ADM_Assignment_1

Vdodda

03/04/2023

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

Regularization: Each machine learning model aims to discover patterns from training data and generalize them to accurately predict data that hasn't been seen before. Generalization, then, refers to a model's capacity to respond to fresh data. The model's lack of generalization has a variety of factors. One of them is excessive model fitting. Overfitting occurs when a model becomes so complex that it learns the detail and noise in the training data to the point that it has a detrimental effect on the model's performance on fresh data. In other words, the model picks up on the noise or random oscillations in the training data and learns them as ideas. Hence, the estimator's variance has grown.

By making the model less complicated, regularization is employed as an approach to improve the model's ability to generalize. Regularization makes an effort to make a model more effective by making it simpler. Regularization penalizes the model by reducing its complexity and simplification. Machine learning models are tuned via regularization to reduce the adjusted loss function and avoid overfitting or underfitting. We can properly fit our machine learning model on a particular test set using regularization, which lowers the mistakes in the test set. There are several sorts of regularization procedures, each with its own set of advantages and disadvantages.

L1 regularization or Lasso Regularization: It introduces a penalty term proportional to the absolute value of the model's weights. This encourages sparse solutions, in which just a fraction of the features is incorporated in the final model. L1 regularization can be effective when the dataset contains a large number of characteristics, but only a few of them are likely to be significant.

L2 regularization or Ridge Regularization: It introduces a penalty term proportional to the square of the model's weights. This encourages lighter weights generally, which can assist to minimize overfitting. L2 regularization can be beneficial when the dataset contains a large number of characteristics, all of which are potentially important. Dropout Regularization: This is a strategy in which random units in the model are momentarily "dropped out" or ignored during training, thus decreasing the model's capacity and inhibiting unit co-adaptation. When the dataset is huge and complicated, and the model has a high capacity, dropout regularization can be effective.

To summarize, regularization is a strategy for preventing overfitting in predictive models by introducing restrictions that encourage the model to learn only the most significant elements of the input. The regularization approach utilized is determined by the specific situation at hand as well as the characteristics of the data being used.

2. 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.

Loss functions quantify how far an estimated value provided by the predictive model differs from its real value. They specify an objective against which the model's performance is measured, and the parameters learned by the model are selected by minimizing a predefined loss function. Therefore, the loss function is the function that computes the difference between the algorithm's present output and its predicted output. It is a tool for assessing how well your algorithm models data. The loss is reduced further by modifying the model parameters until the lowest feasible loss is attained. These are some examples of Loss functions and how they operate.

For regression models, the following loss functions are commonly used:

Mean Squared Error (MSE): Mean Squared Error is the average of the squared differences between the actual and the predicted values. Mean Absolute Error (MAE): Mean Absolute Error is a basic yet robust loss function that is used in regression models. Because of the occurrence of outliers, variables in regression problems may not be precisely Gaussian (values that are very different from the rest of the data). In such instances, Mean Absolute Error would be an excellent choice because it does not take into account the direction of the outliers (unrealistically high positive or negative values). MAE computes the average sum of the absolute discrepancies between the actual and anticipated values.

Classification loss functions that are commonly used:

Binary Cross Entropy: This is the most popular loss function for two-class classification tasks. The seemingly out-of-place term "entropy" has a statistical meaning. Entropy measures the unpredictability of the information being processed, whereas cross-entropy measures the difference in randomness between two random variables.

If the divergence of the predicted probability from the actual label increases, the cross-entropy loss increases.

Categorical Cross-Entropy Loss:

Categorical Cross Entropy Loss is just Binary Cross Entropy Loss multiplied by the number of classes. One prerequisite for using the categorical cross-entropy loss function is that the labels be encoded only once. As a result of the vector's other members being multiplied by zero, just one element will be non-zero. This attribute is extended to a SoftMax activation function.

To summarize, the loss function is an important part of a predictive model since it quantifies the difference between expected and actual outputs and is used to train the model by altering its parameters. The loss function to be utilized is determined by the task at hand and the type of model being employed.

3. 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.

When developing a prediction model with a training set, it should be tested on both the training and validation sets. The model is deemed to be a good prediction model if both the train error and the validation error (error on data not observed by the model) are minimal. However, without more examination, it is not advisable to entirely trust such a model, as there are various reasons why the model may not be trustworthy.

When the model is underfitted, it indicates that it cannot correctly reflect the connection between the input and output variables. As a result, the performance of such a model is lower on both the training and validation sets. As a result, we must raise the model's complexity such that it correctly depicts the link between the inputs and the output variables.

