Non-Parametric Regression and Shrinkage Methods - Lab

Blake Pappas

December 17, 2023

Non-Parametric Regression

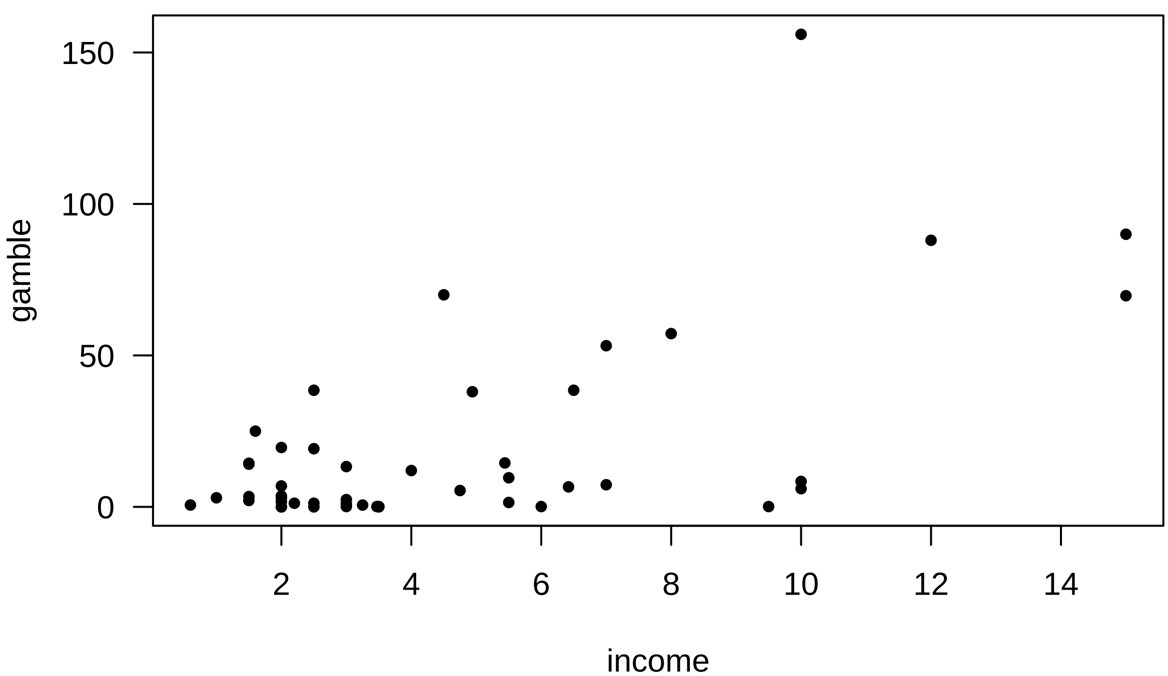
The dataset `teengamb` concerns a study of teenage gambling in Britain. Type 'r ?teengamb' to get more details about the the dataset. In this lab we will take the variables `gamble` as the response and `income` as the predictor.

Data Source:

1. Make a scatterplot to examine the relationship between the predictor `income` and the response `gamble`.

Code:

```
library(faraway)
data(teengamb)
with(teengamb, plot(gamble ~ income, pch = 16, cex = 0.8, las = 1))
```



2. Fit a curve to the data using regression spline with `df = 8`. Produce a plot for the fit and a 95% confidence band (using *RegSplinePred <- predict(RegSplineFit, data.frame(income = xg), interval = "confidence")*) for the fit. Is a linear fit plausible?

