MIS 64037 - Assignment 1

Tejasvini Mavuleti

2022-10-29

ADVANCED DATA MINING & PREDICTIVE ANALYTICS INDIVIDUAL ASSIGNMENT 1

Part A

A1) What is the main purpose of regularization when training predictive models?

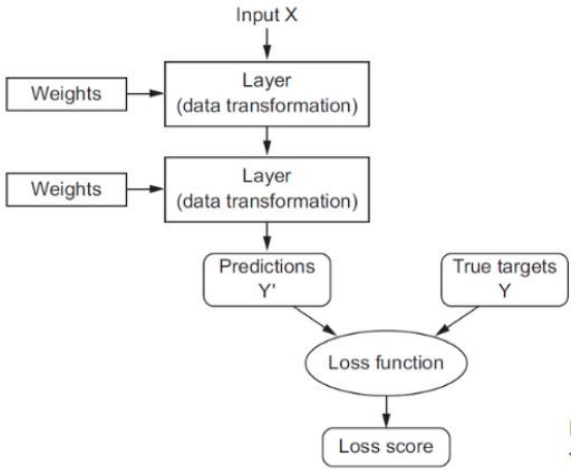
When training machine learning models, one major aspect is to evaluate whether the model is overfitting the data and to learn the patterns from the training data and generalize them to predict the unseen data effectively. Here, generalization is a term used to describe a model's ability to react to new data. There are many causes for the model's lack of generalization, and overfitting of the model is one among them. Overfitting occurs when a model attempts to fit all the data points, capturing noises in the process and leading to inaccurate model development.

This is when regularization plays an important role. Regularization is used to increase the model's generalization capacity by reducing the complexity of the model. It improves a model's performance by simplifying it. We can appropriately fit our machine learning model on a given test set and reduce its errors using Regularization. This means that regularization discourages learning a model of high complexity and flexibility. A highly flexible model possesses the freedom to fit as many data points as possible. And to simplify the model and reduce its complexity, it penalizes the model. Regularization is used to calibrate machine learning models to minimize the adjusted loss function and prevent overfitting or underfitting. There are two types of regularization techniques - Lasso & ridge regression- used in regression models and the dropout used for regularization of deep Learning models.

One example I could think of is if Amazon wishes to build a model to predict if the user would buy a product or not, given his/her usage history for the last 7 days, and use the data for better decision-making for retargeted digital advertisements. The usage history may include the number of pages visited, total time spent, the number of searches done, average time spent, page revisits, and more. The company could build a model it delivers accurate results on existing data, but when they try to predict with unseen data, it doesn't deliver a good result. Here, it can be concluded that the model does more memorization than learning. If the model has a significant difference between evaluation metrics for the training dataset and testing dataset, then it is said to have an overfitting problem.

A2) 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.

A loss function measures your prediction model's ability to predict the expected outcome(or value). We convert the learning problem into an optimization problem, define a loss function and then optimize the algorithm to minimize the loss function. Loss functions measure how far an estimated value given by the predictive model is from its true value. It defines an objective which the model's performance is evaluated against, and the parameters learned by the model are determined by minimizing a chosen loss function. So, the loss function is the function that computes the distance between the current output of the algorithm and the expected output. It's a method to evaluate how your algorithm models the data. The loss is reduced by changing the model's parameters until the lowest possible loss is achieved.



These are some examples of loss functions and how they work.

