Assignment1_ADM

Mukhtar

3/14/2022

R Markdown

Problem Statement

Data Preparation

```
getwd()

## [1] "C:/Users/Mukht/OneDrive/Desktop/Kent State University/College of Busi
ness Admin-Bus. Analytics Program/Msc. KSU-2nd Semester-2022/Assignments/MIS
64037 Adv. data Mining_Predictive Analytics/Assignment1"

setwd("C:\\Users\\Mukht\\OneDrive\\Desktop\\Kent State University\\College of
Business Admin-Bus. Analytics Program\\Msc. KSU-2nd Semester-2022\\Assignment
s\\MIS 64037 Adv. data Mining_Predictive Analytics\\Assignment1")

ADM_Assignment1<-read.csv("carseats_ADM.csv")
str(ADM_Assignment1)

## 'data.frame':    400 obs. of  11 variables:
##  $ ï..Sales   : num  9.5 11.22 10.06 7.4 4.15 ...
##  $ Income     : int  73 48 35 100 64 113 105 81 110 113 ...
##  $ Advertising: int  11 16 10 4 3 13 0 15 0 0 ...
##  $ Population : int  276 260 269 466 340 501 45 425 108 131 ...
##  $ Price      : int  120 83 80 97 128 72 108 120 124 124 ...
##  $ Age        : int  42 65 59 55 38 78 71 67 76 76 ...
##  $ Education  : int  17 10 12 14 13 16 15 10 10 17 ...
##  $ Urban      : chr  "Yes" "Yes" "Yes" "Yes" ...
##  $ US         : chr  "Yes" "Yes" "Yes" "Yes" ...
##  $ CompPrice  : int  138 111 113 117 141 124 115 136 132 132 ...
##  $ ShelveLoc  : chr  "Bad" "Good" "Medium" "Medium" ...

head(ADM_Assignment1)

##   ï..Sales Income Advertising Population Price Age Education Urban  US
## 1     9.50     73          11        276   120  42        17   Yes Yes
## 2    11.22     48          16        260    83  65        10   Yes Yes
## 3    10.06     35          10        269    80  59        12   Yes Yes
## 4     7.40    100           4        466    97  55        14   Yes Yes
## 5     4.15     64           3        340   128  38        13   Yes  No
## 6    10.81    113          13        501    72  78        16    No Yes
##   CompPrice ShelveLoc
```

```
## 1        138        Bad
## 2        111       Good
## 3        113     Medium
## 4        117     Medium
## 5        141        Bad
## 6        124        Bad
```

#Three of the variables are factors, while the rest are numeric. Currently there are no missing observations.

```
Carseats_Filtered  <- ADM_Assignment1[1:7]
Carseats_Filtered
```

```
##      ï..Sales Income Advertising Population Price Age Education
## 1        9.50     73          11        276   120  42        17
## 2       11.22     48          16        260    83  65        10
## 3       10.06     35          10        269    80  59        12
## 4        7.40    100           4        466    97  55        14
## 5        4.15     64           3        340   128  38        13
## 6       10.81    113          13        501    72  78        16
## 7        6.63    105           0         45   108  71        15
## 8       11.85     81          15        425   120  67        10
## 9        6.54    110           0        108   124  76        10
## 10       4.69    113           0        131   124  76        17
## 11       9.01     78           9        150   100  26        10
## 12      11.96     94           4        503    94  50        13
## 13       3.98     35           2        393   136  62        18
## 14      10.96     28          11         29    86  53        18
## 15      11.17    117          11        148   118  52        18
## 16       8.71     95           5        400   144  76        18
## 17       7.58     32           0        284   110  63        13
## 18      12.29     74          13        251   131  52        10
## 19      13.91    110           0        408    68  46        17
## 20       8.73     76          16         58   121  69        12
## 21       6.41     90           2        367   131  35        18
## 22      12.13     29          12        239   109  62        18
## 23       5.08     46           6        497   138  42        13
## 24       5.87     31           0        292   109  79        10
## 25      10.14    119          16        294   113  42        12
## 26      14.90     32           0        176    82  54        11
## 27       8.33    115          11        496   131  50        11
## 28       5.27    118           0         19   107  64        17
## 29       2.99     74           0        359    97  55        11
## 30       7.81     99          15        226   102  58        17
## 31      13.55     94           0        447    89  30        12
## 32       8.25     58          16        241   131  44        18
## 33       6.20     32          12        236   137  64        10
## 34       8.77     38          13        317   128  50        16
## 35       2.67     54           0        406   128  42        17
## 36      11
```

```
library(glmnet)

## Warning: package 'glmnet' was built under R version 4.1.2

## Loading required package: Matrix

## Loaded glmnet 4.1-3

library(class)
library(caret)

library(modeest)

library(glmnetUtils)

## Warning: package 'glmnetUtils' was built under R version 4.1.2

##
## Attaching package: 'glmnetUtils'

## The following objects are masked from 'package:glmnet':
##
##     cv.glmnet, glmnet
```

