HOMEWORK 8. ISYE 6501

Guillermo de la Hera Casado

October 13th, 2019

Reading the data and creating the regression model

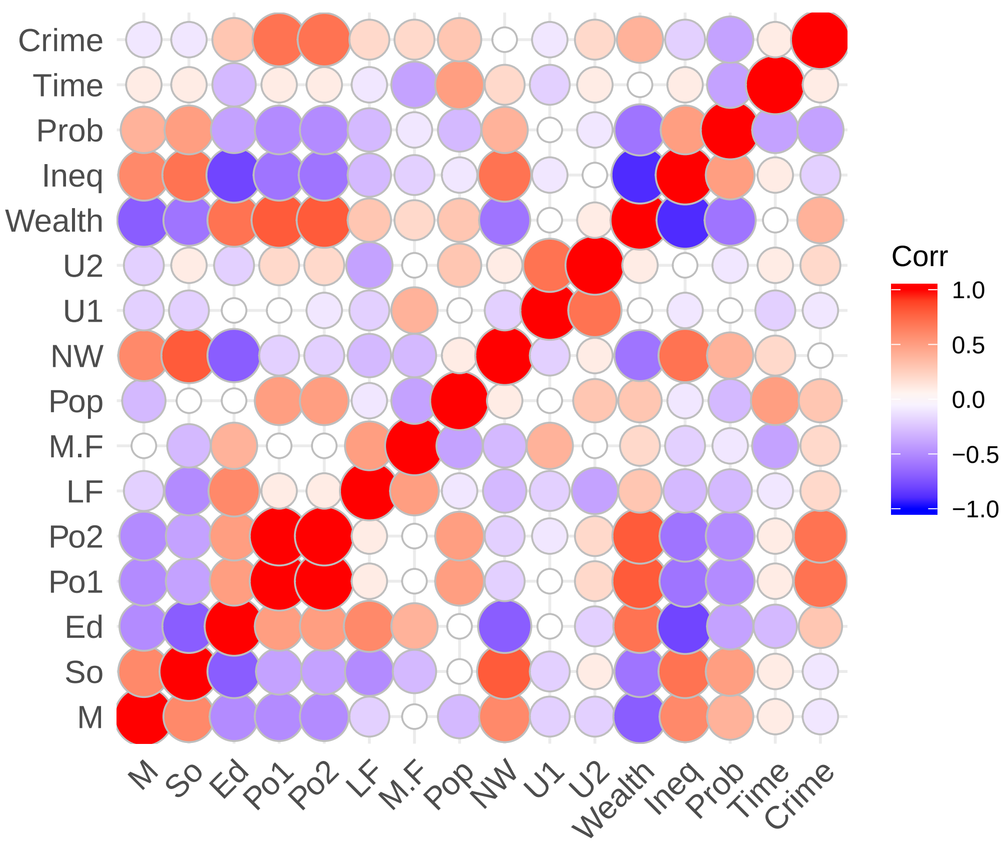
Let's first input the *Crime* data:

```
set.seed(101) # Set Seed so that same sample can be reproduced in future also
usCrime <- read.table("uscrime.txt", header = TRUE, sep = "\t")
```

We may also want to check if the predictors are correlated, so that we can understand better the variable selection outcomes:

- There is a strong negative correlation between Wealth and Ineq
- There is a strong positive correlation between Po1 and Po2

```
corr <- round(cor(usCrime), 1)
ggcorrplot(corr, method = "circle")
```



Let's also leave 30% of the rows aside, just for ilustration purposes when computing Adjusted R Square through each method:

```
sample <- sample.split(usCrime$Crime, SplitRatio = 0.7)
trainSet <- subset(usCrime, sample == TRUE)
```

```r
testSet <- subset(usCrime, sample == FALSE)
```

