Individual Assignment - 1

Rohith Chandra

2022-10-31

Part A

A1. Main purpose of Regularization

Regularization refers to techniques that are used to calibrate machine learning models in order to minimize the adjusted loss function and prevent over fitting or under fitting. Using Regularization, we can fit our machine learning model appropriately on a given test set and hence reduce the errors in it. Regularization is a technique in machine learning that tries to achieve the generalization of the model. It means that our model works well not only with training or test data, but also with the data it'll receive in the future. Regularized linear regression is used to improve stability, reduce the impact of collinearity, and improve computational efficiency and generalization.Also, it controls the coefficient values either by decreasing the values or compltely dropping the variables while minimizing the loss. When the variables are dropped the model's complexity is reduced and over fitting can be reduced.

A2. Loss function in a predictive model

The loss function calculates the difference between the model's output with that of expected output of the model or a variable. If the difference is larger, then the loss function penalizes the model in order to make the difference smaller as the objective is to make the difference smaller.

Loss function regression models

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.

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.

Loss functions classification models

Binary cross entropy - BCE 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. 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

Categorical Cross Entropy - CCE 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. 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.

A3. Classification models with many parameters

No, we cannot trust the model. When the data set is too small , there is a possibility that model may follow the data too closely , learning too little patterns. And it may learn the noise , which is not required , as well. Noise is stochastic and that is difficult to predict. Hence it performs very well on the training data set and may perform very badly on the test data as it hasn't seen the new data and the required patterns are not learnt well.

A4. What is the role of the lambda parameter in regularized linear models such as Lasso or Ridge regression models?

We need to make a note that while increasing the lamda value, one needs to make sure that the model which is otherwise optimal or over fit would not under fit the model as lambda is goes too small. Because, regularization penalizes the variable coefficients in the model to avoid over fitting we need to use the penalty parameter or lambda. In Lasso, when we increase the lambda value it drops variables that are not significant to the model and also reduces the value of the coefficients of the remaining variables in the model, while minimizing the overall loss function. This way the model will get rid of complexity by reducing the number of features in the data set, eliminating the over fitting scenario. On the contrary, ridge model only decreases the coefficient value while keeping all the variables in the model.

Part B

```
# Libraries
library(ISLR)
library(caret)

## Loading required package: ggplot2
```

```
## Loading required package: lattice

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

# Attaching the carsets data set to solve the regression problem
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
##
##
##
##

Carseats_Filtered <- Carseats %>% select("Sales", "Price",
"Advertising","Population","Age","Income","Education")
```

```r
x <- Carseats_Filtered
y <- Carseats %>% select("Sales") %>% as.matrix()

# Converting the data into a matrix
Carseats_Sales <- Carseats_Filtered$Sales
Carseats_Other <- data.matrix(Carseats_Filtered[, c(-1)])
# Preprocess and summarize the data
preProc <- preProcess(Carseats_Other, method=c("center", "scale"))
Carseats_Scaled <- predict(preProc,Carseats_Other)
summary(Carseats_Scaled)
```