Loss functions for Regression Models

Mean Squared Error (MSE) Mean Squared Error is the average of the squared differences between the actual and the predicted values. The smaller the mean squared error, the closer you are to finding the line of best fit. For a data point Yi and its predicted value Yi, where n is the total number of data points in the dataset, the mean squared error is calculated using the following formula –

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

Mean Absolute Error (MAE) Mean Absolute Error is one of regression models' most simple yet robust loss functions. It is an ideal option in such cases because it does not consider the direction of the outliers that are unrealistically high positive or negative values. As the name suggests, MAE takes the average sum of the absolute differences between the actual

and the predicted values. For a data point xi and its predicted value yi, n being the total number of data points in the dataset, the mean absolute error is calculated using the following formula –

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

Loss functions for Classification Models

Binary Cross Entropy Binary cross entropy compares each of the predicted probabilities to the actual class output, which can be either 0 or 1. It then calculates the score that penalizes the probabilities based on the distance from the expected value. That means how close or far from the actual value. This is the most common loss function for classification problems with two classes. The word “entropy”, seemingly out-of-place, has a statistical interpretation. Entropy is the measure of randomness in the information being processed, and cross-entropy is the difference in the randomness between two random variables. If the divergence of the predicted probability from the actual label increases, the cross-entropy loss increases. By this, predicting a probability of .011 when the actual observation label is 1 would result in a high loss value. In an ideal situation, a “perfect” model would have a log loss of 0.

Since binary classification means the classes take either 0 or 1, if yi = 0, that term ceases to exist and if yi = 1, the (1-yi) term becomes 0.

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

Categorical Cross Entropy Categorical cross-entropy is a loss function that is used in multi-class classification tasks. These are tasks where an example can only belong to one of many possible categories, and the model must decide which. Formally, it is designed to quantify the difference between two probability distributions. Categorical Cross Entropy loss is essentially Binary Cross Entropy Loss expanded to multiple classes. One requirement when the categorical cross entropy loss function is used is that the labels should be one-hot encoded. This way, only one element will be non-zero, as other elements in the vector would be multiplied by zero. This property is extended to an activation function called softmax.

$$CE = -\sum_{i=1}^{C'=2} t_i log(f(s_i)) = -t_1 log(f(s_1)) - (1 - t_1) log(1 - f(s_1))$$

A3) 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.

There are two reasons why a prediction model is said to be a dumb model, as below. First, when the model is under-fitted, the model cannot accurately capture the relationship between the input and output variables, and such a model's performance is worse both on the training and validation sets. So, we need to increase the model's complexity to accurately capture the relationship between the inputs and the output variables. When a prediction model is built using a training set, we should evaluate it both on the training and validation sets. The model can be said to be a good prediction model only if the training error and validation error, the error on the data unseen by the model, is low.

When the model is overfitted, the model performs very well on the training set with very high train accuracy. In contrast, when the same model is applied to the validation set, the error is very high, and a high variance is found between the train and the validation error. This means that the model has become too complicated. It learns the train data, including all the noise and outliers, and becomes sensitive even to the slightest change in the input variables. So, the objective of any machine learning model, generalization, still needs to be achieved, which makes the model perform very poorly on unseen data. This overfitting happens mainly when the training data is relatively small, and the complexity of the model is very high. For example, the model with many hyperparameters is built using a relatively small dataset in the given example.

Since the given model has a high chance of overfitting and high variance, it performs very well on the training set but is likely to perform poorly on the unseen data due to overfitting. Moreover, there is a high chance for the model to be overfitted. We saw that the model is said to be a good prediction model only when it performs well on both the train and the unseen data.

A4) 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 to reduce the complexity of the model. Linear regression models try to reduce the loss function to get the optimum weights given for each attribute. But, when the model is overfitted and cannot generalize well, we use regularization to penalize the model. This is done by adding an extra term for the loss function directly proportional to the lambda value. So, when we tune this lambda value to increase it, the total loss value is increased, and the model tries to shrink the coefficients of the parameters to arrive at the optimal solution. The idea is that by shrinking or regularizing the coefficients, prediction accuracy can be improved, variance can be decreased, and model interpretability can also be improved.

In ridge regression, we add a penalty using a tuning parameter lambda chosen using cross-validation. The idea is to make the fit small by making the residual sum or squares small, plus adding a shrinkage penalty. The shrinkage penalty is lambda times the sum of squares of the coefficients, so coefficients that get too large are penalized. As lambda gets larger, the bias is unchanged, but the variance drops. The drawback of the ridge is that it doesn't select variables. It includes all the variables in the final model.