Finally, we are going to fit a regression model with all the variables. As we can see, we have: *Ed, Po1, NW and Ineq* variables as statistically significant at 90% confidence level. These would be potentially variables interesting to keep from a p-value perspective:

```r
full <- lm(Crime~., data = trainSet,na.action = "na.fail")
summary(full)
```

```
##
## Call:
## lm(formula = Crime ~ ., data = trainSet, na.action = "na.fail")
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -264.61  -93.00  -16.56  123.74  256.56
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -6945.9813  1795.1556  -3.869 0.001359 **
## M              45.0899    60.7094   0.743 0.468418
## So           -118.6905   167.0925  -0.710 0.487726
## Ed            324.4806    64.4846   5.032 0.000123 ***
## Po1           228.7795   117.0537   1.954 0.068349 .
## Po2          -188.3460   118.5285  -1.589 0.131615
## LF          -2290.2625  1555.5786  -1.472 0.160340
## M.F            27.7238    23.5689   1.176 0.256686
## Pop             0.5869     1.7744   0.331 0.745137
## NW             12.8895     7.0885   1.818 0.087776 .
## U1          -8570.9776  5312.2018  -1.613 0.126193
## U2            146.6381    86.6995   1.691 0.110155
## Wealth          0.1090     0.1195   0.912 0.375355
## Ineq           97.0862    23.1775   4.189 0.000695 ***
## Prob        -3338.7499  2673.4194  -1.249 0.229677
## Time           -5.7352    11.9761  -0.479 0.638500
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 176 on 16 degrees of freedom
## Multiple R-squared:  0.8741, Adjusted R-squared:  0.7561
## F-statistic: 7.407 on 15 and 16 DF,  p-value: 0.0001298
```

Stepwise regression based on AIC

```r
SR_AIC <- stepAIC(full, direction = "both", trace = FALSE)
SR_AIC$anova
```

```
## Stepwise Model Path
## Analysis of Deviance Table
##
## Initial Model:
## Crime ~ M + So + Ed + Po1 + Po2 + LF + M.F + Pop + NW + U1 +
##     U2 + Wealth + Ineq + Prob + Time
##
```

```
## Final Model:
## Crime ~ Ed + Po1 + Po2 + NW + U1 + U2 + Ineq + Prob
##
##
##       Step Df  Deviance Resid. Df Resid. Dev      AIC
## 1                              16   495371.9 340.7145
## 2    - Pop  1  3386.644        17   498758.5 338.9325
## 3   - Time  1  3877.733        18   502636.3 337.1804
## 4      - M  1 12412.762        19   515049.0 335.9610
## 5 - Wealth  1 16291.460        20   531340.5 334.9575
## 6     - So  1 23296.843        21   554637.3 334.3307
## 7     - LF  1 34332.437        22   588969.8 334.2526
## 8    - M.F  1 29773.905        23   618743.7 333.8307
```

```
lm_AIC <- lm(Crime~Ed+Po1+Po2+NW+U1+U2+Ineq+Prob, trainSet)
pred <- predict(lm_AIC,
                testSet[,c("Ed","Po1", "Po2","NW","U1","U2","Ineq","Prob")])

n <- length(testSet$Crime)
k <- 8
CV_residual <- pred - testSet$Crime
SSyy <- sum((testSet$Crime - mean(testSet$Crime))^2)
SSE <- sum(CV_residual^2)
Adj_R_Squared <- 1 - ((SSE/n-k) / (SSyy/n-1))
print(Adj_R_Squared)
```

```
## [1] 0.2708079
```

The final model that minimizes AIC would be based on the following 8 predictors: **Ed, Po1, Po2, NW, U1, U2, Ineq, Prob**. **The adjusted R-Square based on test set = 0.27**