When the model is overfitted, it indicates that it performs extremely well on the training set with very high train accuracy, but when applied to the validation set, the error is very high with a big variation between the train error and the validation error. This suggests that the model has gotten too sophisticated, learning the train data with all of the noise and outliers and becoming sensitive to even little changes in the input variables. As a result, the goal of any machine learning model, Generalization, is not met, causing the model to perform very poorly on unknown data. Overfitting occurs most frequently when the train data is little and the model's complexity is high.

The model with several hyperparameters in the presented example is generated using a rather modest dataset.

As a result, there is a strong likelihood that the model will be overfitted. As previously stated, a model is deemed to be a good prediction model only when it performs well on both train and unseen data. Because the provided model has a high likelihood of overfitting and a large variance, it performs well on the training set but is likely to perform badly on unknown data owing to overfitting.

As a result, we cannot trust the model until it generates accurate predictions on train data. Instead, it should be applied to unseen data to see whether there is any variance between the train error and the

validation error. If the variance is excessive, the model should be retrained by lowering its complexity or using regularization techniques to allow the model to generalize rather than simply learning the patterns in the train dataset.

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

Lambda is a hyperparameter that is utilized in linear models such as Lasso or Ridge Regression for Regularization, which is used to minimize the model's complexity.

How Lambda works in Lasso and Ridge Regression to regularize the model:

Linear regression models attempt to lower the loss function in order to arrive at the best weights for each attribute. Yet, when the model is overfitted and unable to generalize adequately, we employ regularization to punish it. This is accomplished by including an additional term for the loss function, the value of which is proportional to the Lambda value. As we increase the Lambda value, the total loss value increases, and the model tries to reduce the coefficients of the parameters to find the best answer. The theory is that by decreasing or regularizing the coefficients, prediction accuracy, variance, and model interpretability may all be improved.

In ridge regression, we apply a penalty via a tuning parameter lambda that is determined using cross-validation. The goal is to reduce the fit by reducing the residual sum of squares and applying a shrinkage penalty. The shrinkage penalty is lambda times the sum of the squares of the coefficients, therefore big coefficients are punished. The bias remains constant as lambda increases, but the variance decreases. The disadvantage of the ridge is that it does not allow you to pick variables. It incorporates all the variables into the final model.

The penalty in lasso is the total of the absolute values of the coefficients. When lambda is big enough, lasso decreases the coefficient estimates towards zero and has the effect of setting variables precisely equal to zero, but ridge does not. As a result, lasso accomplishes variable selection in the same way as the best subset selection technique does. Cross-validation is used to choose the tuning parameter lambda. When lambda is small, the resulting estimates are effectively least squares. When lambda rises, shrinkage happens, allowing variables that are at zero to be discarded. Hence, one key benefit of the lasso is that it combines shrinkage with variable selection.

#PART B

```
#Loading Required Packages
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)
```

```
## Warning: package 'glmnet' was built under R version 4.2.2
```

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

```
## Warning: package 'Matrix' was built under R version 4.2.2
```

```
## Loaded glmnet 4.1-6
```

```
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?

```
# Collecting all of the input attributes and scaling them into Carseats_Filtered.
Carseats_Filtered<- Carseats %>% select( "Price", "Advertising", "Population", "Age", "Income", "Educat
a <- as.matrix(Carseats_Filtered)
b <- Carseats[,1]

# Let us perform a basic linear regression on the data to determine the coefficients and R-squared valu

c =lm(b~a)
summary(c)
```

```
##
## Call:
## lm(formula = b ~ a)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.1113 -1.5385 -0.1214  1.4339  6.5244
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   7.49632    0.11258  66.584  < 2e-16 ***
## aPrice       -1.35877    0.11369 -11.951  < 2e-16 ***
## aAdvertising  0.83400    0.11734   7.108 5.60e-12 ***
## aPopulation  -0.13723    0.11774  -1.166   0.2445
## aAge         -0.79358    0.11345  -6.995 1.15e-11 ***
## aIncome       0.29267    0.11335   2.582   0.0102 *
## aEducation   -0.09556    0.11356  -0.841   0.4006
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.252 on 393 degrees of freedom
## Multiple R-squared:  0.3739, Adjusted R-squared:  0.3643
## F-statistic: 39.11 on 6 and 393 DF,  p-value: < 2.2e-16
```

