ADM_Assignment_1

Mudiga

04/04/2023

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

Regularization:

Finding a lambda value that minimizes the test. Each machine learning model seeks to uncover patterns in training data and generalize them to reliably predict data that hasn't been seen before. The capacity of a model to adapt to new data is referred to as generalization. There are several reasons for the model's lack of generality. One of them is excessive model fitting. Overfitting happens when a model grows so sophisticated that it learns the detail and noise in the training data to the point that it degrades the model's performance on new data. In other words, the model detects noise or random oscillations in the training data and stores them as ideas. As a result, the variance of the estimate has increased.

Regularization is used to increase the model's capacity to generalize by making the model less difficult. Regularization attempts to make a model more effective by simplifying it. Regularization penalizes the model by simplifying and lowering its complexity. Regularization is used to optimize machine learning models in order to lower the adjusted loss function and avoid overfitting or underfitting. We may adapt our machine learning model to a specific test set via regularization, which reduces the number of errors in the test set.

There are several types of regularization techniques, each with its own set of benefits and drawbacks.

L1 regularization or Lasso Regularization: It adds a penalty term proportionate to the absolute value of the weights in the model. This promotes sparse solutions, in which just a subset of the features are included in the final model. When the dataset contains a high number of attributes, but just a few of them are likely to be meaningful, L1 regularization can be helpful.

L2 regularization or Ridge Regularization: It introduces a penalty term proportionate to the model's weights squared. This encourages smaller weights in general, which can help to reduce overfitting. L2 regularization might be useful when the dataset has a high number of potentially essential properties. Dropout Regularization is an approach in which random units in the model are "dropped out" or disregarded for a short period of time during training, reducing the model's capacity and discouraging unit co-adaptation. Dropout regularization can be useful when the dataset is large and intricate, and the model has a high capacity.

To conclude, regularization is a method of reducing overfitting in predictive models by imposing constraints that encourage the model to learn only the most significant components of the input. The regularization method chosen is dependent by the scenario at hand as well as the qualities 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 much a predictive model's anticipated value differs from its true value. They provide a goal against which the model's performance is measured, and the model's parameters are chosen by minimizing a predetermined loss function. As a result, the loss function computes the difference between the algorithm's current output and its expected output. It is a tool that evaluates how effectively your algorithm models data. The loss is further decreased by adjusting the model parameters until the lowest practical loss is reached. Here are some instances of Loss functions in action.

The following loss functions are often employed in regression models:

Mean Squared Error (MSE): The average of the squared discrepancies between the actual and anticipated values is the mean squared error. MAE (Mean Absolute Error): Mean Absolute Error is a simple yet reliable loss function used in regression models. Variables in regression problems may not be perfectly Gaussian due to the presence of outliers (values that are very different from the rest of the data). In such cases, Mean Absolute Error is a good choice because it does not consider the direction of the outliers (unrealistically high positive or negative values). MAE calculates the average sum of the absolute differences between the actual and expected values.

Commonly used classification loss functions:

Binary Cross Entropy: For two-class classification problems, this is the most commonly used loss function. The seemingly incongruous term "entropy" has a statistical significance. Cross-entropy evaluates the difference in randomness between two random variables, whereas entropy measures the unpredictability of the information being processed.

The cross-entropy loss grows when the anticipated probability diverges from the actual label.

Categorical Cross-Entropy Loss: Categorical Cross Entropy Loss is just the product of Binary Cross Entropy Loss and the number of classes. The labels must be encoded just once if the categorical cross-entropy loss function is to be used. Because the vector's other members have been multiplied by zero, only one element will be non-zero. This property has been extended to a SoftMax activation function.

To recap, the loss function is a critical component of a predictive model since it quantifies the difference between predicted and actual outputs and is used to train the model by modifying its parameters. The loss function used is governed on the job at hand and the type of model used.

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.

A prediction model developed with a training set should be evaluated on both the training and validation sets. If both the train error and the validation error (error on data not observed by the model) are small, the model is considered to be a good prediction model. However, without more investigation, it is not recommended to completely trust such a model, as there are several reasons why the model may not be trustworthy.

When a model is underfitted, it suggests that it is unable to accurately capture the relationship between the input and output variables. As a result, such a model performs poorly on both the training and validation sets. As a result, we must increase the model's complexity such that it accurately portrays the relationship between the input and output variables.

