ISYE 6501 - Week 5 Homework

Ujjawal Madan

11/06/2020

Contents

0.1 Question 11.1

Using the crime data set uscrime.txt from Questions 8.2, 9.1, and 10.1, build a regression model using:

- Stepwise regression
- Lasso
- Elastic net

For Parts 2 and 3, remember to scale the data first – otherwise, the regression coefficients will be on different scales and the constraint won't have the desired effect. For Parts 2 and 3, use the glmnet function in R.

Let's start by importing the data and preparing it.

```
set.seed(1, sample.kind = 'Rounding')

#Import file
us_crime <- read_delim("http://www.statsci.org/data/general/uscrime.txt", "\t", escape_double = FALSE, tri

attributes <- as.data.frame(scale(us_crime[,1:15]))
response <- as.data.frame(us_crime[,16])
response <- as.vector(unlist(response))
scaled_crime <- cbind(attributes, response)
```

Before we build the models, let's create a helper function that can evaluate the effectiveness of our model. This helper function will return the RMSE as well as the R Squared value.

```
results <- function(true, predicted, df) {
  SSE <- sum((predicted - true)^2)
  SST <- sum((true - mean(true))^2)
  R_square <- 1 - SSE / SST
  RMSE = sqrt(SSE/nrow(df))

  data.frame(
    RMSE = RMSE,
    Rsquare = R_square
  )

}
```

Let's start by running a linear model using all the attributes.

```
full_model <- lm(response ~., data = scaled_crime)
print(full_model)
```

```
##
## Call:
## lm.default(formula = response ~ ., data = scaled_crime)
##
## Coefficients:
## (Intercept)            M           So           Ed          Po1          Po2
##     905.085      110.382       -1.822      210.678      572.995     -305.958
##          LF          M.F          Pop           NW           U1           U2
##     -26.826       51.293      -27.906       43.234     -105.056      141.714
##      Wealth         Ineq         Prob         Time
##      92.792      281.954     -110.394      -24.655
```

```
predictions<- predict(full_model, scaled_crime)
full_linearmodel_results <- results(response, predictions, scaled_crime)
kable(full_linearmodel_results)
```

RMSE	Rsquare
169.79	0.8030868

Great! We have a baseline from which we can compare subsequent models.

Let's now build one using a stepwise greedy algorithm in both directions.

```
step_model <- stepAIC(full_model, direction = "both",
                      trace = FALSE)
print(step_model)
```

```
##
## Call:
## lm.default(formula = response ~ M + Ed + Po1 + M.F + U1 + U2 +
##     Ineq + Prob, data = scaled_crime)
##
## Coefficients:
## (Intercept)            M           Ed          Po1          M.F           U1
##      905.09       117.28       201.50       305.07        65.83      -109.73
##          U2         Ineq         Prob
##      158.22       244.70       -86.31
```

So our stepwise model only uses 7 attributes. Let's see how well it does in comparison to the linear model with all the attributes.

```
predictions<- predict(step_model, scaled_crime)
stepwise_results <- results(response, predictions, scaled_crime)
kable(stepwise_results)
```

RMSE	Rsquare
175.8304	0.7888268

It does really quite well considering we are using less than half of the attributes for our prediction.

By accident, after doing the stepwise regression, I did ridge Regression so why don't we just include it. We start by running the model for various lambdas. We find the optimal lambda. And then we build our final model with it. Let's see how well it does now.

```
#set.seed(1, sample.kind = 'Rounding')

lambdas <- 10^seq(2, -2,-.1)
attributes <- as.matrix(attributes)
response <- as.matrix(response)

#Running it through with various lambda values
ridge_reg = glmnet(attributes, response, alpha = 0, lambda = lambdas)
print(summary(ridge_reg))
```