In lasso, the penalty is the sum of the absolute values of the coefficients. Lasso shrinks the coefficient estimates towards zero and sets variables equal to zero when lambda is large enough while ridge does not. Hence, much like the best subset selection method, lasso performs variable selection. The tuning parameter lambda is chosen by cross-validation. When lambda is small, the result is essentially the least squares estimates. As lambda increases, shrinkage occurs so that variables that are at zero can be thrown away. So, a major advantage of the lasso is that it combines shrinkage and selection of variables.

In simple terms -

In ridge regression: -lambda = 0, coefficients of the model are the exact same as normal -lambda = infinity, all coefficients of the model approach 0 -no feature selection occurs; all variables will stay in the model -helps control multicollinearity and reduce variance

In lasso: -lambda = 0, coefficients of the model are the exact same as normal -lambda = infinity, all coefficients of the model are 0 -feature selection occurs, variables can be completely gotten rid of -helps control multicollinearity and reduce variance ——

Part B

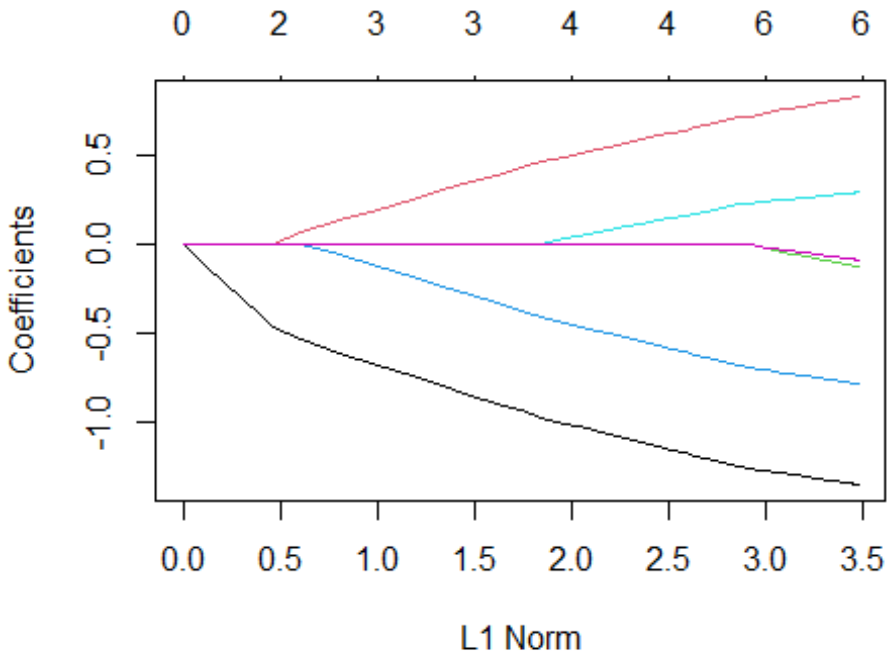
```
# Calling the libraries
library(ISLR)

## Warning: package 'ISLR' was built under R version 4.2.1

library(caret)

## Warning: package 'caret' was built under R version 4.2.1

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 4.2.1

## Loading required package: lattice

## Warning: package 'lattice' was built under R version 4.2.1

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 4.2.1

## Loading required package: Matrix

## Loaded glmnet 4.1-4

# Using the car seats dataset

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
##
##
##
##
```

B1) Building a Lasso regression model to predict Sales based on all other attributes.

```
# Adding all the input attributes to the CarSeats_Filtered - scale the input
attributes as well

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 build a matrix
x <- Carseats_Filtered

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

# Now we build 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)
```



