Homework Week 8

Chris Messer

2022-10-17

Question 11.1

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

1. Stepwise regression

2. Lasso

3. 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.

See analysis below.

Data Prep

First lets bring in the data and look at the structure.

```
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
```

```
data <- read.csv('uscrime.txt', sep = '\t')
set.seed(1)
str(data)
```

```
## 'data.frame':    47 obs. of  16 variables:
##  $ M     : num  15.1 14.3 14.2 13.6 14.1 12.1 12.7 13.1 15.7 14 ...
##  $ So    : int  1 0 1 0 0 0 1 1 1 0 ...
##  $ Ed    : num  9.1 11.3 8.9 12.1 12.1 11 11.1 10.9 9 11.8 ...
##  $ Po1   : num  5.8 10.3 4.5 14.9 10.9 11.8 8.2 11.5 6.5 7.1 ...
##  $ Po2   : num  5.6 9.5 4.4 14.1 10.1 11.5 7.9 10.9 6.2 6.8 ...
##  $ LF    : num  0.51 0.583 0.533 0.577 0.591 0.547 0.519 0.542 0.553 0.632 ...
##  $ M.F   : num  95 101.2 96.9 99.4 98.5 ...
##  $ Pop   : int  33 13 18 157 18 25 4 50 39 7 ...
##  $ NW    : num  30.1 10.2 21.9 8 3 4.4 13.9 17.9 28.6 1.5 ...
##  $ U1    : num  0.108 0.096 0.094 0.102 0.091 0.084 0.097 0.079 0.081 0.1 ...
##  $ U2    : num  4.1 3.6 3.3 3.9 2 2.9 3.8 3.5 2.8 2.4 ...
##  $ Wealth: int  3940 5570 3180 6730 5780 6890 6200 4720 4210 5260 ...
##  $ Ineq  : num  26.1 19.4 25 16.7 17.4 12.6 16.8 20.6 23.9 17.4 ...
##  $ Prob  : num  0.0846 0.0296 0.0834 0.0158 0.0414 ...
##  $ Time  : num  26.2 25.3 24.3 29.9 21.3 ...
##  $ Crime : int  791 1635 578 1969 1234 682 963 1555 856 705 ...
```

We should first consider if we want to scale. Ultimately, for stepwise regression we probably don't need to scale. With linear regression, scaling does not impact the model quality, but it is generally a good idea to scale coefficient because then we can compare coefficients against eachother to see which coefficient is the impacting the y value the most.

Scaling IS however needed for lasso and elastic net, so lets just go ahead and scale our data now.

We can see here that feature So is probably a binomial factor of 0 or 1, so lets first remove that before scaling, otherwise we would end up with values between 0 and 1, which would not make sense. We will also remove the response before scaling.

```
#scale
data.scaled = as.data.frame(scale(data[,-c(2,16)]))
data.scaled <- cbind(data[,2],data.scaled,data[,16]) # Add column 2 back in
colnames(data.scaled)[1] <- "So"
colnames(data.scaled)[16] <- "Crime"
```

Step-wise regression

Step-wise regression is a combination of forward selection and backwards elimination. What that means is we take all of our features, determine their colinearity with the desired response (crime), and then rank them from highest to lowest. If the resulting pvalue for each feature is greater than a certain threshold (we will call that p.enter) then we consider it a candidate for our final model. We will call these factors.best in the below code

```
factors.best <- c()
for (i in seq(15)){
  lm <- lm(Crime~., data.scaled[,c(i,16)])
  lm.pv <- summary(lm)$coefficients[-1,4]
  lm.rs<- summary(lm)$r.squared
  factors.best$pvalue[i]<- lm.pv
  factors.best$rs[i] <- lm.rs
}
factors.best <- data.frame(factors.best)
rownames(factors.best) <- colnames(data[,-16])
factors.best<-factors.best[order(factors.best$pvalue),]
factors.best
```

```
##              pvalue          rs
## Po1    9.338016e-08 0.472799888
## Po2    3.114182e-07 0.444507747
## Wealth 1.901669e-03 0.194763297
## Prob   2.730202e-03 0.182689727
## Pop    2.035060e-02 0.113888742
## Ed     2.687817e-02 0.104222353
## M.F    1.487937e-01 0.045759311
## LF     2.035810e-01 0.035670497
## Ineq   2.285772e-01 0.032049496
## U2     2.331077e-01 0.031442612
## Time   3.146771e-01 0.022459836
## M      5.445860e-01 0.008215058
## So     5.497891e-01 0.008005310
## U1     7.361504e-01 0.002548020
## NW     8.277983e-01 0.001062684
```