#Check to explore Misssing Data

```
#We Look at the summary of the dataset and see if there are NA's present in v
ariables/columns
NA_perct <- function(df, fmt = F) {
  return (df %>%
            is.na() %>%
            colMeans() %>%
            sapply(function(x) {
              if (fmt) {
                return(sprintf("%.5f%", x))
              }
              return (x)
            })
          )
}
NA_perct_df <- NA_perct(Carseats_Filtered) %>%
  data_frame(Columns = names(.), `NA %` = .) %>%
  mutate_at(
    vars(`NA %`),
    funs(round(. * 100, 2))
  ) %>%
  mutate(label = sprintf("%g%%", `NA %`)) %>%
  arrange(desc(`NA %`))

## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## Please use `tibble()` instead.
```

```
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
## Warning: `funs()` was deprecated in dplyr 0.8.0.
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
NA_perct_df %>% select(-label)
```

```
## # A tibble: 7 x 2
##   Columns     `NA %`
##   <chr>        <dbl>
## 1 ï..Sales         0
## 2 Income           0
## 3 Advertising      0
## 4 Population       0
## 5 Price            0
## 6 Age              0
## 7 Education        0
```

```
summary(Carseats_Filtered)
```

```
##     ï..Sales          Income        Advertising       Population
##  Min.   : 0.000   Min.   : 21.00   Min.   : 0.000   Min.   : 10.0
##  1st Qu.: 5.390   1st Qu.: 42.75   1st Qu.: 0.000   1st Qu.:139.0
##  Median : 7.490   Median : 69.00   Median : 5.000   Median :272.0
##  Mean   : 7.496   Mean   : 68.66   Mean   : 6.635   Mean   :264.8
##  3rd Qu.: 9.320   3rd Qu.: 91.00   3rd Qu.:12.000   3rd Qu.:398.5
##  Max.   :16.270   Max.   :120.00   Max.   :29.000   Max.   :509.0
##      Price            Age          Education
##  Min.   : 24.0   Min.   :25.00   Min.   :10.0
##  1st Qu.:100.0   1st Qu.:39.75   1st Qu.:12.0
##  Median :117.0   Median :54.50   Median :14.0
##  Mean   :115.8   Mean   :53.32   Mean   :13.9
##  3rd Qu.:131.0   3rd Qu.:66.00   3rd Qu.:16.0
##  Max.   :191.0   Max.   :80.00   Max.   :18.0
```

```
glimpse(Carseats_Filtered)
```

```
## Rows: 400
## Columns: 7
```

```
## $ ï..Sales    <dbl> 9.50, 11.22, 10.06, 7.40, 4.15, 10.81, 6.63, 11.85, 6.
54, ~
## $ Income      <int> 73, 48, 35, 100, 64, 113, 105, 81, 110, 113, 78, 94, 3
5, 2~
## $ Advertising <int> 11, 16, 10, 4, 3, 13, 0, 15, 0, 0, 9, 4, 2, 11, 11, 5,
0, ~
## $ Population  <int> 276, 260, 269, 466, 340, 501, 45, 425, 108, 131, 150,
503,~
## $ Price       <int> 120, 83, 80, 97, 128, 72, 108, 120, 124, 124, 100, 94,
136~
## $ Age         <int> 42, 65, 59, 55, 38, 78, 71, 67, 76, 76, 26, 50, 62, 53
, 52~
## $ Education   <int> 17, 10, 12, 14, 13, 16, 15, 10, 10, 17, 10, 13, 18, 18
, 18~
```

Make all predictors numeric

```
#Let's try to create a dummy variable to check if the given factor is appropr
iate
# Fit a model of Sales with all predictors
carseat_model <- lm(ï..Sales ~ ., data = Carseats_Filtered)
# Extract the top 6 rows of the model matrix
model.matrix(carseat_model) %>%
  head

##   (Intercept) Income Advertising Population Price Age Education
## 1           1     73          11        276   120  42        17
## 2           1     48          16        260    83  65        10
## 3           1     35          10        269    80  59        12
## 4           1    100           4        466    97  55        14
## 5           1     64           3        340   128  38        13
## 6           1    113          13        501    72  78        16
```

#We now confirm that all of these columns are numeric

```
#Create dummy variables
dummies <- dummyVars(ï..Sales ~ ., data = Carseats_Filtered, fullRank = T)
numeric_frame <- predict(dummies, newdata = Carseats_Filtered)
# or, putting the two steps together into one code chunk
numeric_frame <- dummyVars(ï..Sales ~ ., data = Carseats_Filtered, fullRank =
T) %>%
  predict(newdata = Carseats_Filtered)
head(numeric_frame)

##   Income Advertising Population Price Age Education
## 1     73          11        276   120  42        17
## 2     48          16        260    83  65        10
## 3     35          10        269    80  59        12
## 4    100           4        466    97  55        14
## 5     64           3        340   128  38        13
## 6    113          13        501    72  78        16
```