When a model is overfitted, it performs exceedingly well on the training set with very high train accuracy, but when applied to the validation set, the error is extremely high with a large variance between the train error and the validation error. This indicates that the model has become very sophisticated, learning the train data with all of the noise and outliers and becoming overly sensitive to even little changes in the input variables. As a result, the generalization aim of any machine learning model is not realized, causing the model to perform very badly on unfamiliar data. Overfitting is particularly common when the train data is limited and the model is sophisticated.

In the offered example, a model with numerous hyperparameters is constructed using a small dataset.

As a result, there is a high probability that the model will be overfitted. As previously stated, a model is only considered a good prediction model if it performs well on both train and unknown data. The proposed model performs well on the training set but is likely to perform poorly on unknown data due to overfitting because it has a high likelihood of overfitting and a significant variance.

As a result, we can't put our faith in the model until it produces correct predictions on train data. It should instead be applied to unseen data to check whether there is any difference between the train error and the validation error. If the variance is too high, the model should be retrained by reducing its complexity or

employing regularization techniques to allow the model to generalize rather than only 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 used in linear models such as Lasso or Ridge Regression for Regularization, which is intended to reduce model complexity.

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

Linear regression models try to reduce the loss function to find the optimal weights for each attribute. Yet, when the model is overfitted and unable to generalize sufficiently, regularization is used to penalize it. This is achieved by introducing a new term in the loss function, the value of which is proportional to the value of Lambda. The total loss value grows as we increase the Lambda value, and the model tries to minimize the coefficients of the parameters to get the optimal response. According to the theory, lowering or regularizing the coefficients can increase prediction accuracy, variance, and model interpretability.

In ridge regression, a penalty is applied by a tuning parameter lambda, which is computed via cross-validation. The purpose is to improve the fit by lowering the residual sum of squares and imposing a shrinkage penalty. The shrinkage penalty is equal to lambda times the sum of the squares of the coefficients, therefore large coefficients are penalized. When lambda grows, the bias remains constant, but the variance drops. The ridge's downside is that it does not enable you to choose variables. It includes all of the variables in the final model.

In lasso, the penalty is the sum of the absolute values of the coefficients. When lambda is large enough, lasso reduces the coefficient estimates to zero and has the effect of setting variables to zero, but ridge does not. As a consequence, lasso performs variable selection similarly to the best subset selection approach. The tuning parameter lambda is chosen via cross-validation. The resultant estimates are practically least squares when lambda is small. When lambda increases, shrinkage occurs, allowing zero-valued variables to be removed. As a result, one of the lasso's primary advantages is that it combines shrinkage with variable selection.

**********PART_B**********

```
# Importing the 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
p <- as.matrix(Carseats_Filtered)
q <- Carseats[,1]

# Let's do a simple linear regression on the data to get the coefficients and R-squared value.

r =lm(q~p)
summary(r)
```

```
##
## Call:
## lm(formula = q ~ p)
##
## 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 ***
## pPrice       -1.35877    0.11369 -11.951  < 2e-16 ***
## pAdvertising  0.83400    0.11734   7.108 5.60e-12 ***
## pPopulation  -0.13723    0.11774  -1.166   0.2445
## pAge         -0.79358    0.11345  -6.995 1.15e-11 ***
## pIncome       0.29267    0.11335   2.582   0.0102 *
## pEducation   -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(r$residuals^2)
```