```
mean(c$residuals^2)
```

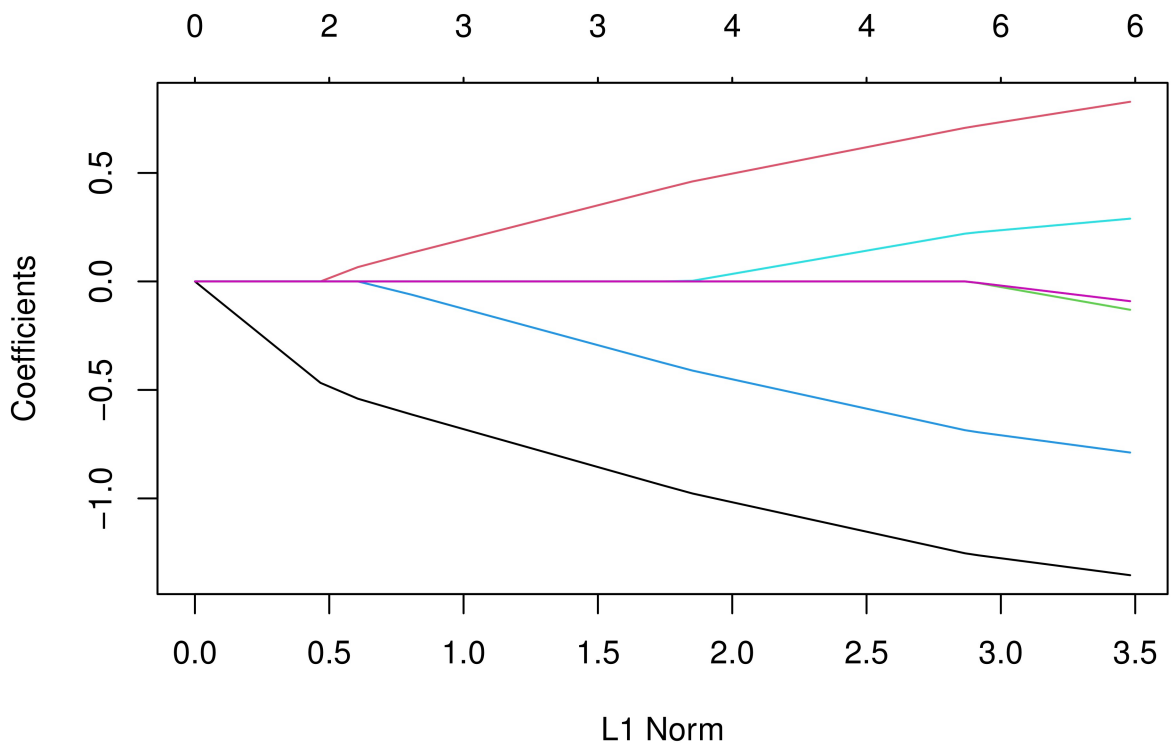
```
## [1] 4.981366
```

As a consequence of the foregoing findings, we can conclude that the provided variables explain only 36.43% of the variation in the dependent variable (Sales). Now, let's apply Lasso Regression on the same data and see if we can enhance the R-Squared value.

```
fit <- glmnet(a, b )

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)
```

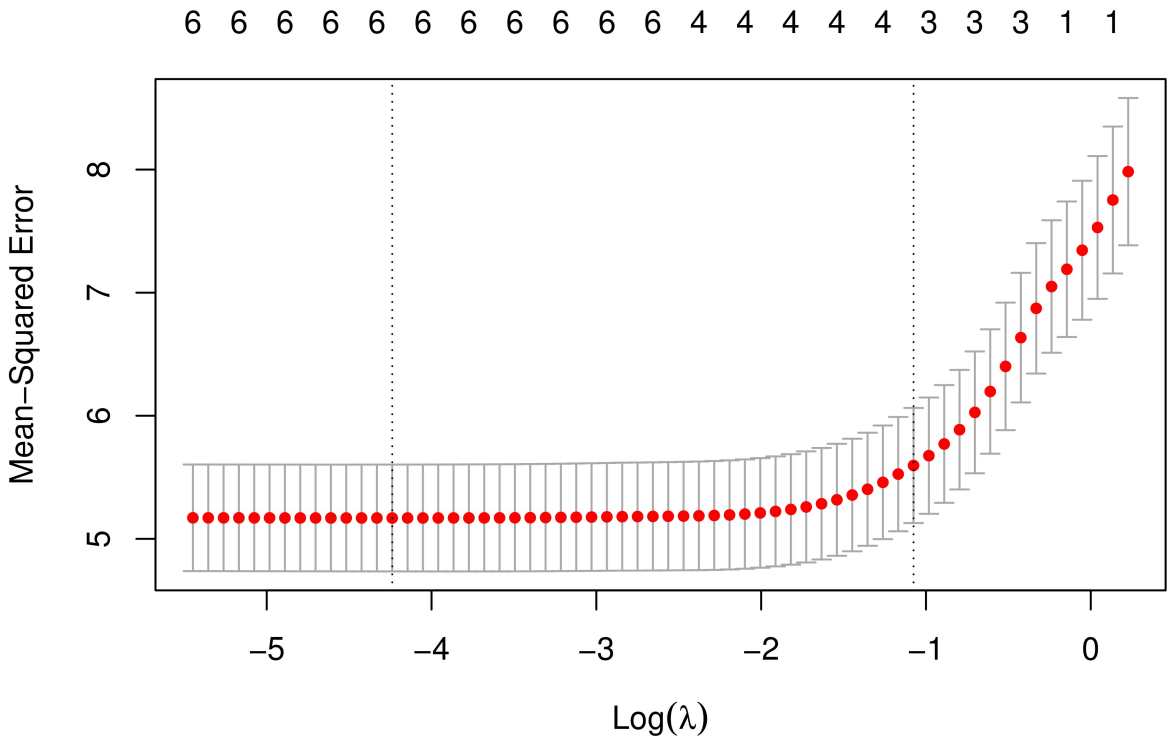
```
##
## Call:  glmnet(x = a, y = b)
##
##    Df  %Dev  Lambda
## 1   0  0.00 1.25500
## 2   1  3.36 1.14400
## 3   1  6.15 1.04200
## 4   1  8.47 0.94940
## 5   1 10.39 0.86500
## 6   1 11.99 0.78820
## 7   2 14.62 0.71820
## 8   3 18.08 0.65440
## 9   3 21.12 0.59620
## 10  3 23.64 0.54330
## 11  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
```