#We may notice that: #1. dummyVars()has produced exactly the same predictor matrix aslm(), minus the intercept column. #2. The first argument todummyVars()is a model formula:y ~ x. #3. We usefullRank = Tas an argument todummVars()to return the appropriate number of dummy variables #4. To obtain the numeric predictor matrix fromdummyVars()requires using the resulting output as an input to thepredict()function, with the original dataset specified in thenewdata` argument.

```
# Remove near zero variance predictors
# Find nzv predictors
nzv(numeric_frame)

## integer(0)
```

#This result tells us that all the columns in this dataset have enough variance to function as useful predictors

Imputation with medians

###Now that we have an entirely numeric predictor frame, we can use the `train()` function to simultaneously fit the model and impute missings with variable medians. `train()` has, among others, the following arguments:

`x`: the numeric predictor matrix.

`y`: the outcome variable. `caret` will also fit a model using model formula syntax (`y ~ x`)

```
(caret_lm <- train(x = numeric_frame,
                   y = Carseats_Filtered$ï..Sales,
                   method = "lm",
                   preProcess = c("medianImpute")))

## Linear Regression
##
## 400 samples
##   6 predictor
##
## Pre-processing: median imputation (6)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 400, 400, 400, 400, 400, 400, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   2.274046  0.3430736  1.818713
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```r
# Get model summary
summary(caret_lm)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## 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) 15.9570842  1.0275674  15.529  < 2e-16 ***
## Income       0.0104576  0.0040504   2.582   0.0102 *
## Advertising  0.1254063  0.0176440   7.108 5.60e-12 ***
## Population  -0.0009312  0.0007989  -1.166   0.2445
## Price       -0.0573886  0.0048020 -11.951  < 2e-16 ***
## Age         -0.0489852  0.0070027  -6.995 1.15e-11 ***
## Education   -0.0364657  0.0433361  -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
```

#The degrees of freedom reported here should be the number of observations minus the number of coefficients minus 1. And it is: 400 - 6 - 1 = 393. This tells us that `caret` has successfully imputed the missing observations, otherwise the rows with NAs would have been removed, making degrees of freedom much smaller.

Imputation with missForest

There are other approaches to imputation in R that treat missing data as a prediction problem. The `misForest()` function in the `missForest` package uses the random forest algorithm to predict missing observations in a given column using the non-missing data in the other columns.

```r
library(missForest)
# We remove the target and set the seed
set.seed(123)
imputed <- missForest(select(Carseats_Filtered, -ï..Sales))$ximp
```

```
##   missForest iteration 1 in progress...done!
##   missForest iteration 2 in progress...done!
```

```
summary(imputed)

##      Income        Advertising      Population        Price      
##  Min.   : 21.00   Min.   : 0.000   Min.   : 10.0   Min.   : 24.0  
##  1st Qu.: 42.75   1st Qu.: 0.000   1st Qu.:139.0   1st Qu.:100.0  
##  Median : 69.00   Median : 5.000   Median :272.0   Median :117.0  
##  Mean   : 68.66   Mean   : 6.635   Mean   :264.8   Mean   :115.8  
##  3rd Qu.: 91.00   3rd Qu.:12.000   3rd Qu.:398.5   3rd Qu.:131.0  
##  Max.   :120.00   Max.   :29.000   Max.   :509.0   Max.   :191.0  
##       Age          Education   
##  Min.   :25.00   Min.   :10.0  
##  1st Qu.:39.75   1st Qu.:12.0  
##  Median :54.50   Median :14.0  
##  Mean   :53.32   Mean   :13.9  
##  3rd Qu.:66.00   3rd Qu.:16.0  
##  Max.   :80.00   Max.   :18.0
```