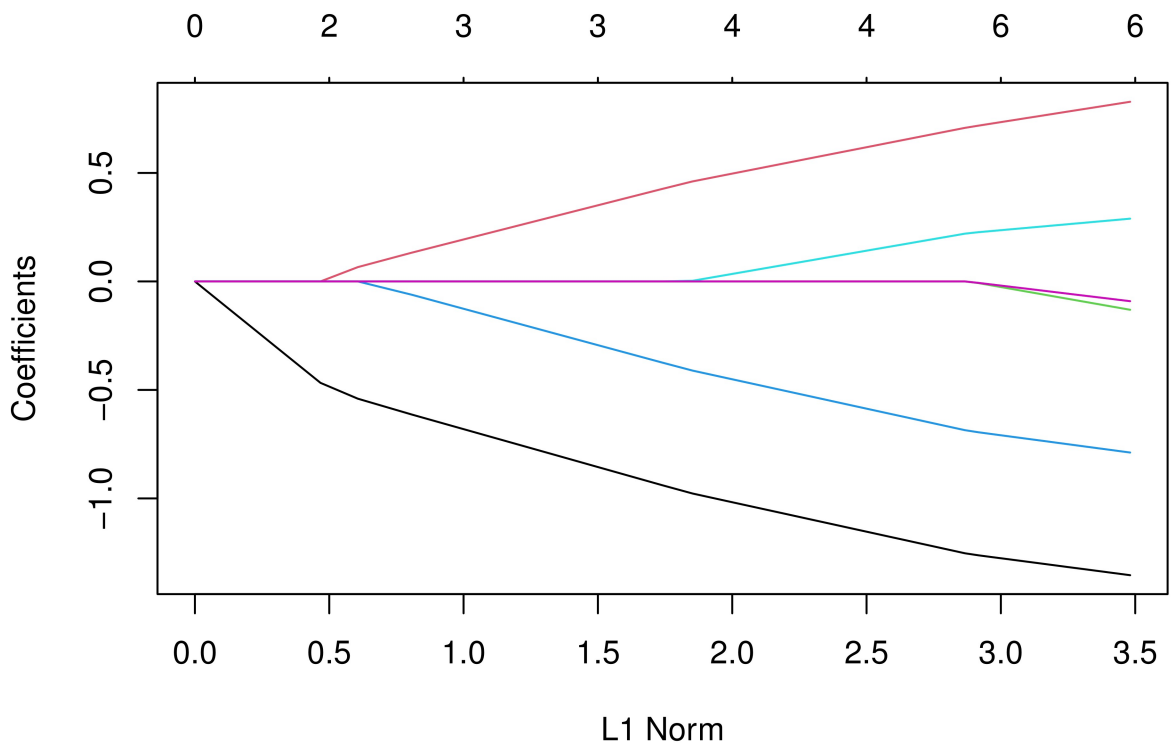
```
## [1] 4.981366
```

As a result of the above data, we can infer that the factors presented account for only 36.43% of the variation in the dependent variable (Sales). Let us now use Lasso Regression on the same data to see if we can improve the R-Squared value.

```
fit <- glmnet(p, q )

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)
```

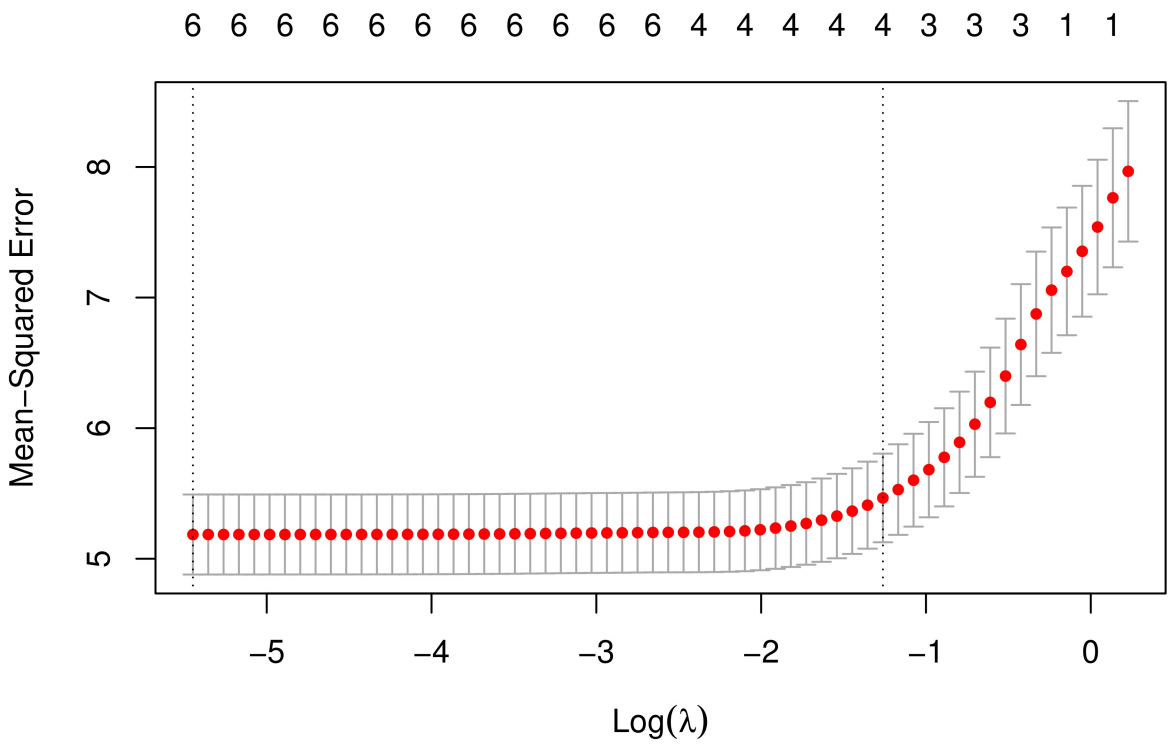
```
##
## Call:  glmnet(x = p, y = q)
##
##    Df  %Dev  Lambda
## 1   0  0.00 1.25500
## 2   1  3.36 1.14400
## 3   1  6.15 1.04200
## 4   1  8.47 0.94940
## 5   1 10.39 0.86500
## 6   1 11.99 0.78820
## 7   2 14.62 0.71820
## 8   3 18.08 0.65440
## 9   3 21.12 0.59620
## 10  3 23.64 0.54330
## 11 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