Code:

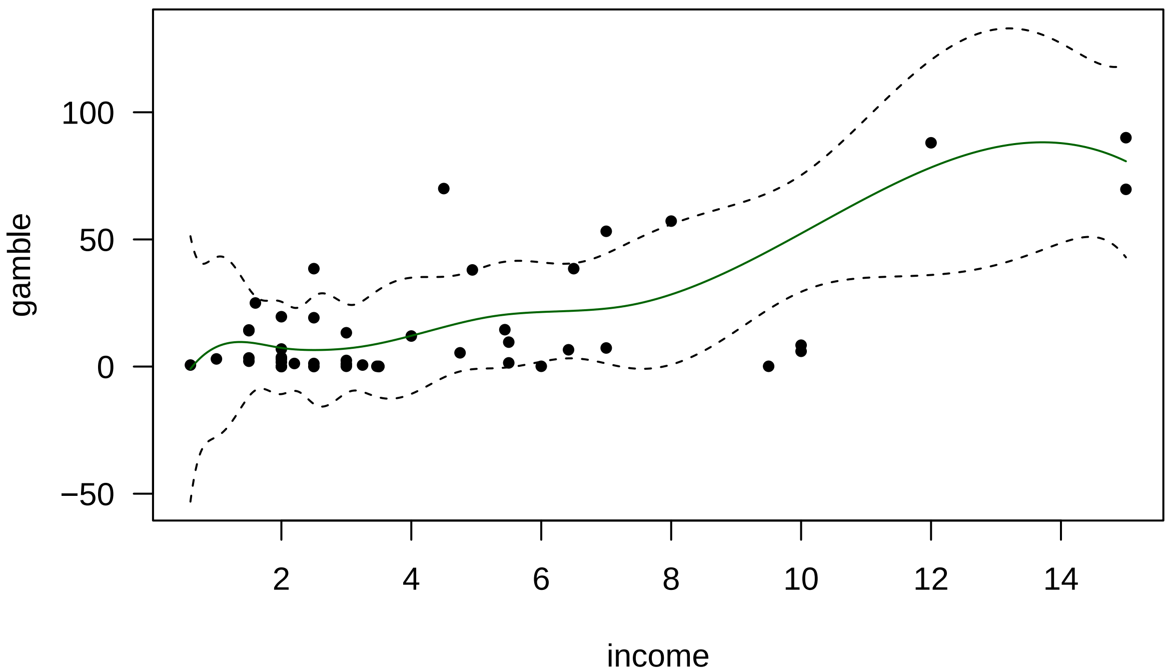
```
library(splines)
RegSplineFit <- lm(gamble ~ bs(income, df = 8), data = teengamb)
summary(RegSplineFit)
```

```
##
## Call:
## lm(formula = gamble ~ bs(income, df = 8), data = teengamb)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -46.324  -8.878  -5.565   7.737 103.676
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)           -0.9238    25.7802  -0.036   0.9716
## bs(income, df = 8)1  15.2822    53.9892   0.283   0.7787
## bs(income, df = 8)2   8.1681    34.4536   0.237   0.8139
## bs(income, df = 8)3   7.1312    32.4370   0.220   0.8272
## bs(income, df = 8)4   8.9985    34.2986   0.262   0.7945
## bs(income, df = 8)5  24.9733    33.7835   0.739   0.4643
## bs(income, df = 8)6  15.8732    53.4413   0.297   0.7681
## bs(income, df = 8)7 112.9416    64.9684   1.738   0.0902 .
## bs(income, df = 8)8  81.6489    31.8210   2.566   0.0144 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 26.47 on 38 degrees of freedom
## Multiple R-squared:  0.4174, Adjusted R-squared:  0.2948
## F-statistic: 3.404 on 8 and 38 DF,  p-value: 0.004829
```

```
(rg <- range(teengamb$income))
```

```
## [1]  0.6 15.0
```

```
xg <- seq(0.6, 15, 0.01)
RegSplinePred <- predict(RegSplineFit, data.frame(income = xg), interval = "confidence")

with(teengamb, plot(gamble ~ income, pch = 16, cex = 0.8, las = 1, ylim = range(RegSplinePred)))
lines(xg, RegSplinePred[, 1], col = "darkgreen")
lines(xg, RegSplinePred[, 2], lty = 2)
lines(xg, RegSplinePred[, 3], lty = 2)
```