Now that we have ranked all of the predicting features, we will exclude any with a p value less than p.enter (.15)

```
p.enter <- .15
p.exit <- .15
factors.best.enter <- factors.best[factors.best$pvalue <= p.enter,]
```

Now we will consider each of the above features for our final model. We will start with our best feature (Po1), and then add the next best feature (Po2) and run a regressino model with both features predicting the response (crime). If either of the features in that model result in a pvalue greater than a threshold (we will call this p.exit, and set it to .15) then we remove that feature from our model forever. This process of removing features and writing a new formula can be code extensive, so lets just write a function that takes in a linear model, and a threshold, and returns a new formula excluding any coefficients over the threshold.

```
write_formula <- function(lm_model, pvalue_filter) {
          coeff <- data.frame(pvalue = (summary(lm_model)$coefficients)[-1,4])
          formula <- (paste("Crime ~ ", paste(rownames(filter(coeff, pvalue < pvalue_filter)), collapse = " + "))
)
  return(formula)
}
```

Great! Now that that is written, lets start our process of adding in one feature at a time, and then removing any features that have a p value over p.exit.

```
formula <- paste("Crime ~ ", rownames(factors.best.enter[1,]))

for (i in seq(nrow(factors.best.enter[-1,]))){

  # create a formula for a linear model with the next best factor so far
  formula <- paste(c(formula, rownames(factors.best.enter[-1,][i,])), collapse = " + ")

  #create a new linear model with the resulting formula
  lm <- lm(as.formula(formula), data.scaled)

  #using our function from above, strip out any coefficients with pvalue > p.exit (.15)

  formula<- write_formula(lm, p.exit)

  #start the loop again, and add the next highest (i) pvalue coefficient

}

#the resulting model using our final formula
m1 <- lm(as.formula(formula),data.scaled)
summary(m1)
```

```
##
## Call:
## lm(formula = as.formula(formula), data = data.scaled)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -586.94 -140.39   14.46  142.68  513.22
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   905.09      37.66  24.032  < 2e-16 ***
## Po1           365.48      62.85   5.816 7.31e-07 ***
## Wealth       -184.74      67.70  -2.729  0.00924 **
## Prob          -89.90      45.94  -1.957  0.05703 .
## M.F            99.00      39.37   2.515  0.01582 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 258.2 on 42 degrees of freedom
## Multiple R-squared:  0.5931, Adjusted R-squared:  0.5543
## F-statistic:  15.3 on 4 and 42 DF,  p-value: 8.481e-08
```

And that is our final model.

y = 365.48 * Po1 -184.74 * Wealth -89.90 * Prob + 99 * M.F + 905. Now we should cross validate that model to see how accurate it is. Again, this can be a cumbersome process, so lets write a function that takes in a linear model and a dataset, and returns the cross validated r squared.

```
library('DAAG')
cross_validate <- function(lm, dataset){
  cv <- cv.lm(dataset, lm, m=5, seed=10, printit=F, plotit =F)

  #the model does not output r^2, so we will have to calculate ourselves. The formula is 1-(Sum of Squared Errors
/sumofsquared differences)
  SSres <- attr(cv,"ms")*nrow(dataset)
  SStotal <- sum((dataset$Crime - mean(dataset$Crime))^2)

  rs <- 1- SSres/SStotal
  return(rs)
  }
```

Now lets put it in action.

```
cross_validate(m1, data.scaled)
```

```
## [1] 0.5001406
```

It appears our model has a .5 R^2 value after cross validation

LASSO Regression

LASSO regression is different from regular regression in that instead of minimizing the sum of squares, it minimizes the sum of squares + lambda * |x - y|.