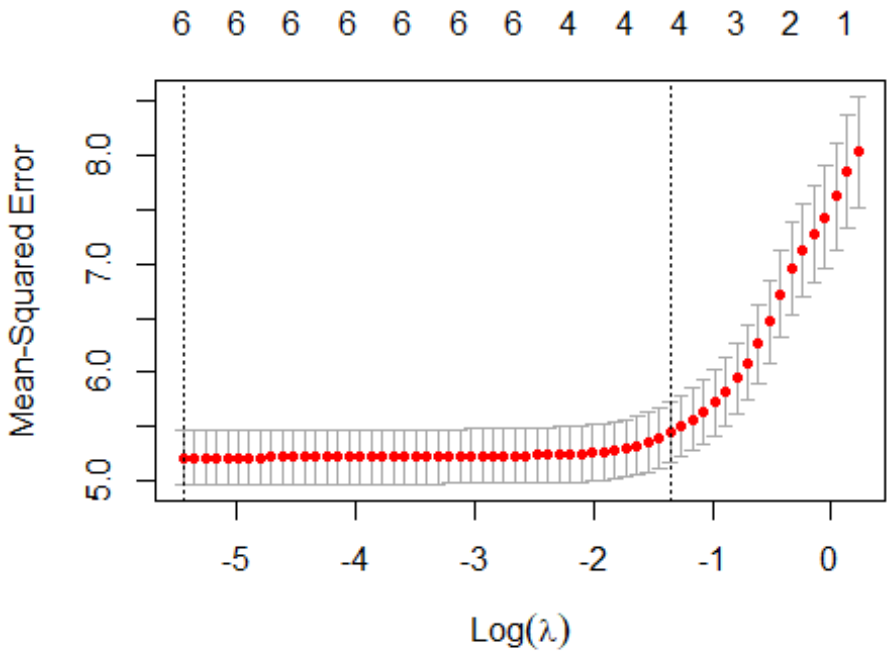
```
##      Price           Advertising        Population            Age          
##  Min.   :-3.87702   Min.   :-0.9977   Min.   :-1.72918   Min.   :-1.74827  
##  1st Qu.:-0.66711   1st Qu.:-0.9977   1st Qu.:-0.85387   1st Qu.:-0.83779  
##  Median : 0.05089   Median :-0.2459   Median : 0.04858   Median : 0.07268  
##  Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.00000  
##  3rd Qu.: 0.64219   3rd Qu.: 0.8067   3rd Qu.: 0.90693   3rd Qu.: 0.78255  
##  Max.   : 3.17633   Max.   : 3.3630   Max.   : 1.65671   Max.   : 1.64673  
##      Income           Education       
##  Min.   :-1.70290   Min.   :-1.48825  
##  1st Qu.:-0.92573   1st Qu.:-0.72504  
##  Median : 0.01224   Median : 0.03816  
##  Mean   : 0.00000   Mean   : 0.00000  
##  3rd Qu.: 0.79834   3rd Qu.: 0.80137  
##  Max.   : 1.83458   Max.   : 1.56457
```

B1. Lasso regression to predict sales

```r
Lasso_model <- cv.glmnet(Carseats_Scaled,Carseats_Sales , alpha = 1)
Lambda_Best <- Lasso_model$lambda.min
Lambda_Best
```

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

```r
#Testing MSE by lambda value
plot(Lasso_model)
```



The result shows that the optimal lambda value is 0.004305309.

B2. The coefficient for the price

```
Lambda_Best<- glmnet(x,y, alpha = 1, lambda = Lambda_Best)
coef(Lambda_Best)

## 8 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept) 0.01144231
## Sales       0.99847361
## Price       .
## Advertising .
## Population  .
## Age         .
## Income      .
## Education   .
```

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

B3. Finding the attruibutes if lambda is set to 0.01 and looking at the changes in the value

```
Lambda_Best1 <- glmnet(x, y, alpha = 1, lambda = 0.01)
coef(Lambda_Best1)
```

```
## 8 x 1 sparse Matrix of class "dgCMatrix"
##                      s0
## (Intercept) 0.02657722
## Sales       0.99645463
## Price       .
## Advertising .
## Population  .
## Age         .
## Income      .
## Education   .
```

```
Lambda_Best2 <- glmnet(x, y, alpha = 1, lambda = 0.1)
coef(Lambda_Best2)
```