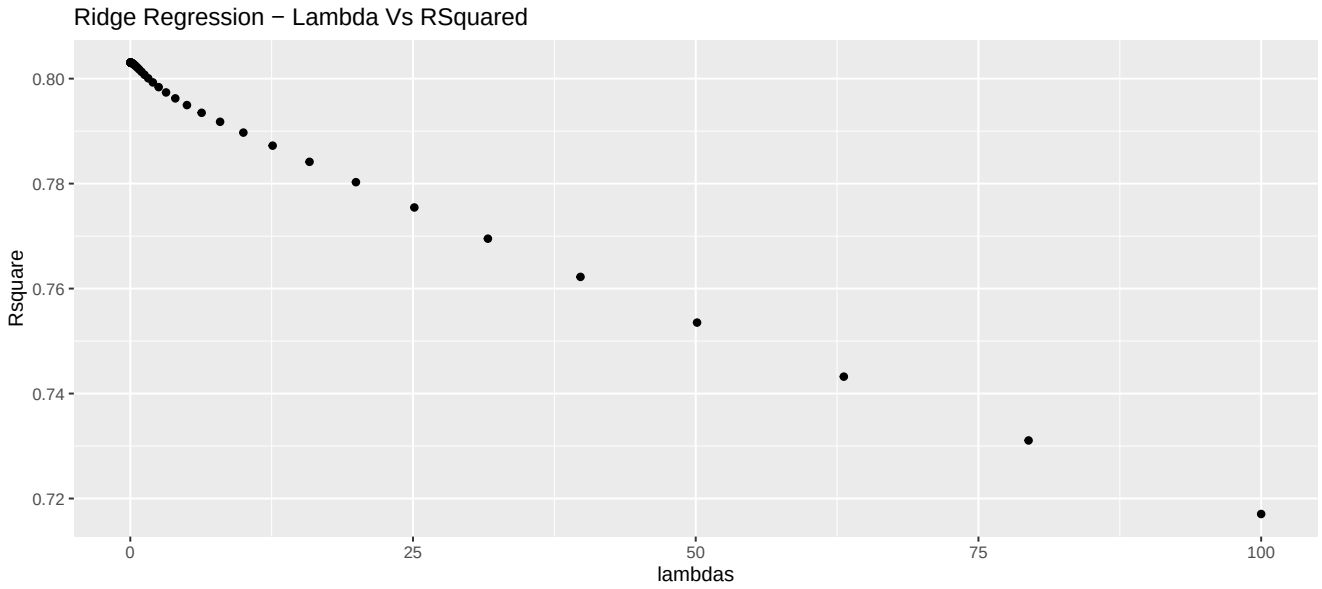
```
##           Length Class     Mode
## a0         41    -none-    numeric
## beta      615    dgCMatrix S4
## df         41    -none-    numeric
## dim         2    -none-    numeric
## lambda     41    -none-    numeric
## dev.ratio  41    -none-    numeric
## nulldev     1    -none-    numeric
## npasses     1    -none-    numeric
## jerr        1    -none-    numeric
## offset      1    -none-    logical
## call        5    -none-    call
## nobs        1    -none-    numeric
```

Out of curiousity, I wanted to see how lambda was affecting the Rsquared values and RMSE. Let's take a look...

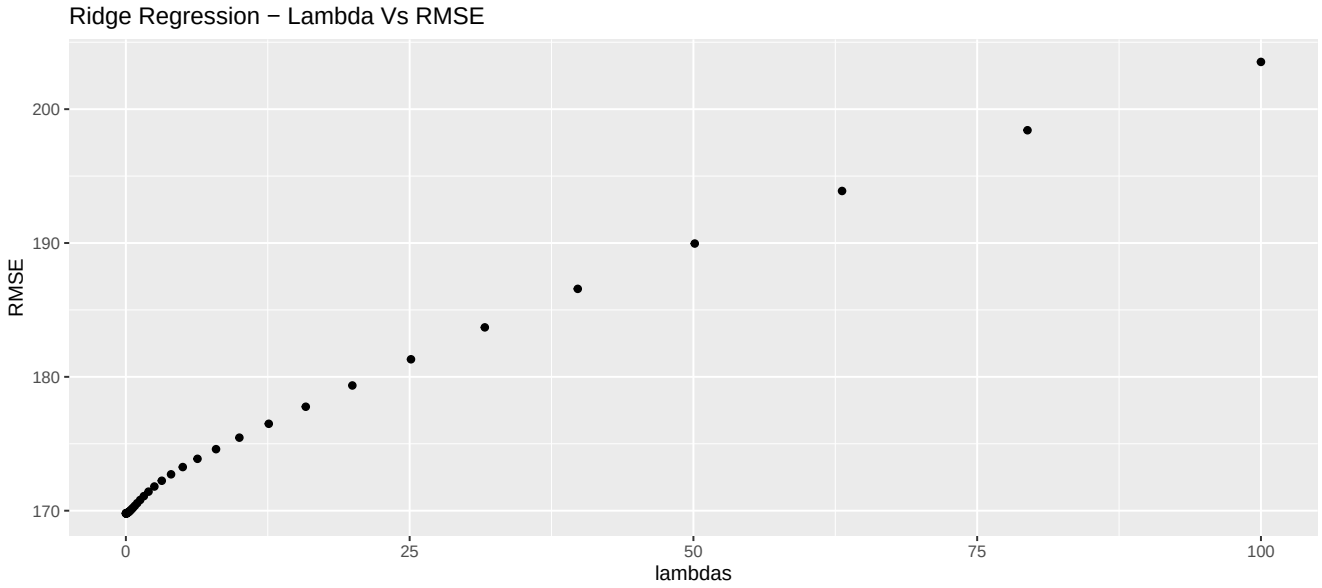
```
df <- data.frame(matrix(ncol = 3, nrow = 0))
colnames(df) <- c('Lambda Value', 'RMSE', 'RSquared')

for (i in seq(1, ncol(ridge_reg$beta))){
  predictions <- (attributes %*% ridge_reg$beta[,i]) + ridge_reg$a0
  values <- cbind(lambdas[i], results(response, predictions, scaled_crime))
  df <- rbind(df, values)
}

df %>% ggplot(aes(lambdas, Rsquare)) + geom_point() + ggtitle('Ridge Regression - Lambda Vs RSquared')
```




```
df %>% ggplot(aes(lambdas, RMSE)) + geom_point() + ggtitle('Ridge Regression - Lambda Vs RMSE')
```