Answer:

3. Fit a curve using either generalized additive models or smoothing splines.

Code:

GAM Fit

```
library(mgcv)
```

```
## Loading required package: nlme
```

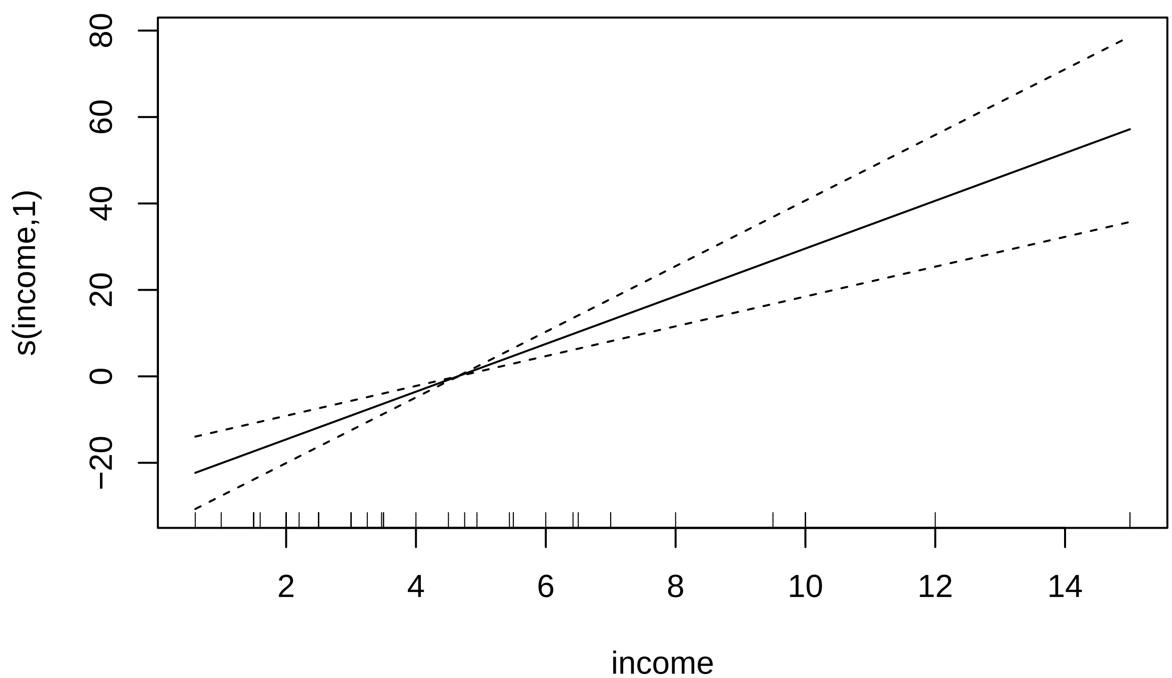
```
## This is mgcv 1.8-35. For overview type 'help("mgcv-package")'.
```

```
GAMFit <- gam(gamble ~ s(income), data = teengamb)
summary(GAMFit)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
```

```
## gamble ~ s(income)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   19.301      3.639   5.304 3.32e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##           edf Ref.df     F  p-value
## s(income)   1      1 28.41 3.52e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.373   Deviance explained = 38.7%
## GCV = 650.08  Scale est. = 622.41    n = 47
```

```r
plot(GAMFit)
```



