Homework 8

2025-10-16

Question 11.1

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

A. Stepwise regression
B. Lasso
C. 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.

Notes on R:

- For the elastic net model, what we called λ in the videos, glmnet calls `alpha`; you can get a range of results by varying alpha from 1 (lasso) to 0 (ridge regression) [and, of course, other values of alpha in between].
- In a function call like `glmnet(x,y,family="mgaussian",alpha=1)` the predictors x need to be in R's matrix format, rather than data frame format. You can convert a data frame to a matrix using as.matrix – for example, `x <- as.matrix(data[,1:n-1])` Rather than specifying a value of T, glmnet returns models for a variety of values of T.

Boilerplate

First we load the necessary libraries:

```
library(printr)     # pretty print for Rmd
library(lubridate)  # dates
library(ggplot2)    # plots
library(dplyr)      # dataframes
library(tidyr)
library(tidyverse)
library(recipes)
library(caret)
library(tree)
library(randomForest)
library(stats)
library(MASS)      # step models
library(glmnet)
library(Metrics)

# set seed for reproducibility
set.seed(42)
```

Then let's load the data and add the same features I have used in all these models. Since we need to scale the data I am going to do that here as well.

```
df <- read.table("./data 10.1/uscrime.txt", header =TRUE)

crime_stats <- df %>% mutate(
  InvProb = 1 / Prob,
  LF2 = LF^2,
  NW2 = NW^2
)

predictor_cols <- setdiff(names(crime_stats), "Crime")

scaler <- preProcess(crime_stats[, predictor_cols],method = c("center", "scale"))
crime_scaled <- cbind(
  predict(scaler, crime_stats[, predictor_cols]),
  Crime = crime_stats$Crime
)

crime_x <- as.matrix(crime_scaled[,predictor_cols])
crime_y <- as.matrix(crime_scaled[,c("Crime")])
```

Note: As we know, our data set is extremely limited, therefore I will do 10-fold cross validation on the final model, but am not going to be creating a final test set. It was recommended in office hours to use a consistent cross validation folds.

```
folds <- 10
datapoints <- nrow(crime_x)
folds_for_glmnet <- sample(rep(1:folds, length.out = datapoints))
```

`caret` of course cannot use the same format for cross-validation.

```
folds_for_caret <- lapply(1:folds, function(i) which(folds_for_glmnet != i))

train_control <- trainControl(
  method = "cv",
  number = folds,
  index = folds_for_caret,
  savePredictions = "final"
)
```

I'm also going to compare our standard point just out of curiosity.

```
crime_test <- data.frame(
  M    = 14.0, So   = 0    , Ed   = 10.0, Po1    = 12.0,
  Po2  = 15.5, LF   = 0.640, M.F  = 94.0, Pop    = 150,
  NW   = 1.1 , U1   = 0.120, U2   = 3.6 , Wealth = 3200,
  Ineq = 20.1, Prob = 0.04 , Time = 39.0
) %>% mutate(
  InvProb = 1 / Prob,
  LF2 = LF^2,
  NW2 = NW^2
)

crime_test_scaled <- predict(scaler, crime_test)
```

11.1.A - Stepwise regression

I actually used stepwise regression on 8.2 (linear regression) as (essentially) a GLM with an log linking function (`log(Crime) ~ .`). However, I could not figure out how to achieve this in Parts B and C. Therefore I will be fitting against an untransformed crime rate for better comparison. I have included the GLM stepwise regression in Appendix 11.1.A - GLM Stepwise.

```
constant_model <- lm(
  Crime ~ 1,
  data = crime_scaled
)

full_model <- glm(
  Crime ~ .,
  data = crime_scaled
)

# pages of output
invisible(
  capture.output(
    step_model <- stepAIC(
      constant_model,
      scope = list(lower = constant_model, upper = full_model),
      direction = "both"
    )
  )
)

summary(step_model)
```

```
##
## Call:
## lm(formula = Crime ~ Po1 + Ineq + Ed + InvProb + Wealth + M +
##     U2, data = crime_scaled)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -396.30  -93.78   -7.41  110.10  445.13
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   905.09      28.23  32.057  < 2e-16 ***
## Po1           282.42      49.64   5.690 1.40e-06 ***
## Ineq          384.11      70.36   5.460 2.91e-06 ***
## Ed            231.94      51.04   4.544 5.21e-05 ***
## InvProb        93.21      34.47   2.704  0.01009 *
## Wealth        151.46      84.21   1.799  0.07983 .
## M             115.39      42.09   2.741  0.00919 **
## U2             63.38      33.63   1.884  0.06697 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 193.6 on 39 degrees of freedom
## Multiple R-squared:  0.7877, Adjusted R-squared:  0.7495
## F-statistic: 20.67 on 7 and 39 DF,  p-value: 2.684e-11
```