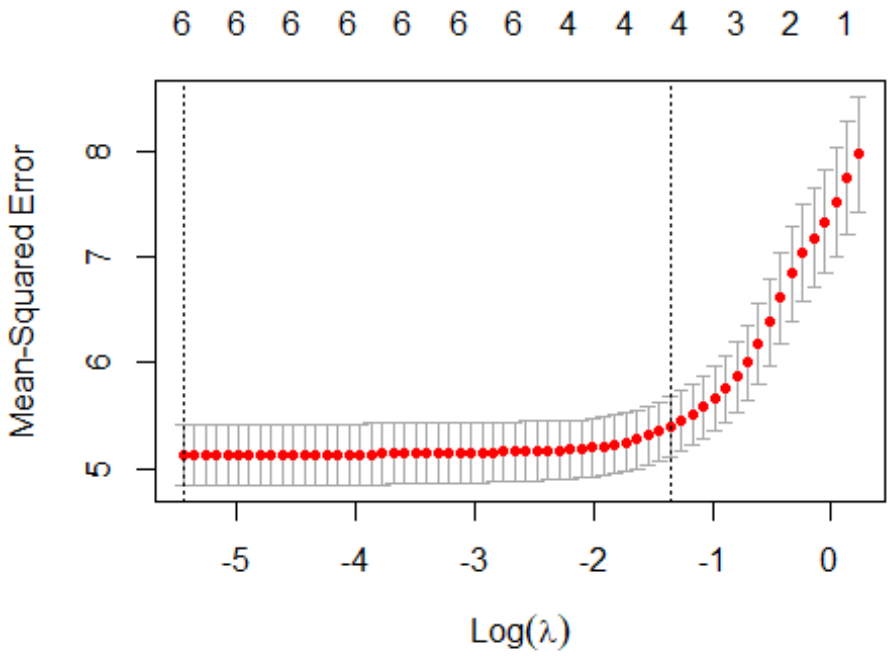

```
print(fit)

##
## 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
```

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

# Finding the minimum lambda value
best_lambda <- cv_fit$lambda.min
best_lambda
```

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

```
plot(cv_fit)
```



There is only 37.38% variance in the target variable and sales with regularization. Also, the best lambda value is 0.0043 in this model.

B2) What is the coefficient for the price (normalized) attribute in the best model (i.e. model with the optimal lambda)?

```
# The coefficient for the price attribute in the best model is -
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       -1.35384596
## Advertising  0.82808291
## Population  -0.13062237
## Age         -0.78855156
## Income       0.28931642
## Education   -0.09102494
```

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

B3) How many attributes remain in the model if lambda is set to 0.01?

```
# a - The coefficients of the attributes that are still left if lambda 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
```

These are the coefficients of the independent attributes with the lambda value 0.01. There are no coefficients are eliminated here. -

```
# b - How is that number changing if lambda is increased 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    .
```

Now these values of the independent attributes have shrink-ed to some extent and there are two coefficients of the attributes that are removed when the lambda is set to 0.1.