Smoothing Spline Fit

```r
library(fields)
```

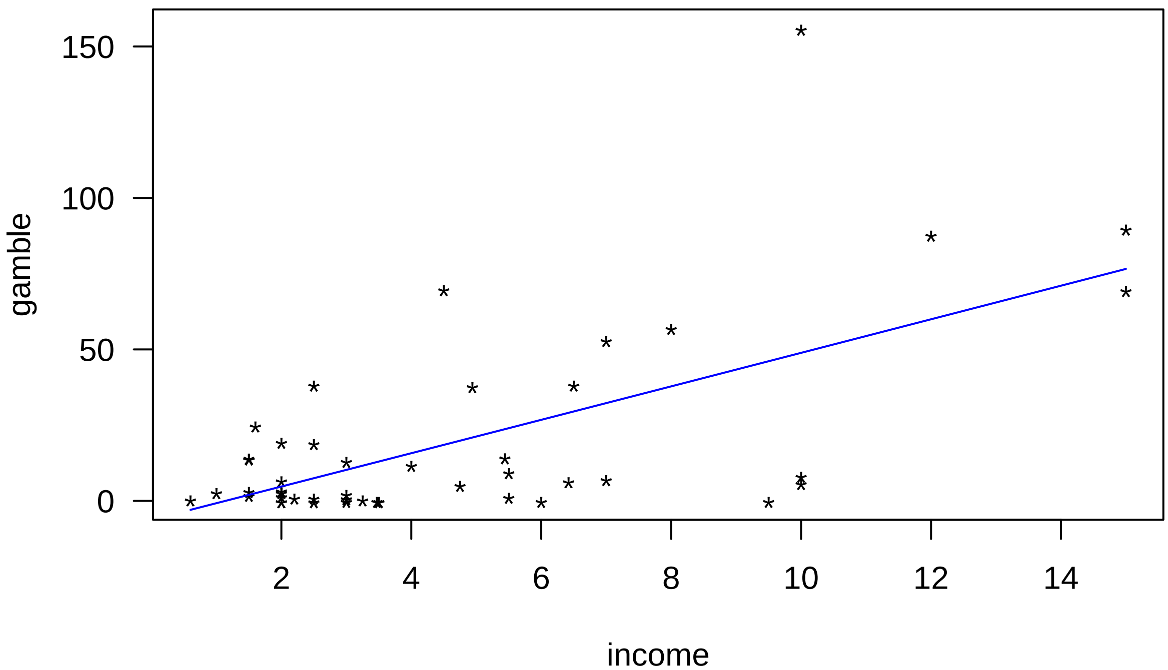
```
## Loading required package: spam
```

```
## Spam version 2.9-1 (2022-08-07) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.

##
## Attaching package: 'spam'

## The following objects are masked from 'package:base':
##
##     backsolve, forwardsolve

## Loading required package: viridis

## Loading required package: viridisLite

##
## Try help(fields) to get started.
```

```
SpFit <- with(teengamb, sreg(income, gamble))
```

```
## Methods at endpoints of grid search:
##            Warning Refine indexMIN leftEndpoint rightEndpoint       lambda
## GCV           TRUE  FALSE       80         TRUE         FALSE 1.060237e+03
## GCV.model     TRUE  FALSE        1        FALSE          TRUE 1.873756e-07
## GCV.one       TRUE  FALSE       80         TRUE         FALSE 1.060237e+03
## pure error    TRUE  FALSE        1        FALSE          TRUE 1.873756e-07
##                effdf
## GCV         2.010005
## GCV.model  25.739944
## GCV.one     2.010005
## pure error 25.739944
```

```
SpPred <- predict(SpFit, xg)

with(teengamb, plot(gamble ~ income, pch = "*", cex = 1, las = 1))
lines(xg, SpPred, col = "blue")
```



Ridge Regression and LASSO: Meat Spectrometry to Determine Fat Content

A Tecator Infratec Food and Feed Analyzer working in the wavelength range 850 - 1050 nm by the Near Infrared Transmission (NIT) principle was used to collect data on samples of finely chopped pure meat. 215 samples were measured. For each sample, the fat content was measured along with a 100 channel spectrum of absorbances. Since determining the fat content via analytical chemistry is time-consuming, we would like to build a model to predict the fat content of new samples using the 100 absorbances which can be measured more easily.

Data Source: H. H. Thodberg (1993) "Ace of Bayes: Application of Neural Networks With Pruning", report no. 1132E, Maglegaardvej 2, DK-4000 Roskilde, Danmark

Load the data and partition the data into *training set* (the first 150 observations) and *testing set* (the remaining 65 observations).

