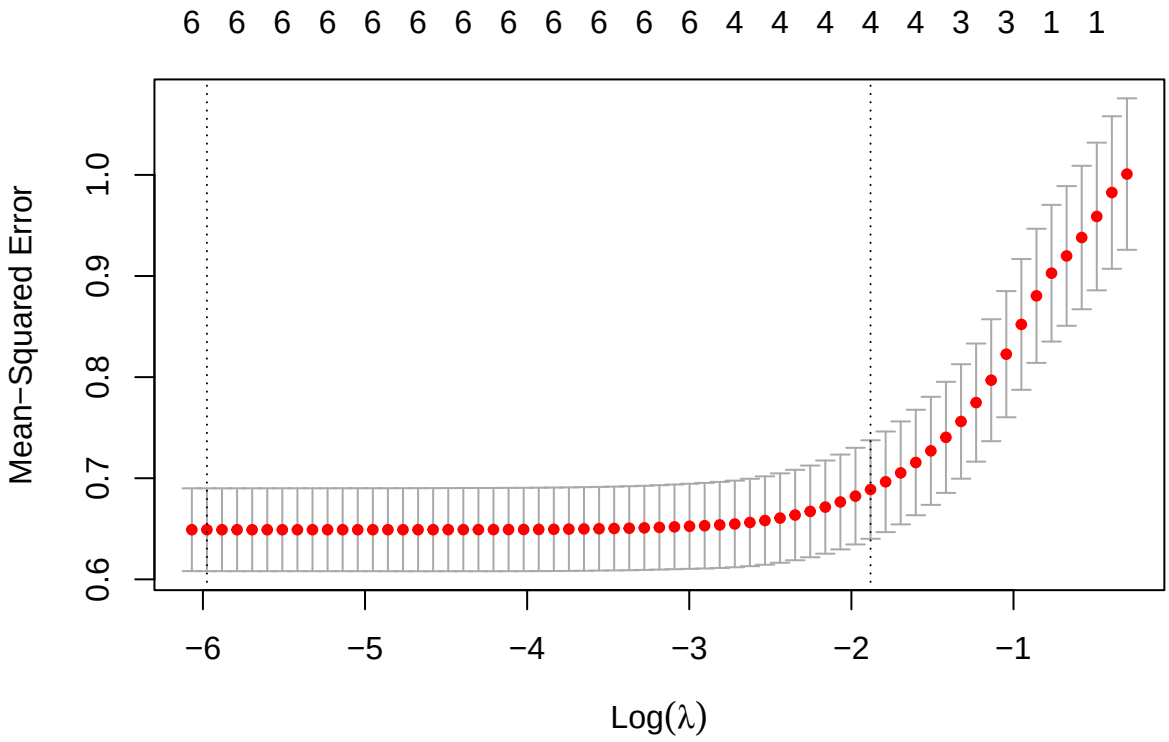
#As it can be seen from the above, we are not losing any attributes in the model When the Lambda value is decreased from 0.0015 to 0.01. Whereas, some of the attributes i.e. Population and Education have been lost when we increased the lambda value from 0.01 to 0.1. With an increase in the lambda value,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

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

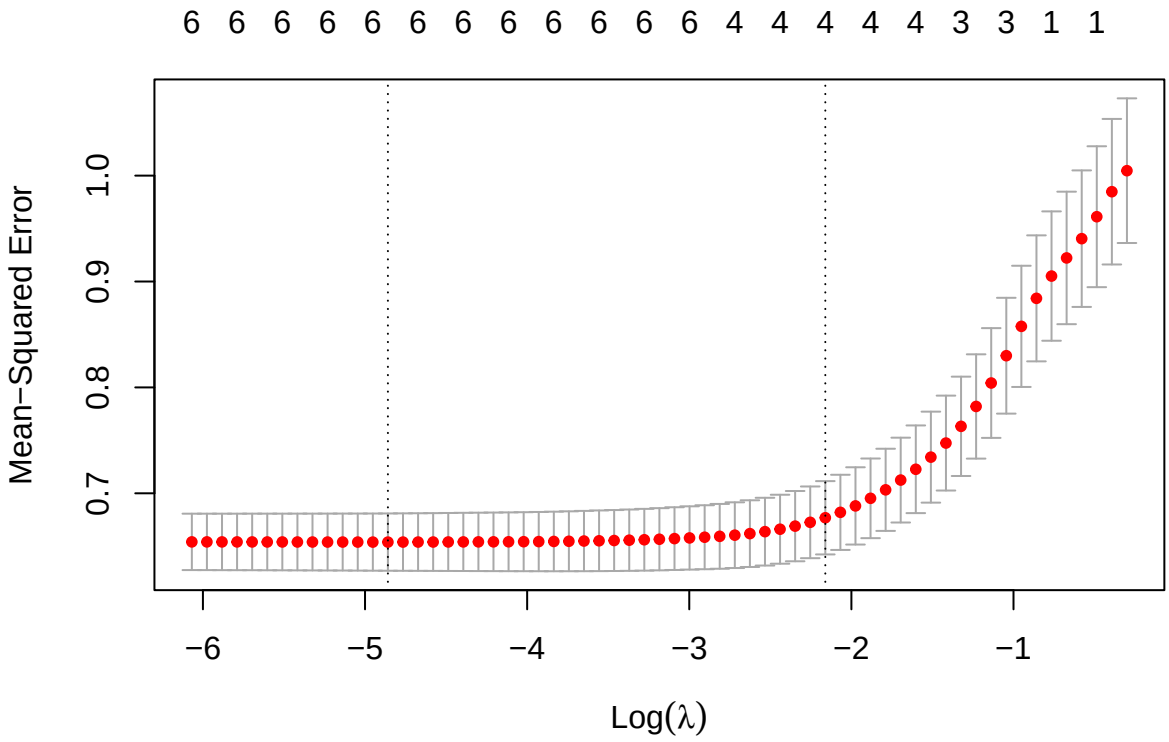
```
fit.elastic <- glmnet(X,Y,alpha = 0.6)
plot(fit.elastic,xvar = "lambda")
```




```
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