So we have 7 factors that are significant. Let's look at the cross-validated metrics.

```
cross_val_steps <- train(
  formula(step_model),
  data = crime_scaled,
  method = "lm",
  trControl = train_control
)

cross_val_steps
```

```
## Linear Regression
##
## 47 samples
##  7 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 42, 42, 42, 42, 42, 42, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   218.4621  0.7862543  180.7897
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

Now, let's predict our standard point.

```
predict(
  step_model,
  crime_test_scaled
)
```

```
##        1
## 880.1001
```

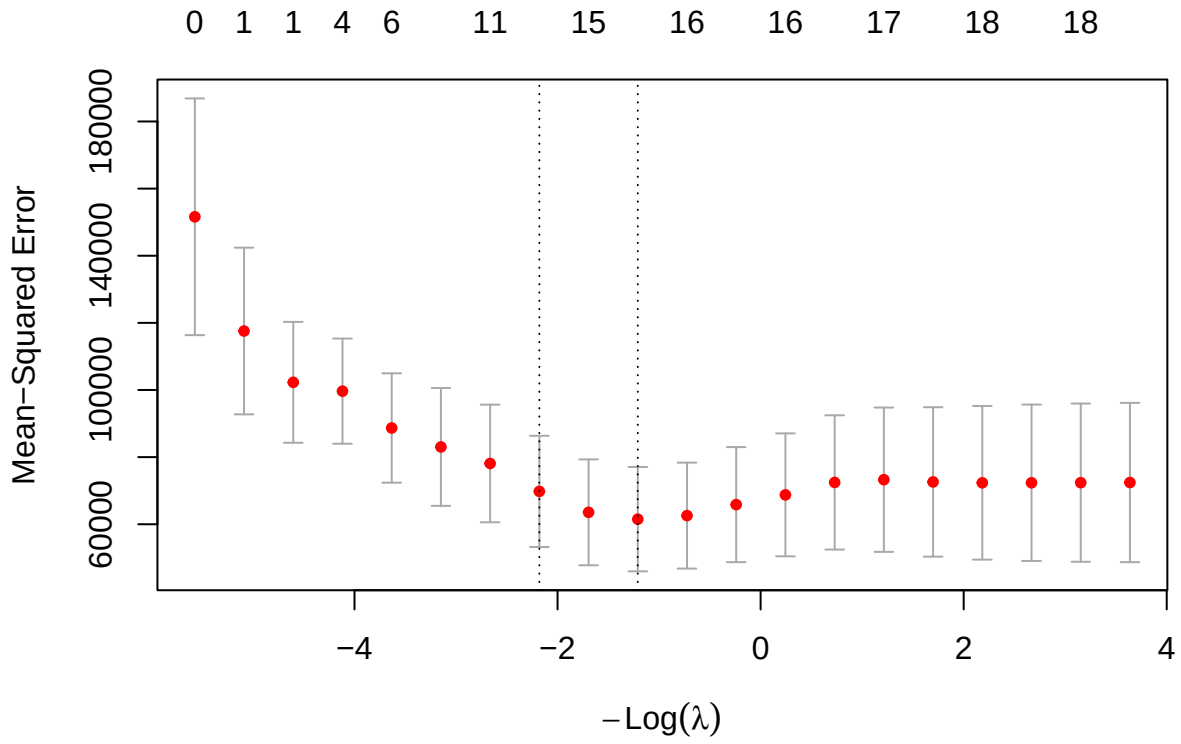
Let's go on to Lasso.

11.1.B - Lasso

For Lasso and Elastic net we were told to use glmnet. Prof. Sobel said in lectures that this is a better global optimization strategy; however, I am finding the function more cumbersome to use. Let's choose our parameters.