Stepwise regression based on BIC

```
SR_BIC <- step(full, direction = "both", k=log(nrow(usCrime)),trace = FALSE)
SR_BIC$anova
```

```
##        Step Df  Deviance Resid. Df Resid. Dev      AIC
## 1           NA        NA        16   495371.9 370.3169
## 2     - Pop  1  3386.644        17   498758.5 366.6847
## 3    - Time  1  3877.733        18   502636.3 363.0824
## 4       - M  1 12412.762        19   515049.0 360.0129
## 5  - Wealth  1 16291.460        20   531340.5 357.1593
## 6      - So  1 23296.843        21   554637.3 354.6823
## 7      - LF  1 34332.437        22   588969.8 352.7541
## 8     - M.F  1 29773.905        23   618743.7 350.4821
## 9      - U1  1 41504.808        24   660248.5 348.7095
## 10     - U2  1 23348.956        25   683597.4 345.9715
## 11   - Prob  1 55858.184        26   739455.6 344.6348
## 12    - Po2  1 60219.919        27   799675.5 343.2900
## 13     - NW  1 64626.024        28   864301.6 341.9267
```

```
lm_BIC <- lm(Crime~Ed+Po1+Ineq, trainSet)
pred <- predict(lm_BIC, testSet[,c("Ed","Po1","Ineq")])
n <- length(testSet$Crime)
k <- 3
```

```
CV_residual <- pred - testSet$Crime
SSyy <- sum((testSet$Crime - mean(testSet$Crime))^2)
SSE <- sum(CV_residual^2)
Adj_R_Squared <- 1 - ((SSE/n-k) / (SSyy/n-1))
print(Adj_R_Squared)
```

```
## [1] 0.2321677
```

As we know, BIC penalizes more than AIC having a large number of predictors. It's not strange to see that **it only keeps 3 predictors: Ed, Po1, Ineq**.
The Adj R Squared on the training set is only: 0.23

Dredging - MuMIn package

Dredging executes automated model selection, testing all different combinations of predictors. Part of the *MuMIn* package, It's more expensive computationally but may provide a comprehensive analytical solution. Let's run it using BIC for ranking and asking also to get the adjusted R-squared metric. We want to display the best model out of all computations:

```
dredge_BIC <- dredge(full, evaluate = TRUE, rank = "BIC",extra= c("adjR^2"),trace = FALSE)
```

```
## Fixed term is "(Intercept)"
```

```
summary(get.models(dredge_BIC, 1)[[1]])
```

```
##
## Call:
## lm(formula = Crime ~ Ed + Ineq + Po1 + 1, data = trainSet, na.action = "na.fail")
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -321.37 -140.47   23.15   91.46  335.86
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4935.46     697.12  -7.080 1.06e-07 ***
## Ed            281.03      44.56   6.307 8.05e-07 ***
## Ineq          106.78      13.52   7.896 1.34e-08 ***
## Po1            95.86      13.64   7.027 1.21e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 175.7 on 28 degrees of freedom
## Multiple R-squared:  0.7804, Adjusted R-squared:  0.7568
## F-statistic: 33.16 on 3 and 28 DF,  p-value: 2.347e-09
```