```
c_f <- cv.glmnet(a, b, alpha = 1)

# finding the minimum lambda value

best_lambda <- c_f$lambda.min

best_lambda
```

```
## [1] 0.01442968
```

```
plot(c_f)
```



Hence, based on the data above, we can see that there is only 37.38% variation in the target variable, sales with regularization, and a best lambda value of 0.0043.

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

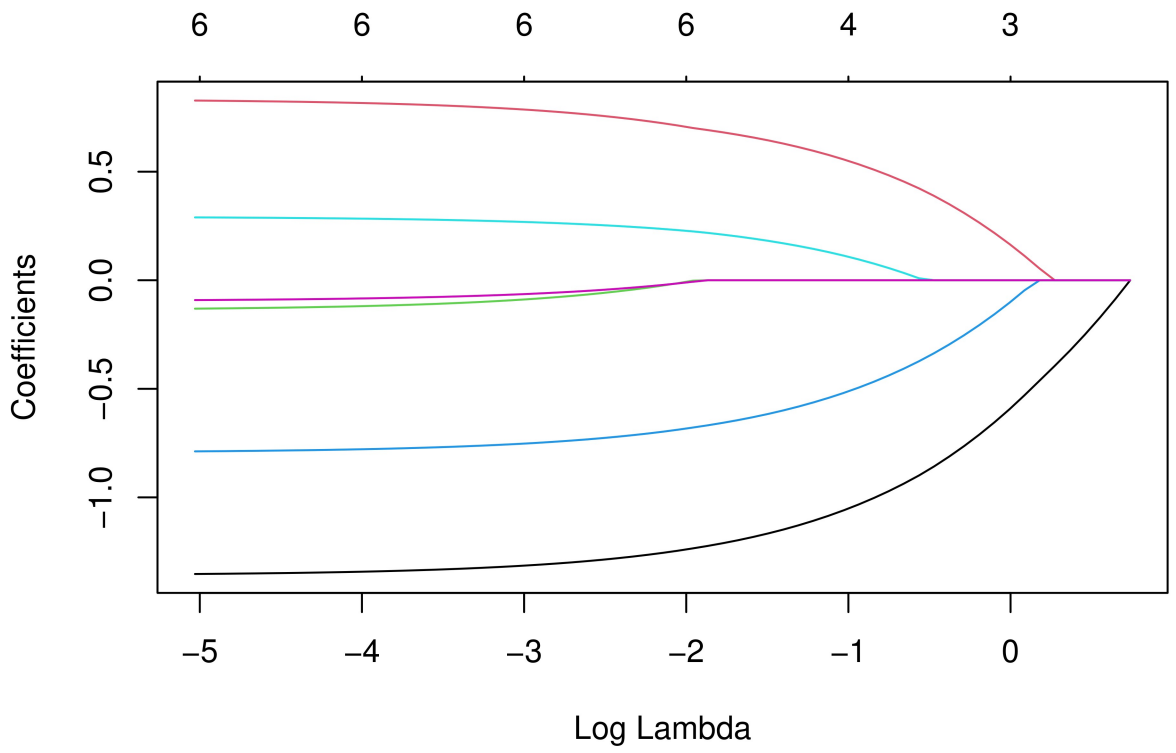
```
best_model <- glmnet(a, b, alpha = 1, lambda = best_lambda)