This introduces a penalty factor, lambda, and ultimately helps prevent overfitting when we have few data points.

The glmnet package takes in the error parameter (called alpha instead of lambda). The way this package is written is to understand alpha = 1 to be lasso regression. Alpha < 1 performs elastic-net regression.

Elastic net regression is essentially a combination of lasso and ridge regression. So by using an alpha of less than 1, we are telling it to use an error penalty of alpha for the lasso regression part of the equation, and 1- alpha for the ridge regression part of the equation. So to perform just lasso regression, we use alpha = 1 so that the error penalty for ridge regression is 0.

```
library('glmnet')
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-4
```

```
set.seed(1)
#glmnet takes in data matricies and not data frames, so we need to convert
x.matrix <- data.matrix(data.scaled[,-16])
y.matrix <- data.matrix(data.scaled$Crime)

#build the model
lasso=cv.glmnet(x=x.matrix,
                y=y.matrix,
                alpha=1,
                nfolds = 5,
                type.measure="mse",family="gaussian")


important_coefficients <- (coef(lasso, s=lasso$lambda.min))
important_coefficients
```

```
## 16 x 1 sparse Matrix of class "dgCMatrix"
##                     s1
## (Intercept) 889.648600
## So           45.344736
## M            77.635868
## Ed           98.182711
## Po1         311.196426
## Po2           .
## LF            2.887732
## M.F          46.955007
## Pop           .
## NW            2.654767
## U1            .
## U2           29.302470
## Wealth        .
## Ineq        164.140537
## Prob        -77.051224
## Time          .
```

Now we can see which coefficients lasso regression determined to be important, and which ones it minimized down to 0. Now lets use those coefficients to build a new linear model, and then cross validate it to get an r^2.

```
coeff.list <- important_coefficients@i[-1]

lasso_coeffs <- data.scaled[,c(coeff.list,16)]
m2 <- lm(Crime~., lasso_coeffs)

lasso.cv <- cross_validate(m2, lasso_coeffs)
summary(m2)
```

```
##
## Call:
## lm(formula = Crime ~ ., data = lasso_coeffs)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -410.55 -121.42    5.76  110.54  550.24
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  863.284     53.510  16.133  < 2e-16 ***
## So           122.792    129.896   0.945  0.35080
## M            113.614     49.962   2.274  0.02903 *
## Ed           188.755     64.750   2.915  0.00608 **
## Po1          333.470     49.353   6.757 6.86e-08 ***
## LF            25.162     49.588   0.507  0.61496
## M.F           31.513     43.179   0.730  0.47021
## NW            -2.883     59.595  -0.048  0.96169
## U2            72.881     41.795   1.744  0.08973 .
## Ineq         229.121     68.845   3.328  0.00203 **
## Prob         -99.145     40.243  -2.464  0.01867 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 206.6 on 36 degrees of freedom
## Multiple R-squared:  0.7767, Adjusted R-squared:  0.7147
## F-statistic: 12.53 on 10 and 36 DF,  p-value: 5.374e-09
```

```
lasso.cv
```

```
## [1] 0.4828613
```