Overfitting and Regularization

```
# Fit and assess a complicated linear model
caret_overfit <- train(ï..Sales ~ .^2,
                  method = "lm",
                  data = cbind(ï..Sales = Carseats_Filtered$ï..Sales, imputed
))
summary(caret_overfit)

## 
## Call:
## lm(formula = .outcome ~ ., data = dat)
## 
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -5.9614 -1.5334 -0.0956  1.4650  5.8738 
## 
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)    
## (Intercept)               3.198e+01  5.479e+00   5.836 1.15e-08 ***
## Income                   -1.197e-02  3.603e-02  -0.332 0.739994    
## Advertising               1.844e-01  1.689e-01   1.092 0.275692    
## Population               -1.524e-02  6.943e-03  -2.196 0.028718 *  
## Price                    -1.412e-01  3.647e-02  -3.873 0.000127 ***
## Age                      -1.779e-01  6.016e-02  -2.957 0.003304 ** 
## Education                -7.970e-01  3.145e-01  -2.534 0.011691 *  
## `Income:Advertising`      8.955e-07  6.728e-04   0.001 0.998939    
## `Income:Population`       2.765e-05  3.051e-05   0.906 0.365363    
## `Income:Price`           -2.769e-05  1.730e-04  -0.160 0.872911    
## `Income:Age`             -8.299e-05  2.633e-04  -0.315 0.752806    
## `Income:Education`        1.715e-03  1.563e-03   1.097 0.273138    
## `Advertising:Population`  1.656e-05  1.258e-04   0.132 0.895325    
## `Advertising:Price`      -1.041e-04  7.776e-04  -0.134 0.893572    
## `Advertising:Age`        -2.912e-04  1.132e-03  -0.257 0.797200
```

```
## `Advertising:Education`  -2.539e-03  7.096e-03  -0.358 0.720730
## `Population:Price`        3.316e-05  3.375e-05   0.983 0.326416
## `Population:Age`          2.720e-05  5.201e-05   0.523 0.601238
## `Population:Education`    4.971e-04  3.013e-04   1.650 0.099733 .
## `Price:Age`               6.722e-04  3.022e-04   2.224 0.026717 *
## `Price:Education`         3.017e-03  1.947e-03   1.549 0.122130
## `Age:Education`           3.579e-03  2.761e-03   1.296 0.195727
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.253 on 378 degrees of freedom
## Multiple R-squared:  0.3971, Adjusted R-squared:  0.3636
## F-statistic: 11.86 on 21 and 378 DF,  p-value: < 2.2e-16
```

#We now see rather large differences between multiple R^2, adjusted R^2 and the cross-validation estimate of R^2. Basically, this model is overfitting. It is too complicated and will not generalize well to new data.

#A regularized model, either Lasso or ridge would be a good choice to simplify the model, both for interpretation and for improving predictive performance. A good implementation of regularized linear models is in the `glmnet` package. Specifically, `glmnet` will fit a mixture of Lasso and ridge, with the mixture being controlled by `alpha`,

```
#Fit regularized model
set.seed(123)
(caret_glmnet <- train(ï..Sales ~ .^2,
                  method = "glmnet",
                  preProcess = c("center", "scale"),
                  data = cbind(ï..Sales = Carseats_Filtered$ï..Sales, imputed
)))
```

```
## glmnet
##
## 400 samples
##   6 predictor
##
## Pre-processing: centered (21), scaled (21)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 400, 400, 400, 400, 400, 400, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda       RMSE      Rsquared   MAE
##   0.10   0.002510041  2.361522  0.3178962  1.905897
##   0.10   0.025100406  2.339772  0.3283553  1.883909
##   0.10   0.251004065  2.316709  0.3408082  1.859425
##   0.55   0.002510041  2.359459  0.3187452  1.904111
##   0.55   0.025100406  2.325419  0.3361908  1.870193
##   0.55   0.251004065  2.329986  0.3391480  1.869673
##   1.00   0.002510041  2.357507  0.3195775  1.902392
##   1.00   0.025100406  2.316424  0.3410031  1.863776
```

```
##   1.00   0.251004065  2.348533  0.3379115  1.883887
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 1 and lambda = 0.02510041
.
```

#The estimated out-of-sample performance for this model has improved substantially over the unpenalized linear model. Without the seed estimated results will fluctuate each time the model is fit, since the observations in each cross-validation bootstrap sample will be randomly chosen and thus different. In this case the lowest estimated out-of-sample RMSE (and highest R^2) is with the `alpha` and `lambda` combination in the second to last row.

#Here `alpha` and `lambda`. The procedure is to produce a cross-validation estimate of the model's out-of-sample performance at each default combination of the hyperparameters. In this case, the best performing model is at `alpha` = 1, which is a Lasso model (a ridge model would be `alpha` = 0).

#One of the virtues of a lasso model is its simplicity, since many predictors will have been shrunk to zero, and thereby removed from the model. We can retrieve the coefficients from the above model with the following code:

```
# Retrieve coefficients of the best model
coef(caret_glmnet$finalModel, caret_glmnet$finalModel$tuneValue$lambda)
```

```
## 22 x 1 sparse Matrix of class "dgCMatrix"
##                               s1
## (Intercept)            7.49632500
## Income                 0.05139677
## Advertising            0.58315969
## Population            -0.02824022
## Price                 -1.32364242
## Age                   -0.71716178
## Education             -0.13350849
## Income:Advertising     0.25321145
## Income:Population      .
## Income:Price           .
## Income:Age             .
## Income:Education       0.14824246
## Advertising:Population .
## Advertising:Price      .
## Advertising:Age        .
## Advertising:Education  .
## Population:Price       .
## Population:Age        -0.08686404
## Population:Education   .
## Price:Age              .
## Price:Education        .
## Age:Education          .
```

#- `coef()` is a function that pulls coefficients out of the model object. #- `object\$finalModel` extracts the model that `caret`, after having conducted a grid search of optimal `alpha` hyperparameter settings, identifies as the one that will generalize best to new data. #- `object\$finalModel\$tuneValue\$lambda` extracts from the among the models with the best alpha the one with the optimal `lambda` hyperparameter.