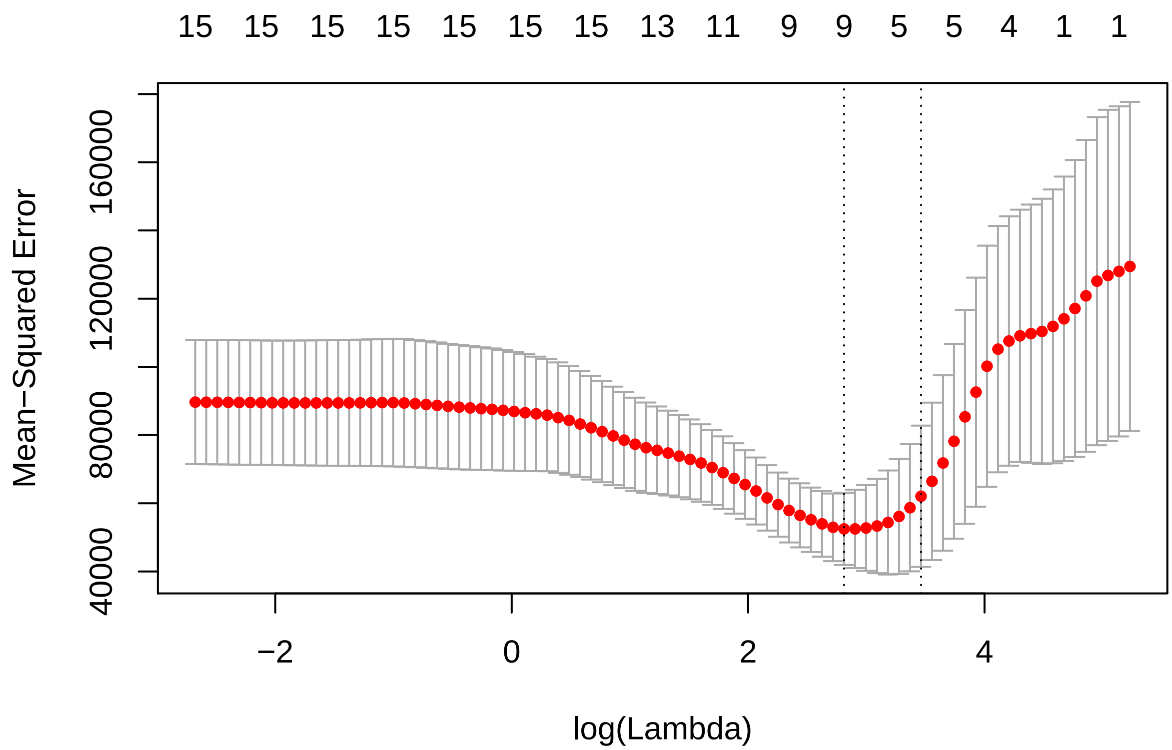
As we can see, the best model includes the following predictors: **Ed, Ineq,and Po1**. It's quite interesting to see that the selection given is the same as the one returned by step BIC! The adjusted R-squared is overestated as it's computed on the training data. We could expect similar performance on the test data as Step BIC.

Lasso

Since the number of rows in our dataset is quite small, We will use the function *cv.glmnet* to run cross validation and determine the best value of lambda for the model (lambda being the regulatization factor).

We will also set up alpha = 1 since we want to run Lasso. We also have to determine the family. We will use *gaussian* as we just have one response variable - we are not interested in studying the relationship between more than 1 response. As the final step, we will set standardize = TRUE in order to have variables on the same units:

```
lasso <- cv.glmnet(as.matrix(trainSet[,1:15]), trainSet[,16], alpha = 1,
                   family = "gaussian", standardize = TRUE)
plot(lasso)
```




```
lasso$lambda.min
```

```
## [1] 16.64044
```

```
coef(lasso, s = "lambda.min")
```

```
## 16 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept) -3.707716e+03
## M            4.081903e-01
## So           .
## Ed           2.227546e+02
## Po1          7.298448e+01
## Po2          .
## LF           .
## M.F          2.102779e+00
## Pop          9.189864e-02
## NW           6.249716e+00
## U1           .
```

```
## U2           6.351092e+00
## Wealth       .
## Ineq         7.313509e+01
## Prob        -1.511193e+03
## Time         .
```