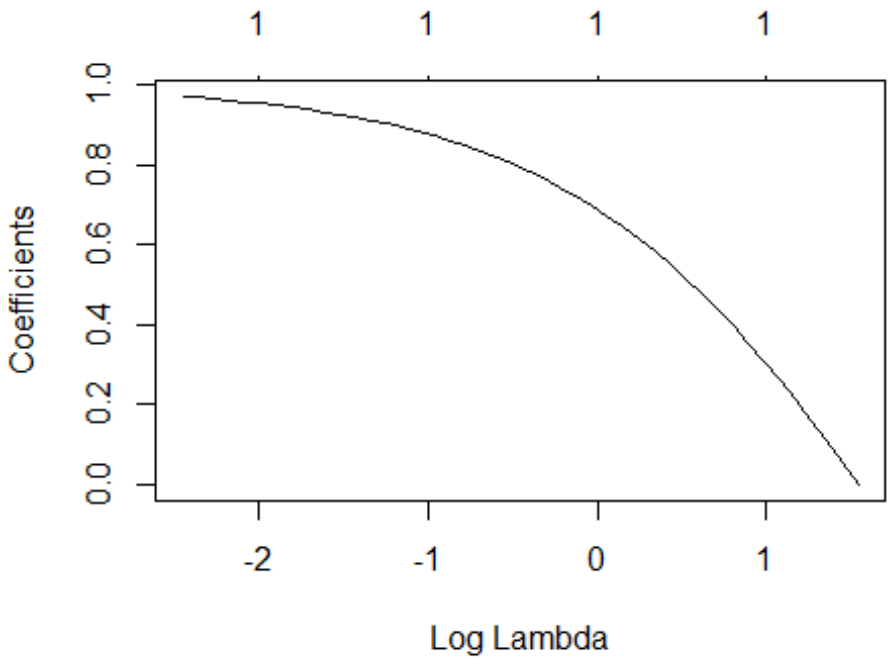
```
## 8 x 1 sparse Matrix of class "dgCMatrix"
##                     s0
## (Intercept) 0.2657722
## Sales       0.9645463
## Price       .
## Advertising .
## Population  .
## Age         .
## Income      .
## Education   .
```

It is clear that with Lambda = 0.01 the variables are remaining but when we change it to 0.1 population and education are removed. So, when the Lambda increases the variables will drop.

B4. Build an elastic-net model with alpha set to 0.6

```
el_net = glmnet(x, y, alpha = 0.6)
plot(el_net, xvar = "lambda")
```

```
## Warning in plotCoef(x$beta, lambda = x$lambda, df = x$df, dev =
x$dev.ratio, : 1
## or less nonzero coefficients; glmnet plot is not meaningful
```




```
summary(el_net)

##           Length Class     Mode
## a0         44    -none-    numeric
## beta      308    dgCMatrix S4
## df         44    -none-    numeric
## dim         2    -none-    numeric
## lambda     44    -none-    numeric
## dev.ratio  44    -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 4.7010
## 2   1 10.75 4.2830
## 3   1 20.66 3.9030
## 4   1 29.76 3.5560
## 5   1 38.05 3.2400
```

```
## 6   1 45.56 2.9520
## 7   1 52.34 2.6900
## 8   1 58.42 2.4510
## 9   1 63.84 2.2330
## 10  1 68.65 2.0350
## 11  1 72.91 1.8540
## 12  1 76.65 1.6890
## 13  1 79.93 1.5390
## 14  1 82.80 1.4030
## 15  1 85.29 1.2780
## 16  1 87.44 1.1640
## 17  1 89.31 1.0610
## 18  1 90.91 0.9668
## 19  1 92.29 0.8809
## 20  1 93.47 0.8026
## 21  1 94.48 0.7313
## 22  1 95.34 0.6664
## 23  1 96.07 0.6072
## 24  1 96.69 0.5532
## 25  1 97.22 0.5041
## 26  1 97.66 0.4593
## 27  1 98.04 0.4185
## 28  1 98.36 0.3813
## 29  1 98.62 0.3474
## 30  1 98.85 0.3166
## 31  1 99.03 0.2884
## 32  1 99.19 0.2628
## 33  1 99.33 0.2395
## 34  1 99.44 0.2182
## 35  1 99.53 0.1988
## 36  1 99.61 0.1812
## 37  1 99.67 0.1651
## 38  1 99.73 0.1504
## 39  1 99.77 0.1370
## 40  1 99.81 0.1249
## 41  1 99.84 0.1138
## 42  1 99.87 0.1037
## 43  1 99.89 0.0945
## 44  1 99.91 0.0861
```

Out of all these, the variance is 37.38 in the sales and when we set the alpha value to 0.6 and then the best lambda value is 0.00654.