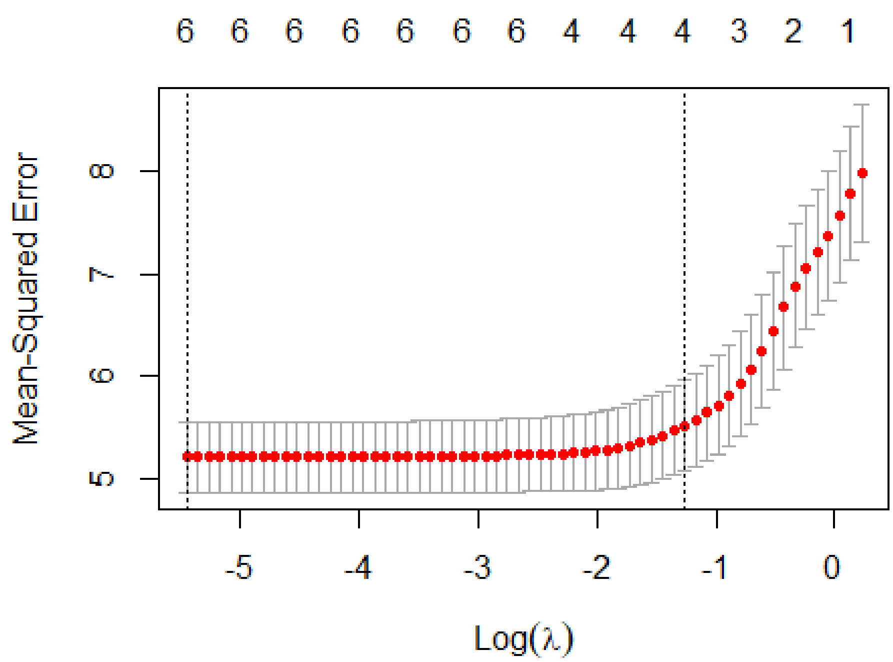
#The coefficients that have been removed are represented with dots. The result is a much simpler model that will tend to generalize better than a complicated model to new data.

###Fit a linear model with Lasso

```
library(glmnet)
library(faraway)

## Warning: package 'faraway' was built under R version 4.1.2

##
## Attaching package: 'faraway'

## The following object is masked from 'package:rpart':
##
##     solder

## The following object is masked from 'package:lattice':
##
##     melanoma

set.seed(1233)
data("Carseats_Filtered")

## Warning in data("Carseats_Filtered"): data set 'Carseats_Filtered' not fou
nd

head(Carseats_Filtered)

##   ï..Sales Income Advertising Population Price Age Education
## 1     9.50     73          11        276   120  42        17
## 2    11.22     48          16        260    83  65        10
## 3    10.06     35          10        269    80  59        12
## 4     7.40    100           4        466    97  55        14
## 5     4.15     64           3        340   128  38        13
## 6    10.81    113          13        501    72  78        16

library(faraway)
set.seed(123)
#we need to define the model equation
  X <- model.matrix(ï..Sales ~  Price + Advertising + Population + Age + Inco
me + Education, data= Carseats_Filtered)[,-1]
#and the outcome
  head(Carseats_Filtered)
```

```
##   ï..Sales Income Advertising Population Price Age Education
## 1     9.50     73          11        276   120  42        17
## 2    11.22     48          16        260    83  65        10
## 3    10.06     35          10        269    80  59        12
## 4     7.40    100           4        466    97  55        14
## 5     4.15     64           3        340   128  38        13
## 6    10.81    113          13        501    72  78        16
```

```
  Y <- Carseats_Filtered[,1]
  #Penalty type (alpha=1 is lasso
#and alpha=0 is the ridge)
  fit <- cv.glmnet(X,Y,alpha = 1)
 #MSE for several lambdas
 plot(fit)
```




```
print(fit)

##
## Call:  glmnet::cv.glmnet(x = X, y = Y, alpha = 1)
##
## Measure: Mean-Squared Error
##
##      Lambda Index Measure     SE Nonzero
## min 0.00431    62   5.199 0.3442       6
## 1se 0.28326    17   5.513 0.4472       4
```

#Effect ofb Lambda for Lasso

```
#An unregularized linear regression
#model using the old lm( ) function.
#OLS (λ=0)
lm_fit=lm(Y~X)
as.data.frame(lm_fit$coefficients)
```

```
##               lm_fit$coefficients
## (Intercept)         15.9570841633
## XPrice              -0.0573886222
## XAdvertising         0.1254062935
## XPopulation         -0.0009311833
## XAge                -0.0489852497
## XIncome              0.0104575575
## XEducation          -0.0364657266
```

```
#The model's coefficients when lambda is set to 0.01
#(λ=0.01)
coef(fit,s=0.01)
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                        s1
## (Intercept) 15.8120357462
## Price       -0.0569053296
## Advertising  0.1233400736
## Population  -0.0008269647
## Age         -0.0482649088
## Income       0.0101796053
## Education   -0.0324465331
```

```
#The model's coefficients when lambda was set to 0.1
#(λ=0.1)
coef(fit,s=0.1)
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept) 14.590306998
## Price       -0.052573918
## Advertising  0.105366130
## Population   .
## Age         -0.041822865
## Income       0.007643889
## Education    .
```