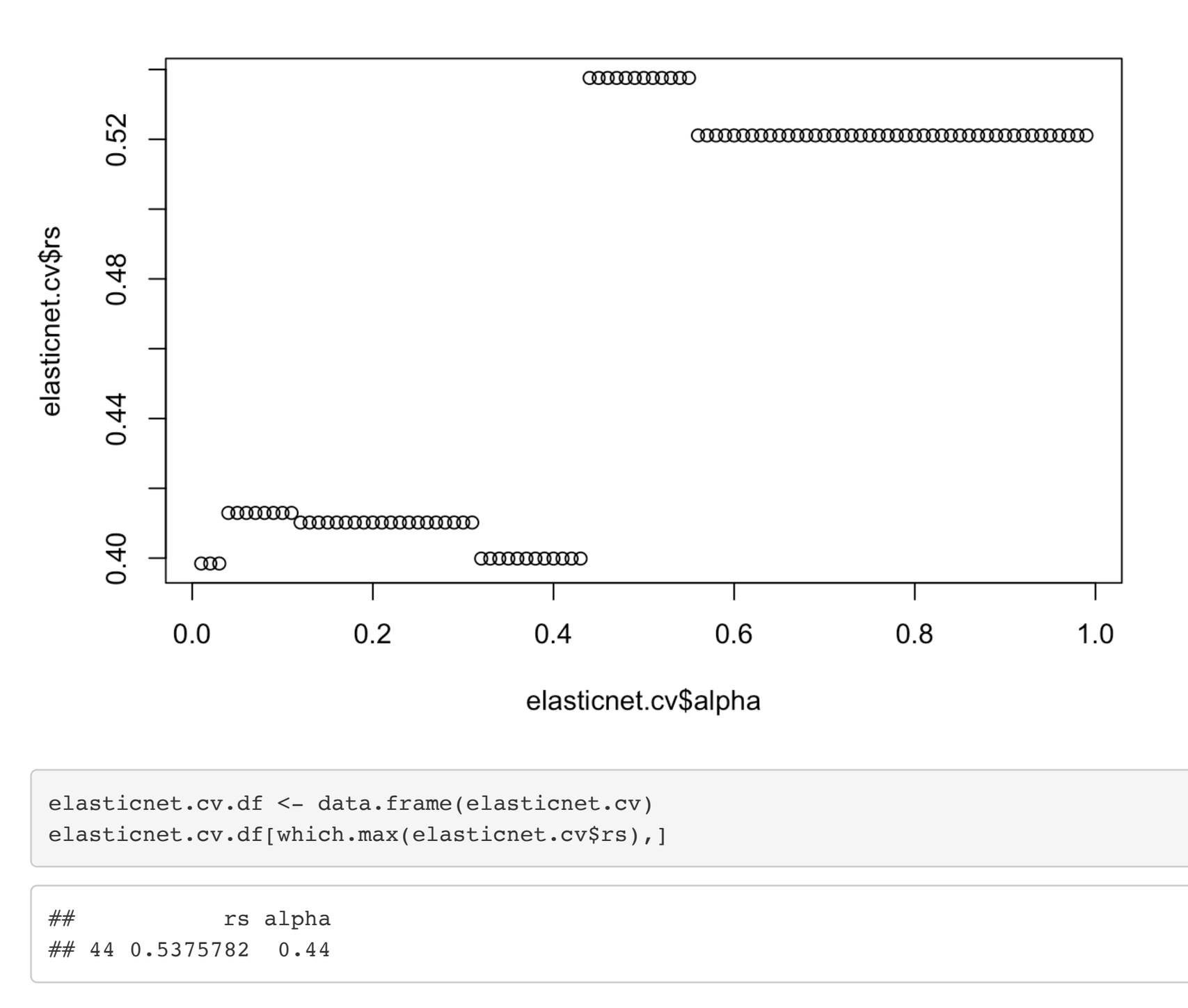
And the resulting r^2 is .52, and uses 12 factors. Some of which have very high p values.

Elastic-Net Regression

Finally, we will perform elastic net regression. As mentioned above, it is a combination of lasso and ridge regression. It is best used when we have many attributes, and we are unsure if there are many unimportant features or not.

Here we will loop over many values of alpha to determine which gives us the highest r^2 value.

```
elasticnet.cv <- c()
for (a in seq(99)){
    lasso=cv.glmnet(x=x.matrix,
                    y=y.matrix,
                    alpha=a/100,
                    nfolds = 5,
                    type.measure="mse",family="gaussian")

    important_coefficients <- (coef(lasso, s=lasso$lambda.min))
    coeff.list <- important_coefficients@i[-1]

    lasso_coeffs <- data.scaled[,c(coeff.list,16)]
    m2 <- lm(Crime~., lasso_coeffs)

    elasticnet.cv$rs[a] <- cross_validate(m2, lasso_coeffs)
    elasticnet.cv$alpha[a] <- a/100

}


plot(x = elasticnet.cv$alpha,
     y = elasticnet.cv$rs)
```




```
elasticnet.cv.df <- data.frame(elasticnet.cv)
elasticnet.cv.df[which.max(elasticnet.cv$rs),]
```

```
##           rs alpha
## 44 0.5375782  0.44
```