```
lasso <- cv.glmnet(
  crime_x,
  crime_y,
  alpha=1,
  nlambda=20,
  foldid = folds_for_glmnet
)

plot(lasso)
```



As we can see in the plot there are two choices of λ that the function highlights as possible "best": the minimum and the 1 standard error which allows a slightly worse model on the training set that is more parsimonious in the hopes that it will generalize better. However, given our splits the 1 standard error model is actually higher error and more terms; we will not be choosing that. So let's extract the best model.

```
co <- coef(lasso, s = "lambda.min")
vars_nonzero <- rownames(co)[as.vector(co != 0)]
vars_nonzero <- vars_nonzero[vars_nonzero != "(Intercept)"]

co
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
##              lambda.min
## (Intercept)  905.085106
## M             74.559265
## So             .
## Ed           204.860148
## Po1          260.120971
## Po2            .
## LF            23.175196
## M.F           29.119347
## Pop          -33.727684
## NW           353.168360
## U1             .
## U2            57.555000
## Wealth        31.914298
## Ineq         246.903349
## Prob         -37.588291
## Time          -8.644876
## InvProb      100.204898
## LF2            .
## NW2         -298.099721
```

Now I'm going to run the train control to get consistent metrics.

```
train(
  as.formula(paste("Crime ~ ", paste(vars_nonzero, collapse = " + "))),
  data = crime_scaled,
  method = "lm",
  trControl = train_control
)
```

```
## Linear Regression
##
## 47 samples
## 14 predictors
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 42, 42, 42, 42, 42, 42, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   241.0518  0.7536623  195.8812
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

And let's predict our test point

```
predict(lasso, crime_test_scaled, s = "lambda.min")
```

lambda.min
842.5082

11.1.C - Elastic Net

Now, let's explore a range of α in Elastic Net. We are excluding $\alpha = 1$ because that was 11.1.B - Lasso. As we go let's record metrics for λ_{min} and λ_{1se}.

```
elastic <- list()
elastic_metrics <- list()

for ( i in seq(10) )
{
  alpha <- (i-1)/10

  elastic_iter <- cv.glmnet(
    crime_x,
    crime_y,
    alpha=alpha,
    nlambda=20,
    foldid = folds_for_glmnet
  )

  elastic[[i]] <- elastic_iter

  i_min <- which(elastic_iter$lambda == elastic_iter$lambda.min)
  i_1se <- which(elastic_iter$lambda == elastic_iter$lambda.1se)

  elastic_metrics[[i]] <- list(
    alpha = alpha,
    mse_min = elastic_iter$cvm[i_min],
    mse_1se = elastic_iter$cvm[i_1se],
    nzero_min = elastic_iter$nzero[i_min],
    nzero_1se = elastic_iter$nzero[i_1se],
    lambda_min = elastic_iter$lambda.min,
    lambda_1se = elastic_iter$lambda.1se
  )
}

elastic_metrics <- do.call(rbind, lapply(elastic_metrics, as.data.frame))

as_tibble(elastic_metrics)
```

alpha	mse_min	mse_1se	nzero_min	nzero_1se	lambda_min	lambda_1se
0.0	65684.89	76494.52	18	18	26.309540	182.902024
0.1	65350.27	77750.53	17	17	12.715217	88.395273
0.2	64633.46	80881.34	16	15	6.357609	71.767094
0.3	64111.84	79676.24	16	12	6.882225	47.844729
0.4	63765.13	79281.37	16	12	5.161668	35.883547
0.5	63470.40	79266.73	16	12	4.129335	28.706837
0.6	63214.40	72580.69	17	15	3.441112	14.732545
0.7	62979.58	72254.64	16	15	4.789370	12.627896
0.8	62369.51	71485.60	15	14	4.190698	11.049409
0.9	61825.58	70554.28	15	14	3.725065	9.821697

Just to choose something significantly different from the Lasso model, I am going to choose λ_{1se} $\alpha = 0.5$ (although I feel that the optimal choice would be λ_{min} $\alpha = 0.9$). Let's extract the model and get our cross validated metrics.

```
co <- coef(elastic[[6]], s = "lambda.1se")
vars_nonzero <- rownames(co)[as.vector(co != 0)]
vars_nonzero <- vars_nonzero[vars_nonzero != "(Intercept)"]

co
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
##             lambda.1se
## (Intercept)  905.08511
## M             47.20201
## So            35.89616
## Ed            89.31229
## Po1          190.10441
## Po2           88.69057
## LF            14.42169
## M.F           60.54645
## Pop            .
## NW            22.38815
## U1             .
## U2            24.72108
## Wealth         .
## Ineq         143.65867
## Prob         -48.84283
## Time           .
## InvProb       54.72446
## LF2            .
## NW2            .
```