#function. The coefficients for those variables that are removed from the model are #shown by a dot. In other words, the coefficient for those variables is zero. ##Setting the lambda to 0.1 results in only 2 coefficients forced to zero where we are left with 4 non zero coefficients

```
#(λ=1)
coef(fit,s=1)
```

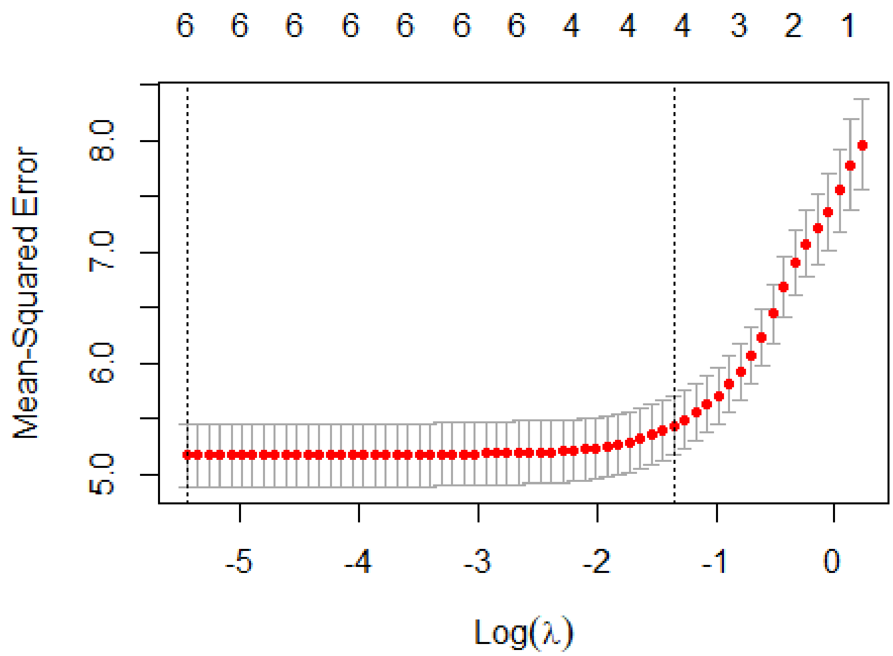
```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                      s1
## (Intercept)  8.74510987
## Price       -0.01078445
## Advertising  .
## Population   .
## Age          .
## Income       .
## Education    .
```

#Further increasing the lambda to 1 results in 5 coefficients forced to zero where we now have only 2 non zero #Also he absolute value of the predictor that remains in the model shrinks as we increase the lambda.

Cross validation

#The function tries different values of lambda to find the optimal choice. The optimal choice is the one that minimizes the cross validation mean square of the error. #lambda.min represents the lambda value that is optimal and minimizes the cross validation mean square of the error.

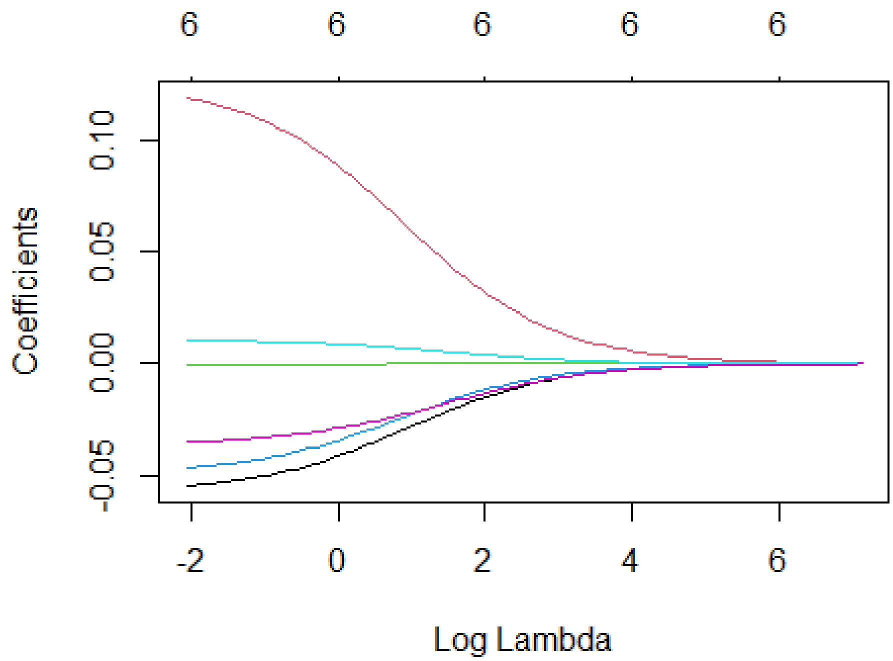
```
cvfit = cv.glmnet(X,Y)
cvfit

##
## Call:  glmnet::cv.glmnet(x = X, y = Y)
##
## Measure: Mean-Squared Error
##
##      Lambda Index Measure     SE Nonzero
## min 0.00431    62   5.167 0.2802       6
## 1se 0.25810    18   5.439 0.2670       4

plot(cvfit)
```