Now we need to rerun the model to see what coefficients were used.

```
lasso=cv.glmnet(x=x.matrix,
                    y=y.matrix,
                    alpha=44/100,
                    nfolds = 5,
                    type.measure="mse",family="gaussian")

    important_coefficients <- (coef(lasso, s=lasso$lambda.min))
    coeff.list <- important_coefficients@i[-1]

    lasso_coeffs <- data.scaled[,c(coeff.list,16)]
    m3 <- lm(Crime~., lasso_coeffs)

summary(m3)$coefficients
```

```
##               Estimate Std. Error    t value     Pr(>|t|)
## (Intercept)  893.37574   52.51499 17.0118239 6.735214e-18
## So            34.39626  127.11999  0.2705810 7.883985e-01
## M            109.87417   49.82415  2.2052392 3.451406e-02
## Ed           202.40772   63.99746  3.1627463 3.345642e-03
## Po1          501.62901  287.30141  1.7460026 9.011530e-02
## Po2         -215.07732  288.65457 -0.7451028 4.614837e-01
## M.F           43.44647   48.99428  0.8867661 3.816220e-01
## Pop          -36.20655   46.10103 -0.7853740 4.378370e-01
## NW            24.90678   58.60583  0.4249881 6.736038e-01
## U1           -86.62493   66.24289 -1.3076864 2.000170e-01
## U2           136.97183   67.40791  2.0319844 5.026947e-02
## Wealth        82.03116   96.17291  0.8529549 3.998330e-01
## Ineq         275.77460   86.79140  3.1774414 3.218840e-03
## Prob         -95.15974   41.52101 -2.2918453 2.842974e-02
```

As seen above, a penalty facotr of .44 gives the highest r^2 of .53, and uses 13 features.

Analysis

So which model is best? How does it compare to using all features?

```
m4 <- lm(Crime~., data.scaled)
summary(m4)
```

```
##
## Call:
## lm(formula = Crime ~ ., data = data.scaled)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -395.74  -98.09   -6.69  112.99  512.67
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  906.380     59.113  15.333 5.08e-16 ***
## So            -3.803    148.755  -0.026  0.97977
## M            110.382     52.424   2.106  0.04344 *
## Ed           210.678     69.458   3.033  0.00486 **
## Po1          572.995    315.347   1.817  0.07889 .
## Po2         -305.958    328.483  -0.931  0.35883
## LF           -26.826     59.394  -0.452  0.65465
## M.F           51.293     59.977   0.855  0.39900
## Pop          -27.906     49.095  -0.568  0.57385
## NW            43.234     66.642   0.649  0.52128
## U1          -105.056     75.906  -1.384  0.17624
## U2           141.714     69.536   2.038  0.05016 .
## Wealth        92.792    100.028   0.928  0.36075
## Ineq         281.954     90.630   3.111  0.00398 **
## Prob        -110.394     51.667  -2.137  0.04063 *
## Time         -24.655     50.780  -0.486  0.63071
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 209.1 on 31 degrees of freedom
## Multiple R-squared:  0.8031, Adjusted R-squared:  0.7078
## F-statistic: 8.429 on 15 and 31 DF,  p-value: 3.539e-07
```

Using all of the factors results in a very high r^2 value. Why? Overfitting! This is sometimes called the "kitchen sink method" and hardly ever is the best option.

Using stepwise regression, we got a model with 4 predicting features and r^2 of .5

Lasso gave us .52 r^2 and 12 features,

And EN gave us .53 r^2 and 13 features. So which is best?

In this case I would probably use the stepwise function even though it has the lowest r^2 because of how simple the model is. Using too many facotrs (especially when we have few data points) is not good practice, as it is a surefire way to overfit your model.