```r
k_f <- cv.glmnet(p, q, alpha = 1)

# Determining the Smallest lambda value

best_lambda <- k_f$lambda.min

best_lambda
```

```
## [1] 0.004305309
```

```
plot(k_f)
```



As a result of the data given, we can observe that the goal variable, sales with regularization, has just 37.38% volatility 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)?

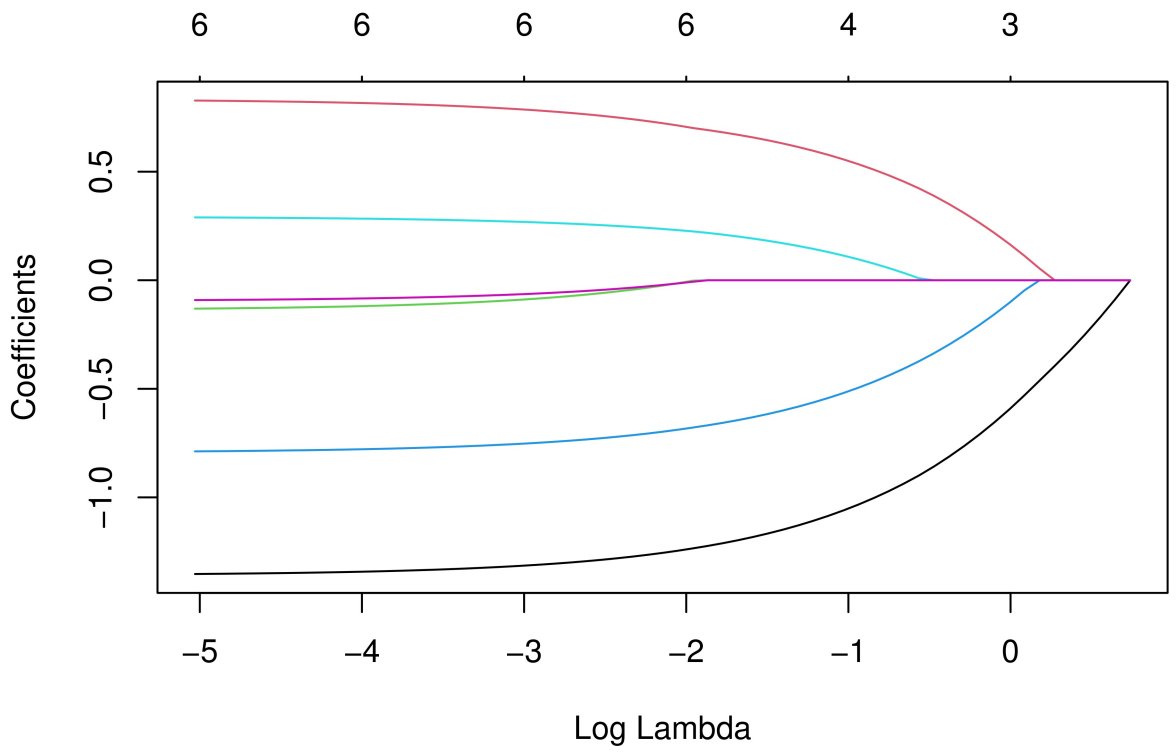
```
best_model <- glmnet(p, q, alpha = 1, lambda = best_lambda)