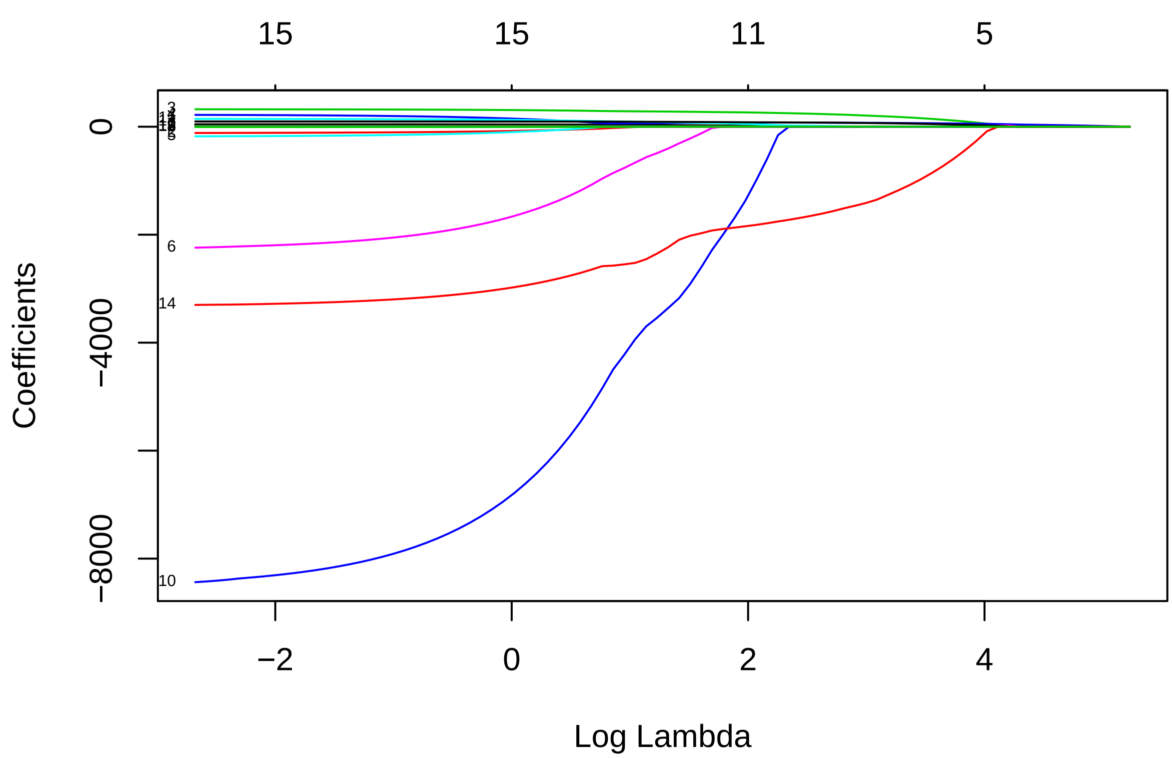
```r
yhat0 <- predict(lasso, s=lasso$lambda.min, newx = as.matrix(testSet[,1:15]))
n <- length(testSet$Crime)
k <- 9
CV_residual <- yhat0 - testSet$Crime
SSE <- sum(CV_residual^2)
SSyy <- sum((testSet$Crime - mean(testSet$Crime))^2)
Adj_R_Squared <- 1 - ((SSE/n-k) / (SSyy/n-1))
Adj_R_Squared
```

```
## [1] 0.3678987
```

Let's analyze the results. By plotting the MSE trend on the basis of log lambda. By looking at *lambda.min* variable, we can see that the best value of lambda that minimizes MSE is: 16.6. We can also see that this optimal configuration is based only on 9 predictors: **M, Ed, Po1, M.F, Pop, NW, U2, Ineq, Prob**. **The adjusted R Squared obtained on the test data is: 0.367**

If we wanted to see how the coefficients shrink to zero when lambda increases, we can easily fit the model with *glmnet* function:

```r
lassot <- glmnet(as.matrix(trainSet[,1:15]), trainSet[,16], alpha = 1,
                  family = "gaussian", standardize = TRUE)
plot(lassot, xvar="lambda", label=TRUE)
```