It's interesting to visualize how our performance metrics change for different values of lambda. Let's find the optimal lambda now.

```
#Tuning Lamba
cv_ridge <- cv.glmnet(attributes, response, alpha = 0, lambda = lambdas)
optimal_lambda <- cv_ridge$lambda.min
print(optimal_lambda)
```

```
## [1] 63.09573
```

```
#Tuning Lambda
lambdas <- seq(optimal_lambda - optimal_lambda, optimal_lambda + 25, 2)
cv_ridge <- cv.glmnet(attributes, response, alpha = 0, lambda = lambdas)
optimal_lambda <- cv_ridge$lambda.min
print(optimal_lambda)
```

```
## [1] 48
```

```
#Final Model
best_ridge <- glmnet(attributes, response, alpha = 0, lambda = optimal_lambda)
coef(best_ridge)
```

```
## 16 x 1 sparse Matrix of class "dgCMatrix"
##                      s0
## (Intercept) 905.085106
## M            78.066391
## So           43.172317
## Ed          101.909208
## Po1         144.529075
## Po2         109.906746
## LF           22.874569
## M.F          69.516996
## Pop           1.597771
## NW           33.638430
## U1          -46.649439
## U2           82.239292
## Wealth       36.354728
## Ineq        135.474265
## Prob        -86.209365
## Time          4.579275
```

```
#Model Results
predictions <- predict(best_ridge, s = optimal_lambda, newx = attributes)
ridge_results <- results(response, predictions, scaled_crime)
kable(ridge_results)
```

RMSE	Rsquare
189.2632	0.7553287

Not bad! It also did quite well all things considered.

Let's move on to using the Lasso Model .The code follows a very similar template to Ridge Regression.

```
lambdas <- 10^seq(2, -3, by = -.1)

lasso_reg <- cv.glmnet(attributes, response, alpha = 1, lambda = lambdas)
optimal_lambda <- lasso_reg$lambda.min
print(optimal_lambda)
```

```
## [1] 3.981072
```

```
#plot(lasso_reg, xvar="lambda")

lambdas <- seq(optimal_lambda - optimal_lambda, optimal_lambda + 15, 2)
lasso_reg <- cv.glmnet(attributes, response, alpha = 1, lambda = lambdas)
optimal_lambda <- lasso_reg$lambda.min
print(optimal_lambda)
```

```
## [1] 12
```

```
best_lasso <- glmnet(attributes, response, alpha = 1, lambda = optimal_lambda)
coef(best_lasso)
```

```
## 16 x 1 sparse Matrix of class "dgCMatrix"
##                      s0
## (Intercept) 905.085106
## M            83.154763
## So           22.811517
## Ed          117.374497
## Po1         310.152432
## Po2           .
## LF            2.111077
## M.F          50.174855
## Pop           .
## NW            3.765910
## U1          -15.152597
## U2           48.172741
## Wealth        .
## Ineq        176.635742
## Prob        -80.644678
## Time          .
```

```
predictions <- predict(best_lasso, s = optimal_lambda, newx = attributes)
lasso_results <- results(response, predictions, scaled_crime)
kable(lasso_results)
```