coef(best_model)
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept)  7.49632500
## Price       -1.35384596
## Advertising  0.82808291
## Population  -0.13062237
## Age         -0.78855156
## Income       0.28931642
## Education   -0.09102494
```

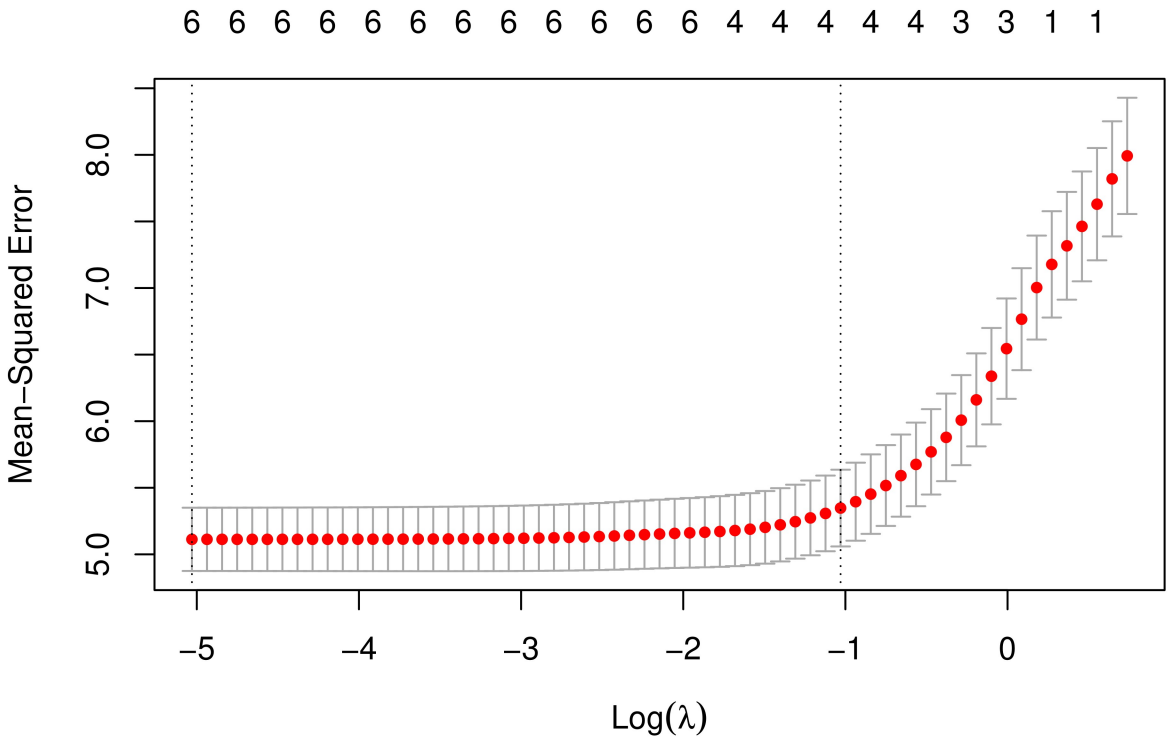
The Price attribute's coefficient with the best lambda value is -1.35384596.

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(p,q, alpha=0.6)
plot(f_e, xvar = "lambda")
```