This is much more parsimonious than our Lasso model! Now I'm going to run the train control to get consistent metrics.

```
train(
  as.formula(paste("Crime ~ ", paste(vars_nonzero, collapse = " + "))),
  data = crime_scaled,
  method = "lm",
  trControl = train_control
)
```

```
## Linear Regression
##
## 47 samples
## 12 predictors
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 42, 42, 42, 42, 42, 42, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   264.0101  0.6674482  216.2902
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

Now let's make our final prediction.

```
predict(elastic[[6]], crime_test_scaled, s = "lambda.1se")
```

lambda.1se
1259.211

Evaluation and comparison to previous homeworks

Comparing all the results observed so far. The first thing that is readily apparent is that the model is inherently non-linear because our exponential, and random forest methods perform significantly better than the purely linear models. Additionally, the models do not appear to generalize significantly better.

Question	Method	R^2	MAE	RMSE	Prediction
8.2	GLM Step AIC Linear Regression	0.901	117.0	139.3	490.5
9.1	GLM Step AIC PCA Linear Regression	0.900	86.46	121.2	614.4
10.1.A	Regression Tree	0.707	153.6	207.0	724.6
10.1.B	Random Forest	0.944	82.58	121.9	1186.4
11.1.A	Step AIC Linear Regression	0.786	180.7897	218.5	880.1
11.1.B	Lasso	0.754	195.8812	241.1	842.5
11.1.C	Elastic Net	0.667	216.2902	264.0	1259.2

Appendix 11.1.A - GLM Stepwise

As stated in 11.1.A - Stepwise regression. I had already been using Stepwise regression in the previous homeworks and I was achieving much better results because I was using the logarithmic linking function. I wanted to demonstrate that.

```
constant_model <- glm(
  Crime ~ 1,
  data = crime_scaled,
  family = gaussian(link = "log")
)

full_model <- glm(
  Crime ~ .,
  data = crime_scaled,
  family = gaussian(link = "log")
)

# pages of output
invisible(
  capture.output(
    step_model <- stepAIC(
      constant_model,
      scope = list(lower = constant_model, upper = full_model),
      direction = "both"
    )
  )
)

summary(step_model)
```

```
##
## Call:
## glm(formula = Crime ~ Po1 + Ineq + Prob + Ed + InvProb + Wealth +
##     Pop + NW + NW2 + LF + LF2 + U2, family = gaussian(link = "log"),
```

```
##     data = crime_scaled)
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.72589    0.02182 308.256  < 2e-16 ***
## Po1          0.23276    0.03448   6.751 9.27e-08 ***
## Ineq         0.34971    0.05718   6.116 6.10e-07 ***
## Prob        -0.05905    0.03206  -1.842  0.07421 .
## Ed           0.19761    0.03812   5.183 9.94e-06 ***
## InvProb      0.13173    0.02425   5.433 4.70e-06 ***
## Wealth       0.11704    0.05654   2.070  0.04613 *
## Pop         -0.10137    0.01873  -5.411 5.02e-06 ***
## NW           0.59612    0.09886   6.030 7.88e-07 ***
## NW2         -0.45375    0.09538  -4.757 3.54e-05 ***
## LF           2.09403    0.69530   3.012  0.00488 **
## LF2         -2.01236    0.68868  -2.922  0.00614 **
## U2           0.05345    0.02190   2.441  0.02003 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 12595.5)
##
##     Null deviance: 6880928  on 46  degrees of freedom
## Residual deviance:  428246  on 34  degrees of freedom
## AIC: 589.89
##
## Number of Fisher Scoring iterations: 6
```

And here is the results

```
cross_val_steps <- train(
  formula(step_model),
  data = crime_scaled,
  method = "glm",
  family = gaussian(link = "log"),
  trControl = train_control
)

cross_val_steps
```

```
## Generalized Linear Model
##
## 47 samples
## 12 predictors
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 42, 42, 42, 42, 42, 42, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   132.6858  0.8495301  116.0333
```