RMSE	Rsquare
187.749	0.7592279

Our lasso regression also looks pretty decent. Let's finish if off with elastic net.

```
lambdas <- 10^seq(2, -3, by = -.1)

elastic <- cv.glmnet(attributes, response, alpha = 0.5, lambda = lambdas)
optimal_lambda <- elastic$lambda.min
print(optimal_lambda)
```

```
## [1] 7.943282
```

```
lambdas <- seq(optimal_lambda - optimal_lambda, optimal_lambda + 15, 2)
elastic <- cv.glmnet(attributes, response, alpha = 0.5, lambda = lambdas)
optimal_lambda <- elastic$lambda.min
print(optimal_lambda)
```

```
## [1] 6
```

```
best_elastic <- glmnet(attributes, response, alpha = 0.5, lambda = optimal_lambda)
coef(best_elastic)
```

```
## 16 x 1 sparse Matrix of class "dgCMatrix"
##                      s0
```

```
## (Intercept) 905.08511
## M           100.70893
## So           18.35615
## Ed          168.17654
## Po1         287.11887
## Po2           .
## LF            .
## M.F          57.12628
## Pop         -15.22462
## NW           19.15486
## U1          -71.10470
## U2          113.63518
## Wealth       53.09962
## Ineq        235.66351
## Prob        -90.51182
## Time          .
```

So as we can see, this ends up removing the LF value. Let's see how the model compares with the others.

```
predictions <- predict(best_elastic, s = optimal_lambda, newx = attributes)
elastic_results <- results(response, predictions, scaled_crime)
kable(elastic_results)
```

RMSE	Rsquare
174.4134	0.7922167

Also quite good! Let's put it all together now.

```
final_results <- rbind(full_linearmodel_results, stepwise_results, ridge_results, lasso_results, elastic_r
models <- c('Linear Regression', 'Stepwise Linear Regression', 'Ridge Regression', 'Lasso Regression', 'El
kable(cbind(models, final_results))
```

models	RMSE	Rsquare
Linear Regression	169.7900	0.8030868
Stepwise Linear Regression	175.8304	0.7888268
Ridge Regression	189.2632	0.7553287
Lasso Regression	187.7490	0.7592279
Elastic Net	174.4134	0.7922167

0.2 Question 12.1

Describe a situation or problem from your job, everyday life, current events, etc., for which a design of experiments approach would be appropriate.

Typically, anything related to polling would I believe require a design of experiments approach. Let me provide an example: If, let's say, we are trying to poll voters a week before their local state/province election, the poll needs to be carefully designed as to be representative of the larger population, whilst also be of a sufficient size so that the level of confidence in the results is high. Acquiring such data is not easy, requiring both time and money. As such, it is imperative for the pollster to design it so as to be able to accurately gauge voter opinions whilst remaining efficient in terms of cost.

Meeting the first requirement maybe the hardest challenge and may not be truly possible as polls inherently suffer

from systematic bias. Those who agree to share their opinions with the pollster, whether it's on the phone or in person, may be disproportionately in favour of a certain political party. And the other challenge may be that it's hard to capture all the opinions of the full voting population in a small sample. Societies are usually diverse with citizens ranging in age, of different backgrounds and living in different parts of state/province. It may be very difficult to represent all these opinions in a sample of, say, a 1000 people. Even if one is able to sufficiently control for these factors, again, it would require both time and money. If the pollster is working under a certain budget, which most pollsters probably are, they may have to then reduce the sample siz which would decrease the level of confidence. That's why a design of experiments is crucial for polling, so as to acquire a subset of the data that is accurate and also sufficiently sized while being economical.

0.3 Question 12.2

To determine the value of 10 different yes/no features to the market value of a house (large yard, solar roof, etc.), a real estate agent plans to survey 50 potential buyers, showing a fictitious house with different combinations of features. To reduce the survey size, the agent wants to show just 16 fictitious houses. Use R's FrF2 function (in the FrF2 package) to find a fractional factorial design for this experiment: what set of features should each of the 16 fictitious houses have? Note: the output of FrF2 is "1" (include) or "-1" (don't include) for each feature.