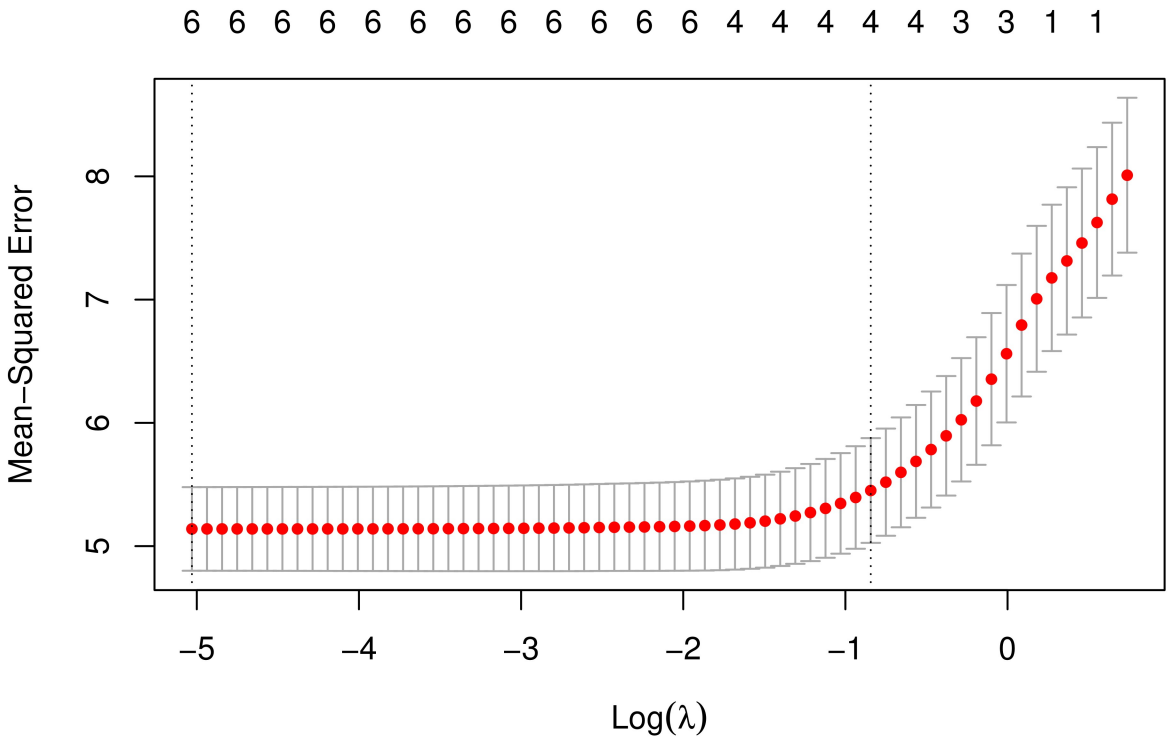

```
plot(cv.glmnet(p,q,alpha=0.6))
```



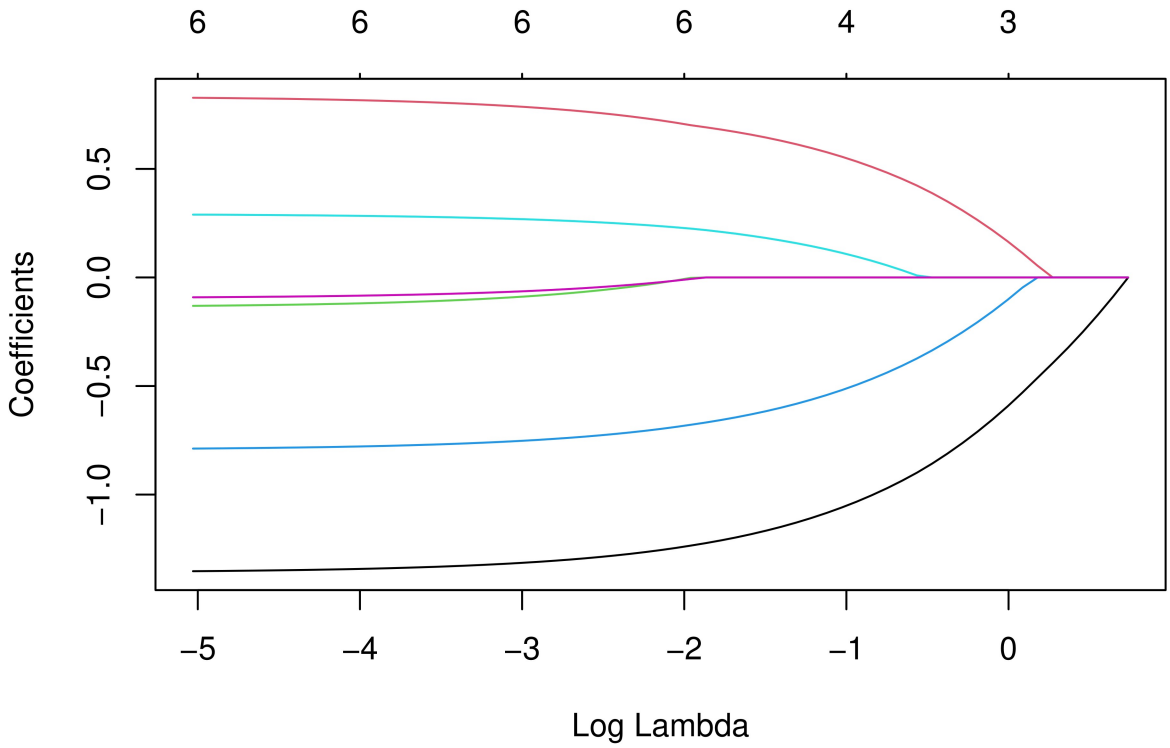

```
# K-fold validation is used to find the best lambda.
ele <- cv.glmnet(p,q, alpha=0.6)
# Finding a lambda value that minimizes the test.
best_lambda <- ele$lambda.min
best_lambda
```