coef(best_model)
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept)  7.49632500
## Price       -1.34226545
## Advertising  0.81417642
## Population  -0.11507405
## Age         -0.77673725
## Income       0.28144029
## Education   -0.08036155
```

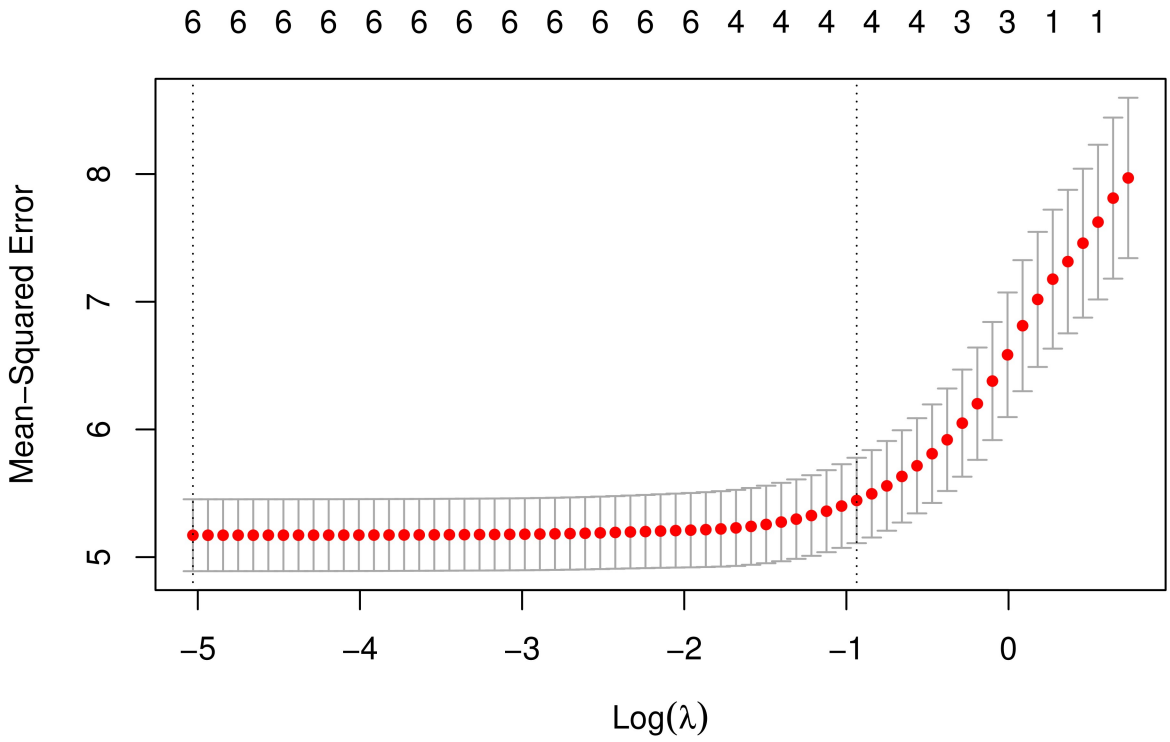
The coefficient of the Price attribute with the best lambda value is -1.35384596.

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?