This question is relatively simple. Since a total of 16 houses can be show, each with 10 different attributes, we use the FrF2 function with parameters 16, and 10.

```
library(FrF2)
kable(FrF2(16, 10))
```

A	B	C	D	E	F	G	H	J	K
-1	1	1	1	-1	-1	1	-1	1	-1
1	-1	1	-1	-1	1	-1	-1	1	1
-1	-1	-1	-1	1	1	1	1	-1	1
1	-1	-1	-1	-1	-1	1	-1	-1	-1
1	1	-1	-1	1	-1	-1	-1	1	1
-1	-1	1	1	1	-1	-1	-1	-1	1
-1	1	-1	1	-1	1	-1	-1	-1	1
1	-1	-1	1	-1	-1	1	1	1	1
-1	-1	-1	1	1	1	1	-1	1	-1
1	-1	1	1	-1	1	-1	1	-1	-1
1	1	1	1	1	1	1	1	1	1
1	1	1	-1	1	1	1	-1	-1	-1
-1	1	-1	-1	-1	1	-1	1	1	-1
1	1	-1	1	1	-1	-1	1	-1	-1
-1	1	1	-1	-1	-1	1	1	-1	1
-1	-1	1	-1	1	-1	-1	1	1	-1

0.4 Question 13.1

For each of the following distributions, give an example of data that you would expect to follow this distribution (besides the examples already discussed in class).

a. Binomial
b. Geometric
c. Poisson
d. Exponential
e. Weibull

0.4.1 Question 13.1.a

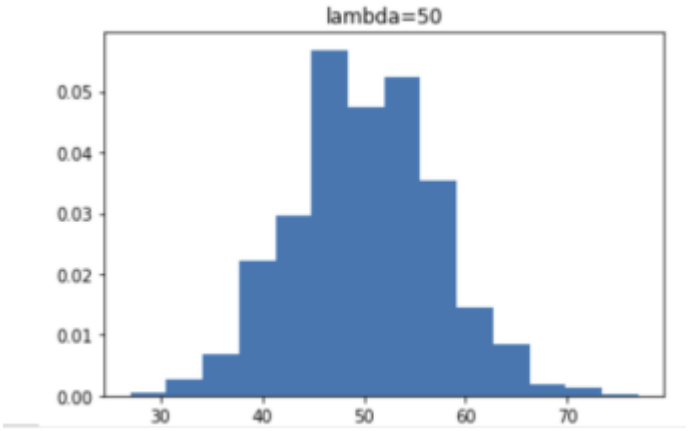
A binomial distribution could be appropriate in trying to predict the probability of you winning at roulette in a casino. If you make the same play every time like betting on red (the probability of winning is the same with each subsequent turn) and given that each event is independent (i.e. a previous play does not affect present or future play) then you can find the probability of breaking even or making more if you make x number of plays.

0.4.2 Question 13.1.b

If you are playing monopoly and trying to see how long you can go before rolling double sixes, you can use the geometric distribution. Since each roll is independent, and the probability of rolling double sixes stay the same, you can find the probability of going x number of moves without rolling double sixes.

0.4.3 Question 13.1.c

A poisson distribution might be used to model pizza delivery times. For example, if the average time is around 50 minutes, then our distribution would be centered around that. As we can see, there is a low probability of the pizza being delivered in 30 minutes as well as a low chance it will delivered after 70 minutes. Intuitively this seems quite plausible. Below is a histogram of delivery times with lambda equaling 50.



0.4.4 Question 13.1.d

Following on our previous example, let's say that I ordered pizza around 6 pm and I am pretty hungry. I want to find out what the probability of getting my order by 7pm. I can use an exponential distribution to model wait times and calculate such a probability. It turns out that there is about 70 percent chance that my pizza will get here by 7pm if the pizza delivery times do indeed follow a poission distribution.

It's important to note that the exponential distribution is memoryless, meaning that the failure rate is constant. So for example, if I wait 40 minutes (6:40pm) and am getting kind of nervous that my pizza hasn't gotten here, then the probability of getting my pizza at 7:40 is still 70 percent.

0.4.5 Question 13.1.e

A weibull distribution may be appropriate in a factory setting. If we were to map the failure rates of a 500 machines in a local factory, it may follow a weibull distribution since the longer the machine has been used, the more likely it is to fail down the line. Poission is not appropriate in this context since it is memoryless (not taking into account a changing failure rate).