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

```
plot(ele)
```



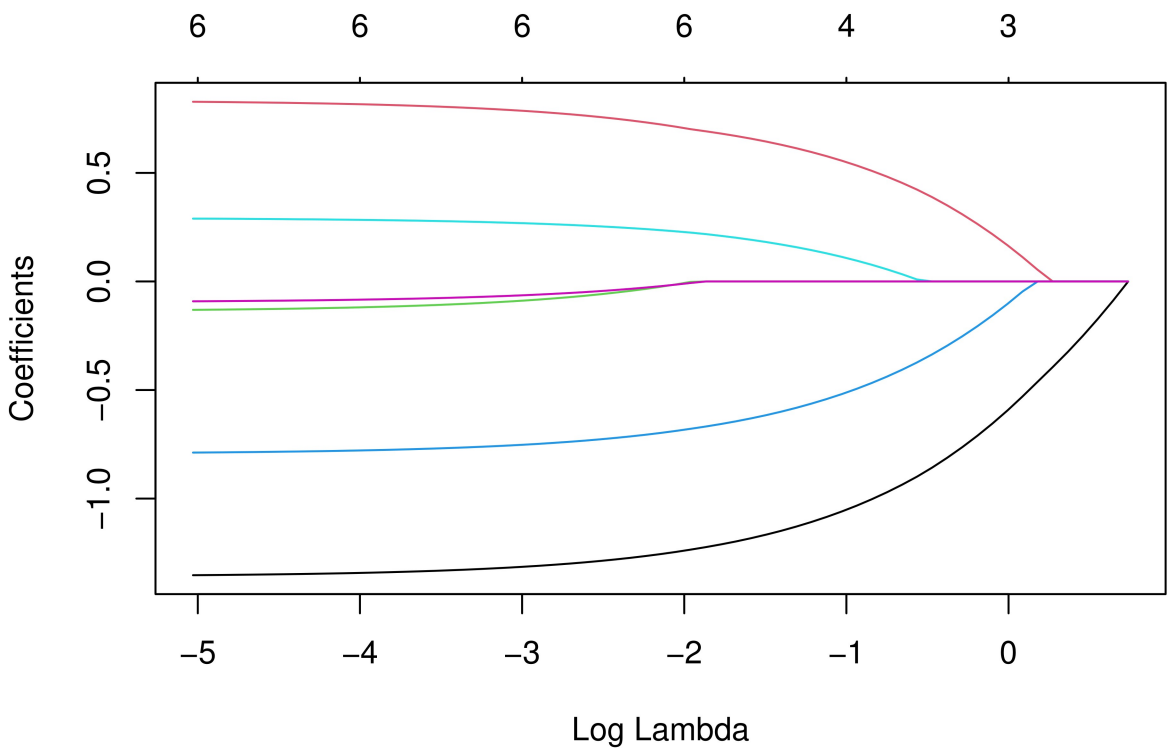

```
fit.e <- glmnet(p,q,alpha=0.6)
plot(fit.e, xvar="lambda")
```



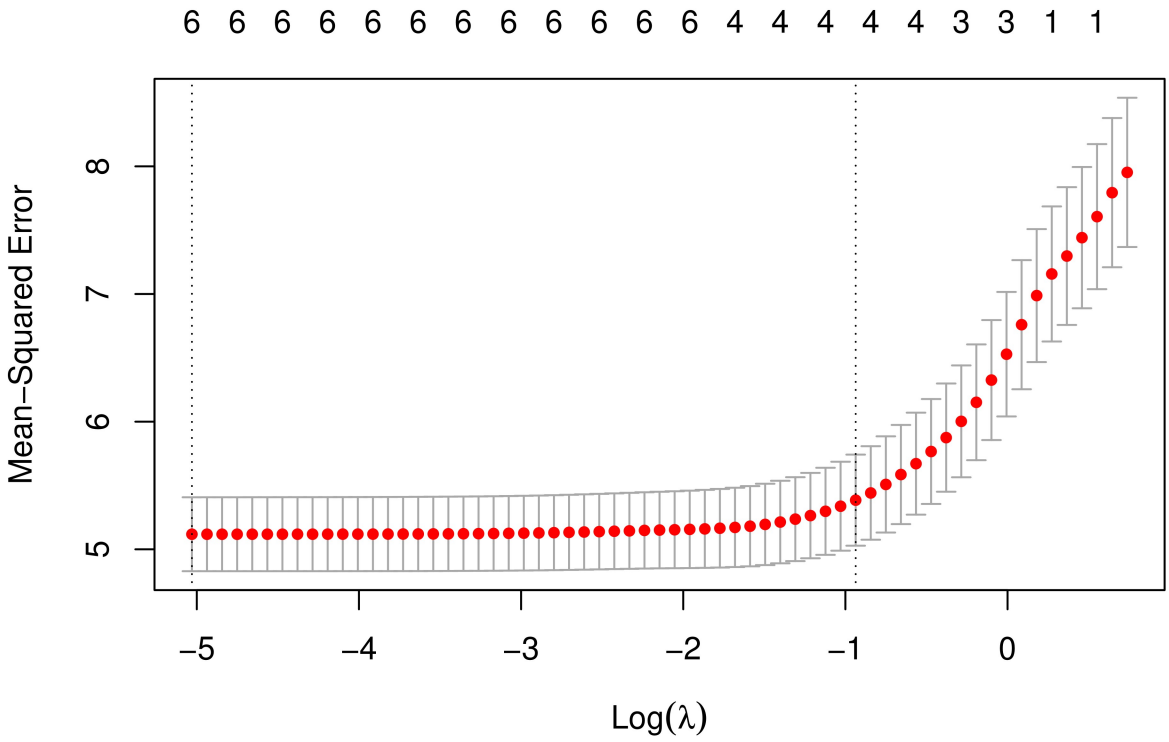
The best lambda value for such elastic-net model is .00653