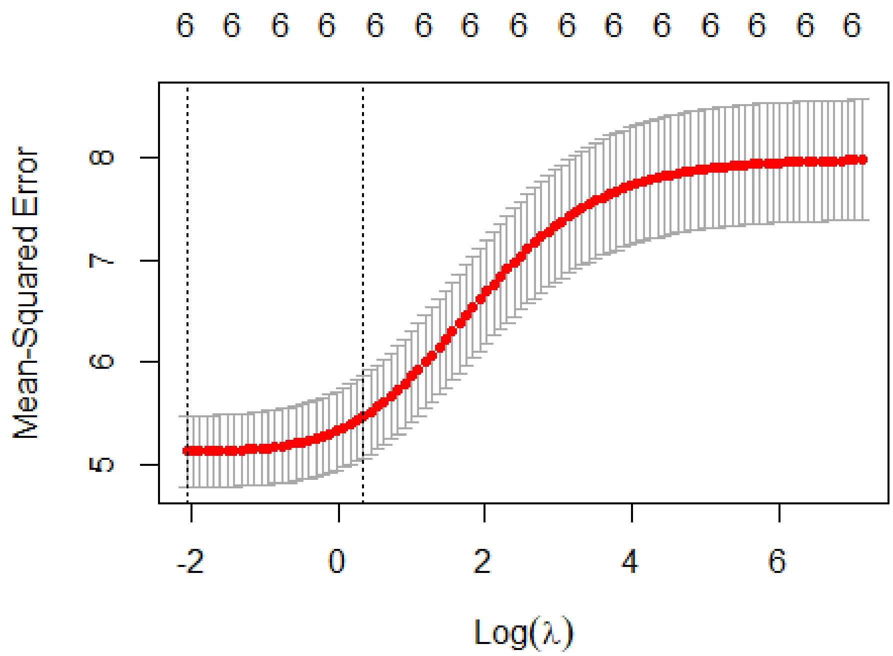

```
#This and next...
cvfit = cv.glmnet(X, Y)
coef(cvfit, s = "lambda.min")

## 7 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept) 15.8945811275
## Price       -0.0571800982
## Advertising  0.1245132007
## Population  -0.0008862575
## Age         -0.0486750291
## Income       0.0103379764
## Education   -0.0347353012

cvfit = cv.glmnet(X, Y, type.measure = "mae", nfolds = 5)
coef(cvfit, s ="lambda.min")

## 7 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept) 15.8945811275
## Price       -0.0571800982
## Advertising  0.1245132007
## Population  -0.0008862575
## Age         -0.0486750291
## Income       0.0103379764
## Education   -0.0347353012
```

#Ridge, Elastic Net, and Lasso Comparison

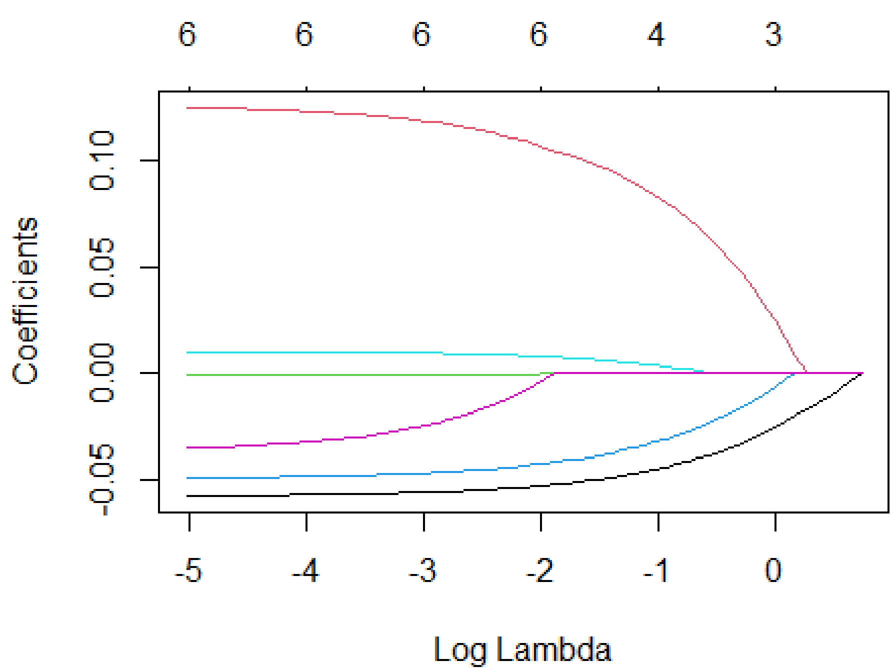
```
#Ridge
fit.ridge<-glmnet(x=X, y=Y, alpha = 0)
plot(fit.ridge, xvar = "lambda")
```