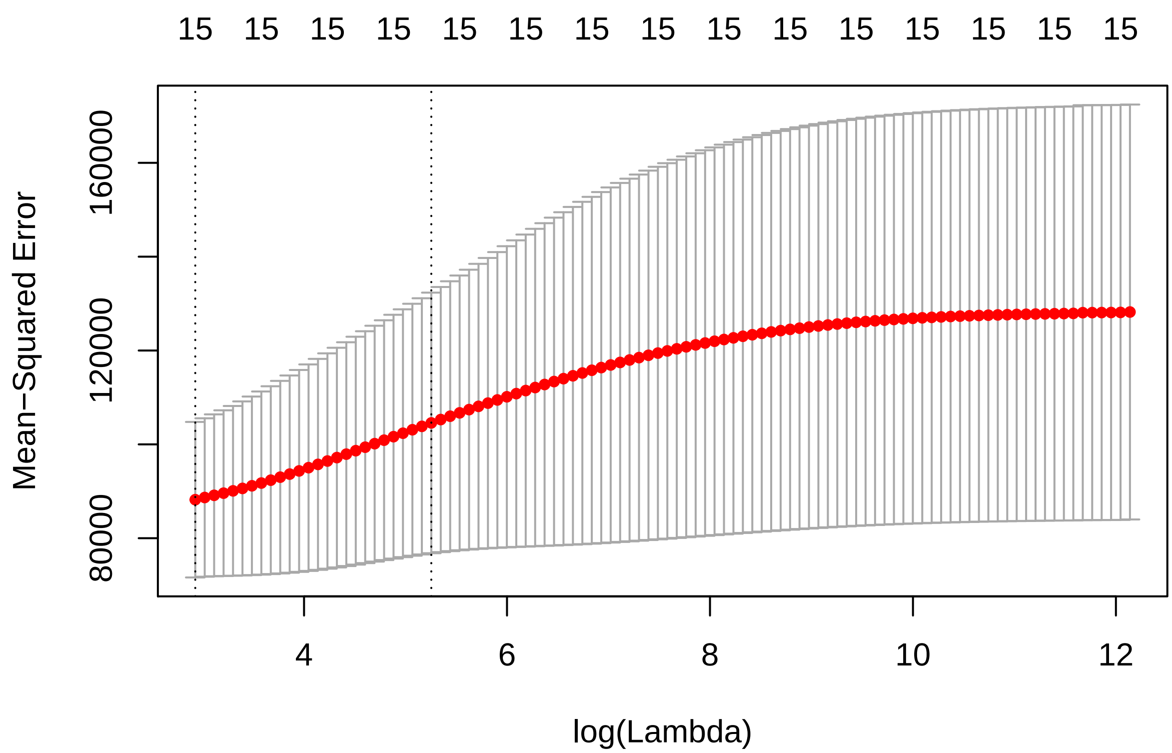


As seen in the plot, coefficient values are shrinked to zero slowly but surely, when lambda is increased.

Ridge

We will use *alpha = 0* to set up a Ridge regression. Let's look at the plot:

```
ridge<- cv.glmnet(as.matrix(trainSet[,1:15]), trainSet[,16], alpha = 0,
                  family = "gaussian", standardize = TRUE)
plot(ridge)
```




```
ridge$lambda.min
```

```
## [1] 18.6926
```

```
coef(ridge, s = "lambda.min")
```

```
## 16 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept) -5.551698e+03
## M            5.345077e+01
## So           3.383032e+01
## Ed           2.185640e+02
## Po1          4.772196e+01
## Po2          1.633228e+01
## LF          -5.543232e+02
## M.F          1.860797e+01
## Pop          1.112293e+00
```

```
## NW            7.900396e+00
## U1           -3.315465e+03
## U2            8.267001e+01
## Wealth        5.186639e-02
## Ineq          6.640476e+01
## Prob         -3.391143e+03
## Time         -5.814991e+00
```