```
# c- Do you expect more variables to stay in the model (i.e., to have non-zer
o coefficients) as we increase lambda?
# Let's say that  the coefficients of the attributes that are still remained
if lambda 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    .
```

Two of the coefficients of the attributes are gone and the independent attributes have shrink-ed more when lambda value is 0.3.

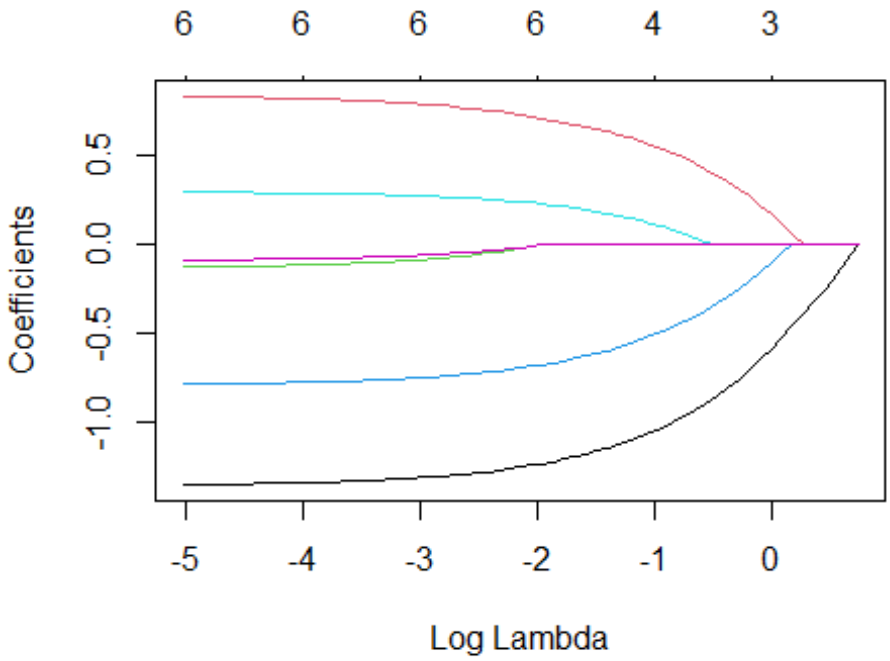
```
# Now for example what will happen if the coefficients of the attributes that are still kept if lambda is set to 0.5?
best_model <- glmnet(x, y, alpha = 1, lambda = 0.5)
coef(best_model)
```

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

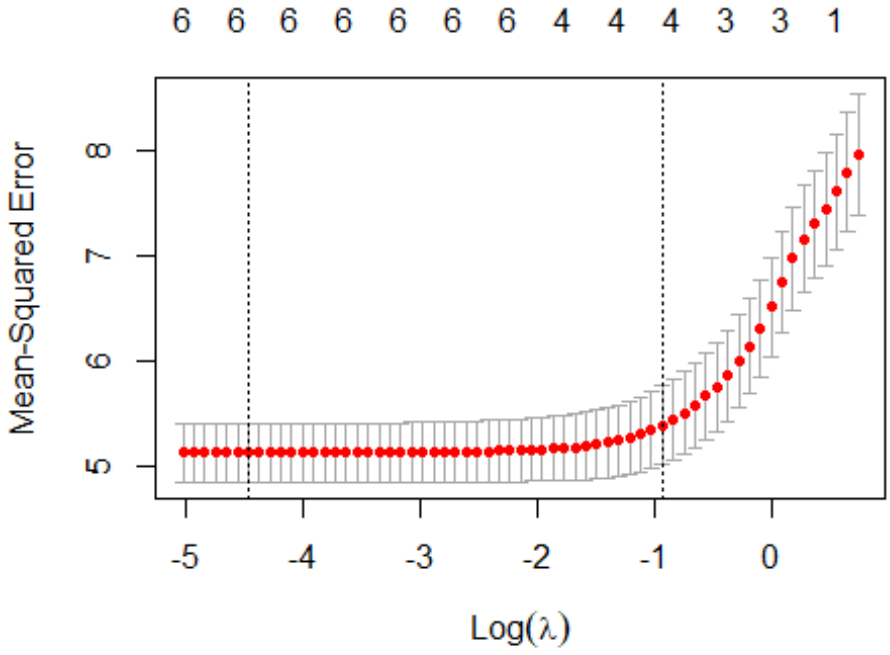
At this point, three of the coefficients of the attributes are eliminated and the independent attributes have shrink-ed further when lambda value is 0.5.

B4) Build an elastic-net model with alpha set to 0.6. What is the best value of lambda for such a model?

```
# Now we build 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
```

Out of all these, the variance is 37.38 in the dependent variable - that is the Sales which shows that the given attributes apply the regularization when we set the alpha value to 0.6 and then the best lambda value is 0.00654.