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

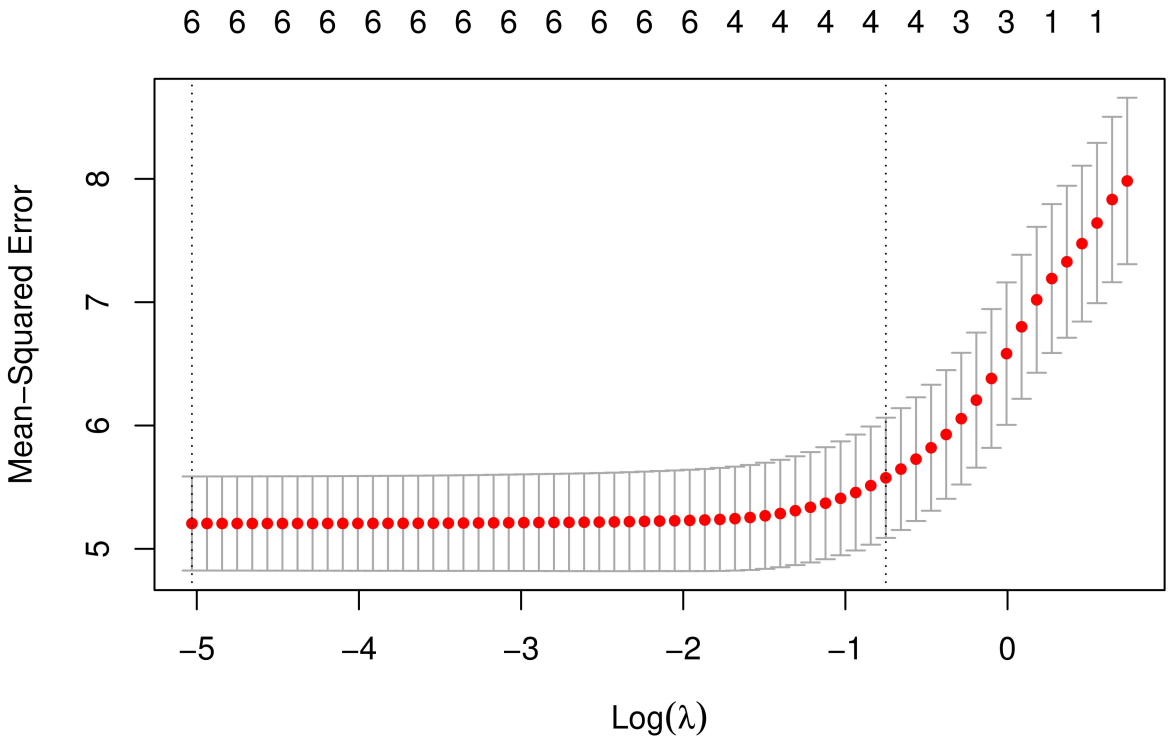
```
fit_ela <- glmnet(p,q,alpha = 0.6)
plot(fit_ela,xvar = "lambda")
```




```
plot(cv.glmnet(p,q,alpha = 0.6))
```




```
cv.fit.ela <- cv.glmnet(p,q, alpha = 0.6)
plot(cv.fit.ela)
```




```
elast<- cv.fit.ela$lambda.min
elast
```

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

The optimal Lambda for an elastic model with alpha set to 0.6 is 0.0023.