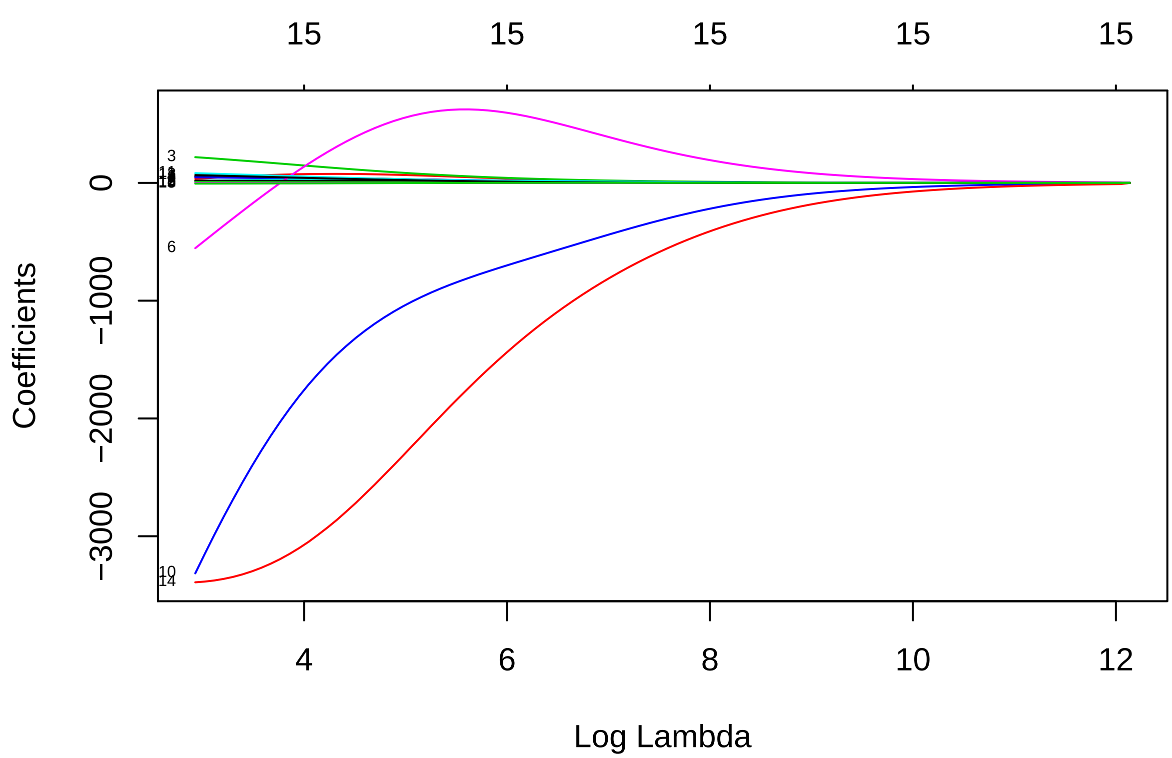
```
yhat0 <- predict(ridge, s=ridge$lambda.min, newx = as.matrix(testSet[,1:15]))
n <- length(testSet$Crime)
k <- 15
CV_residual <- yhat0 - testSet$Crime
SSE <- sum(CV_residual^2)
SSyy <- sum((testSet$Crime - mean(testSet$Crime))^2)
Adj_R_Squared <- 1 - ((SSE/n-k) / (SSyy/n-1))
Adj_R_Squared
```

```
## [1] 0.5482899
```

As we see, Ridge never shrinks the coefficient to zero value, **that's why the number of coefficients remains = 15** for any lambda selection. The best value of lambda that minimizes MSE is: 18.69 . **The adjusted R Squared obtained on the test set with this config is: 0.548**

Let's check this fact looking at the following plot:

```
ridget <- glmnet(as.matrix(trainSet[,1:15]), trainSet[,16], alpha = 0,
                 family = "gaussian", standardize = TRUE)
plot(ridget, xvar="lambda", label=TRUE)
```