```
f_e <- glmnet(a,b, alpha=0.6)
plot(f_e, xvar = "lambda")
```



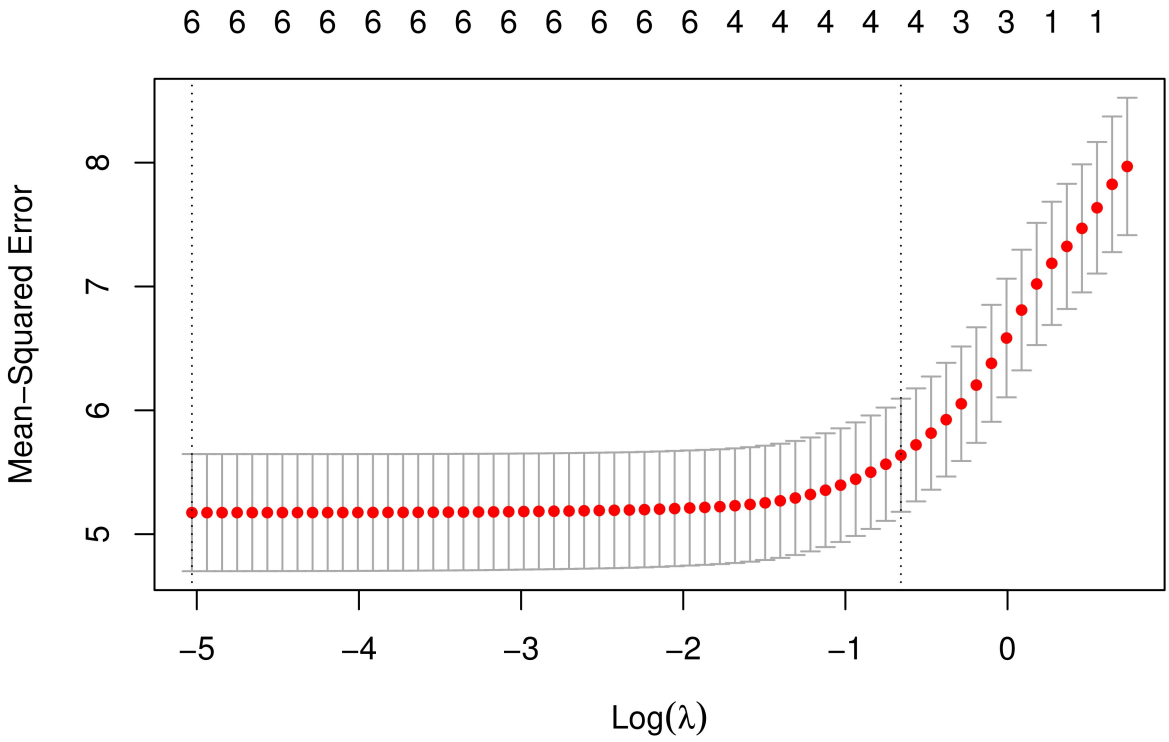

```
plot(cv.glmnet(a,b,alpha=0.6))
```



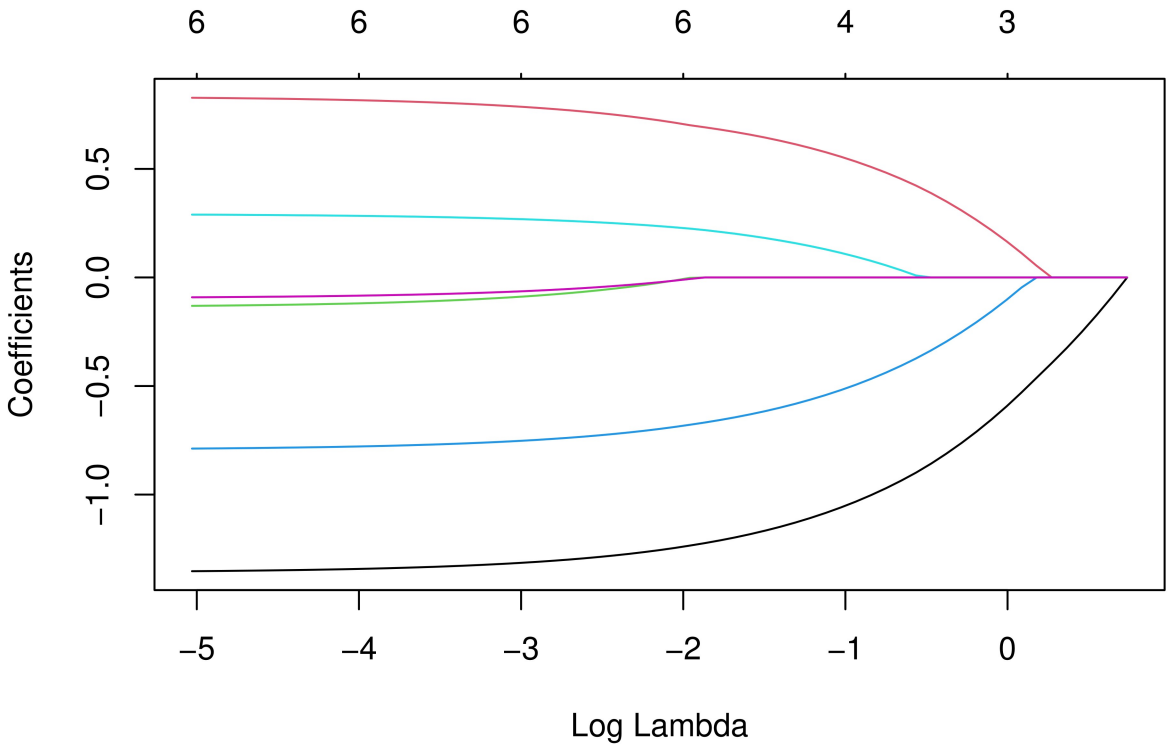

```
#Using K-fold validation to select the best lambda.
e <- cv.glmnet(a,b, alpha=0.6)
#Finding a lambda value that minimizes the test.
best_lambda <- e$lambda.min
best_lambda
```