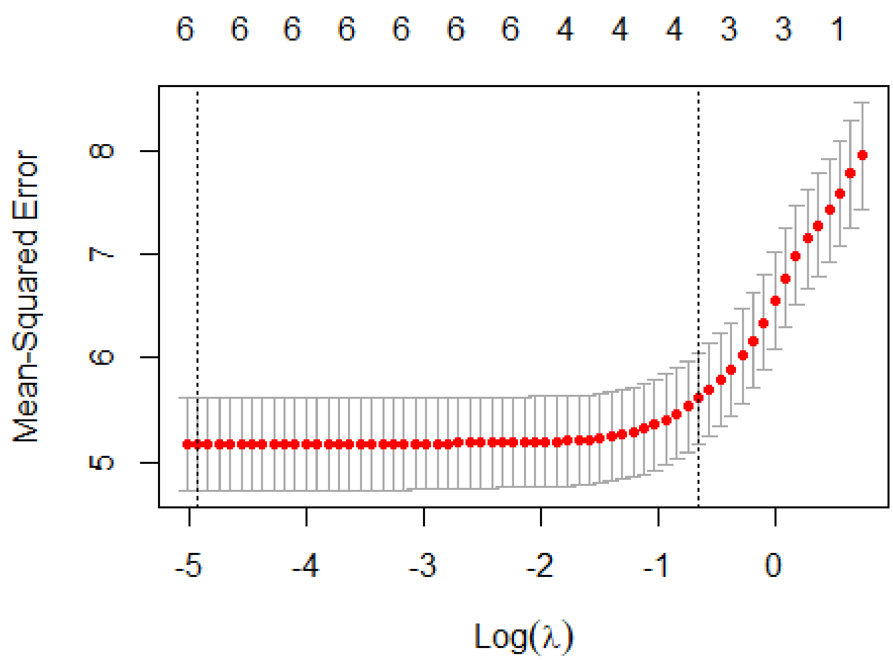

```
plot (cv.glmnet (x=X, y=Y,alpha =0))
```




```r
#Elastic Net
fit.elnet<-glmnet(x=X, y=Y, alpha = 0.6)
plot(fit.elnet, xvar = "lambda")
```

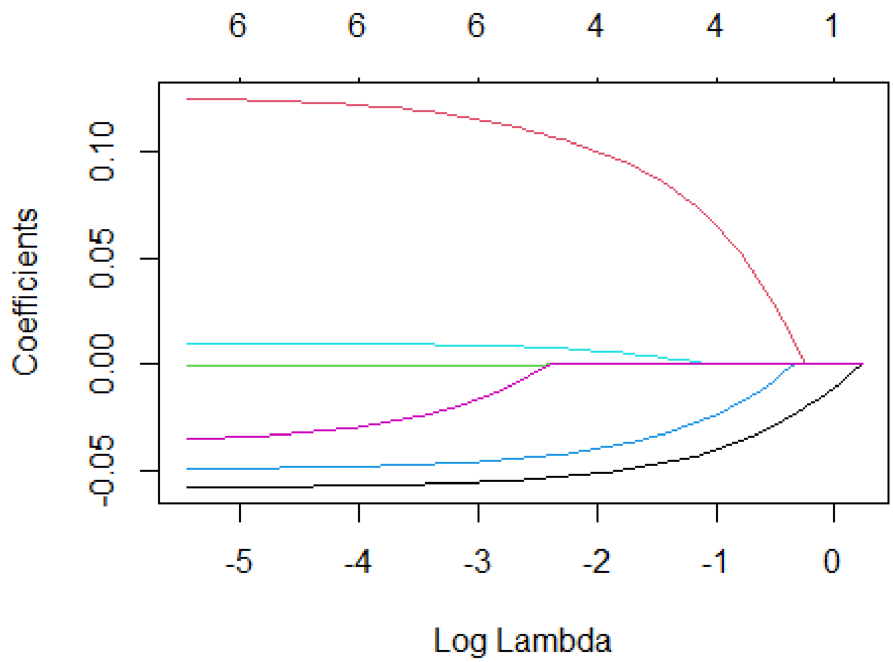


```
plot(cv.glmnet (x=X, y=Y,alpha =0.6))
```




```
print(fit.elnet)

##
## Call:  glmnet::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

#Lasso
fit.lasso<-glmnet(x=X, y=Y, alpha = 1)
plot(fit.lasso, xvar = "lambda")
```




```
print(fit.lasso)

##
## Call:  glmnet::glmnet(x = X, y = Y, alpha = 1)
##
##     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
```