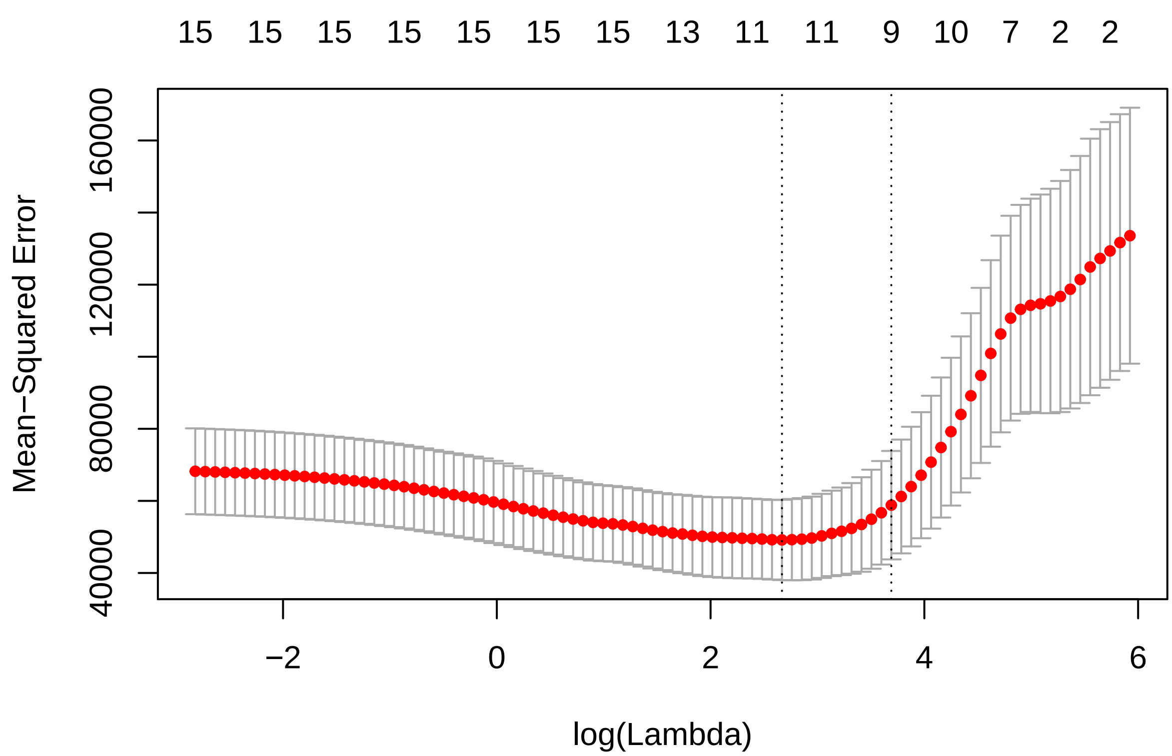


Easy to see here as well how the coefficients get very near to zero value, but none equal zero regardless of the lambda value.

Elastic Net

Let's try alpha: 0.5, as a good example of Elastic Net, combining both Lasso and Ridge at the same rate.

```
elNet<- cv.glmnet(as.matrix(trainSet[,1:15]), trainSet[,16], alpha = 0.5,
                   family = "gaussian", standardize = TRUE)
plot(elNet)
```




```
elNet$lambda.min
```

```
## [1] 14.40649
```

```
coef(elNet, s = "lambda.min")
```

```
## 16 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept) -4755.329209
## M              21.855358
## So              .
## Ed            235.325413
## Po1            72.881007
## Po2             .
## LF              .
## M.F             8.717775
## Pop             0.314484
```

```
## NW            7.043841
## U1        -1505.999196
## U2           50.850918
## Wealth        .
## Ineq         71.313596
## Prob      -1961.102122
## Time          .
```

```
yhat0 <- predict(elNet, s=elNet$lambda.min, newx = as.matrix(testSet[,1:15]))
n <- length(testSet$Crime)
k <- 10
CV_residual <- yhat0 - testSet$Crime
SSE <- sum(CV_residual^2)
SSyy <- sum((testSet$Crime - mean(testSet$Crime))^2)
Adj_R_Squared <- 1 - ((SSE/n-k) / (SSyy/n-1))
Adj_R_Squared
```

```
## [1] 0.4547708
```

As we can see, it shrinks the value of the coefficients (Ridge) but also shrinks to zero the coefficient of the variables: *So, Po2, LF, Wealth, Time* (Lasso) **The adjusted R-Squared obtained on the training set from this config is: 0.45**

Summary

- Going through the assignment, it can be noted that those methods with a more restrictive regularization pattern (BIC, Lasso) or too computationally expensive on a small set (Dredging) are the ones scoring the lowest adjusted R-Squared on the test data. The reason may be just the size of the dataset. It's very easy to overfit and those methods may be dropping variables that are still making an impact on the response, even with small coefficients. Not surprisingly the best score is given by Ridge, whose purpose is to generalize the model by shrinking the coefficients without dropping the variables
- Looking at the variable selection given by the different algorithms, we can see that **Ed, Ineq, Po1** are kept across all of them. It's also visible that variables like **Time, Wealth, Po2** are not really important across the board. The reason may be that Po2 correlates to Po1, and Wealth to Ineq. If Ineq and Po1 are selected as important variables, the other two loses predictive power.