```
## [1] 0.006538062
```

```
plot(e)
```



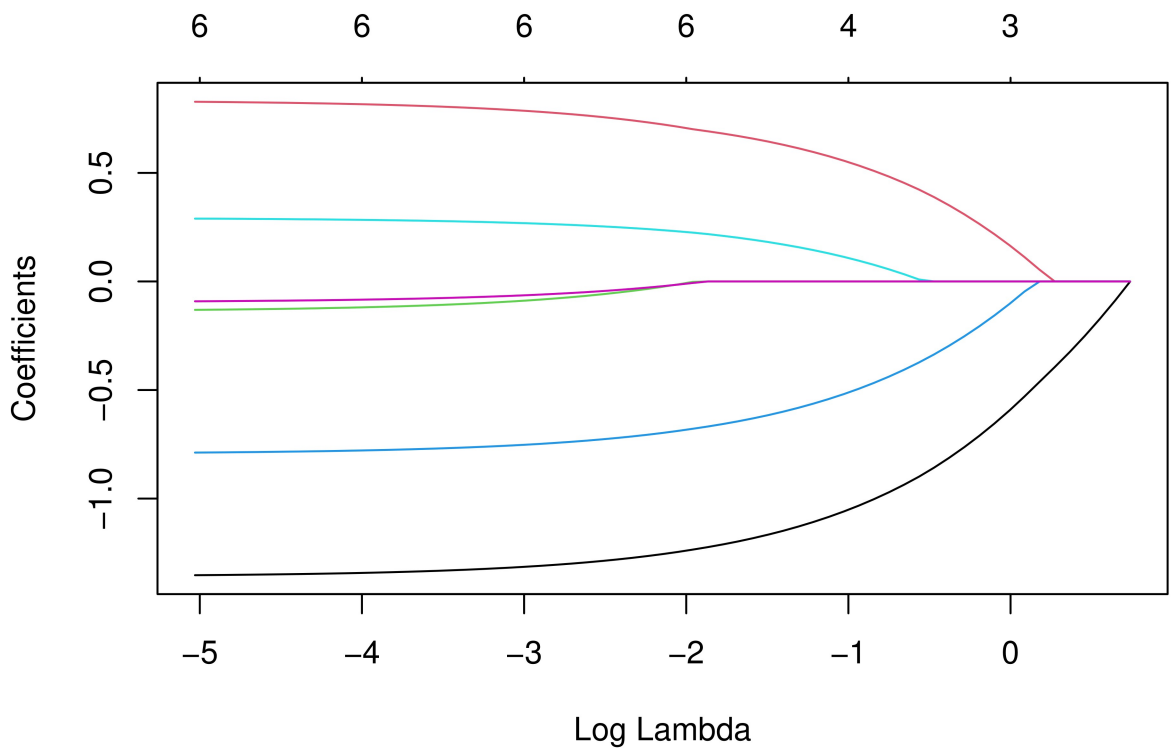

```
fit.e <- glmnet(a,b,alpha=0.6)
plot(fit.e, xvar="lambda")
```



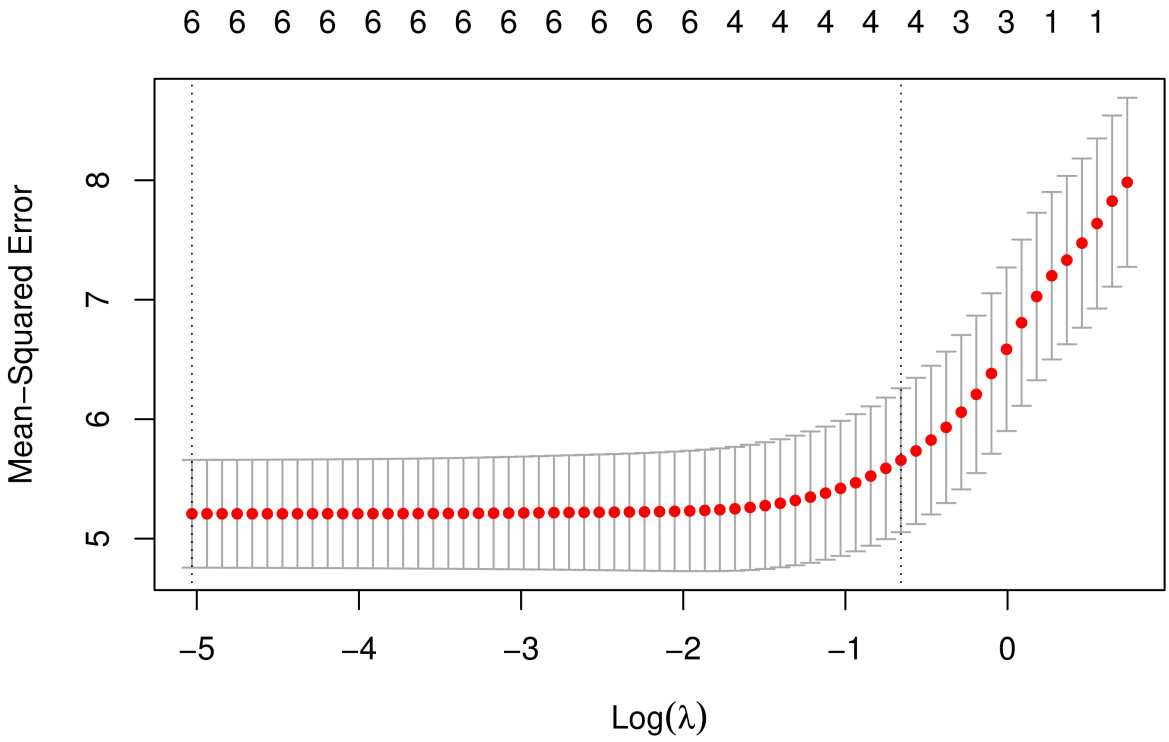
##The best lambda value for such elastic-net model is .00653

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

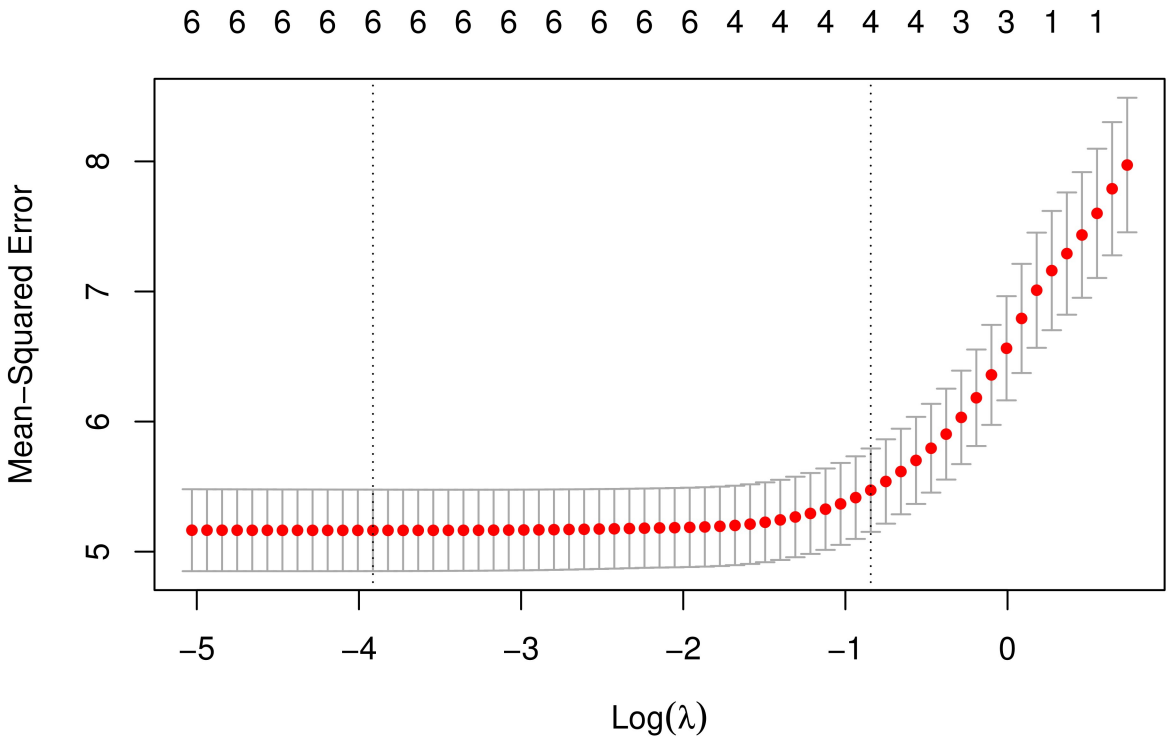
```
f.e <- glmnet(a,b,alpha = 0.6)
plot(f.e,xvar = "lambda")
```




```
plot(cv.glmnet(a,b,alpha = 0.6))
```




```
cv.f.e <- cv.glmnet(a,b, alpha = 0.6)
plot(cv.f.e)
```




```
ec <- cv.f.e$lambda.min
ec
```

```
## [1] 0.0199663
```

For an elastic model with alpha set to 0.6, the optimum Lambda is 0.0023.