Code:

```
data(meatspec, package = "faraway")
train <- 1:150; test <- 151:215
trainmeat <- meatspec[train, ]
testmeat <- meatspec[test, ]
```

4. Fit a linear regression with all the 100 predictors to the training set. Compute the root mean square error (RMSE) for the testing set.

Code:

```r
lmFit <- lm(fat ~ ., data = trainmeat)

# Define a Function to Calculate RMSE
rmse <- function(pred, obs) sqrt(mean((pred - obs)^2))

# Computing RMSE for the Training Set
rmse(predict(lmFit), trainmeat$fat)
```

```
## [1] 0.5919489
```

```r
# Computing RMSE for the Testing Set
rmse(predict(lmFit, testmeat), testmeat$fat)
```

```
## [1] 3.522791
```

Answer:

The RMSE for the testing set is 3.5227911.

5. Fit a ridge regression (using cross-validation to select the "best" λ) and compute the RMSE for the training set.

Code:

```r
library(glmnet)
```

```
## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following object is masked from 'package:spam':
##
##     det

## Loaded glmnet 4.1-7
```

```r
X <- model.matrix(fat ~ ., data = meatspec)[, -1]
y <- meatspec$fat

grid <- 10^seq(10, -2, length = 100)
ridgeFit <- glmnet(X[train, ], y[train], alpha = 0, lambda = grid)

set.seed(1)

# Fit Ridge Regression Model on Training Data
cv.out <- cv.glmnet(X[train,], y[train], alpha = 0, thresh = 1e-12)

# Select Lambda That Minimizes Training MSE
(bestLambda = cv.out$lambda.min)
```

```
## [1] 0.7152024
```

```r
ridge.pred <- predict(ridgeFit, s = bestLambda, newx = X[test, ])
rmse(ridge.pred, y[test])
```

```
## [1] 5.053796
```

Answer:

The RMSE for the testing set is 5.0537959.

6. Fit a LASSO (again using cross-validation to select the "best" λ) and compute the RMSE for the training set.

Code:

```r
LASSOFit <- glmnet(X[train, ], y[train], alpha = 1, lambda = grid)

# Fit Ridge Regression Model on Training Data
cv.out <- cv.glmnet(X[train, ], y[train], alpha = 1)

# Select Lambda That Minimizes Training MSE
(bestLambda = cv.out$lambda.min)
```

```
## [1] 0.00881735
```

```r
LASSO.pred <- predict(LASSOFit, s = bestLambda, newx = X[test, ])
rmse(LASSO.pred, y[test])
```

```
## [1] 3.110983
```

Answer:

The RMSE for the testing set is 3.1109832.

7. Fit a LASSO with all the data points (using the best λ) and report the number of non-zero regression coefficients.

Code:

```r
out <- glmnet(X, y, alpha = 1, lambda = grid)

# Display Coefficients Using Lambda Chosen by CV
(lasso.coef <- predict(out, type = "coefficients", s = bestLambda))
```

```
## 101 x 1 sparse Matrix of class "dgCMatrix"
##                        s1
## (Intercept)  2.234885e+01
## V1           5.378785e+01
## V2           .
## V3           .
## V4           .
## V5           .
```

```
## V6            .
## V7            .
## V8            .
## V9            .
## V10           .
## V11           .
## V12           .
## V13          -8.071676e+00
## V14          -4.031423e+01
## V15          -1.627120e+00
## V16          -3.564300e+01
## V17          -2.560770e+01
## V18          -1.435523e+01
## V19          -4.764256e+00
## V20          -3.494832e+00
## V21          -2.287864e+00
## V22          -3.518089e+00
## V23          -7.456287e-01
## V24          -1.613391e-01
## V25           .
## V26           .
## V27           .
## V28           .
## V29           .
## V30           .
## V31           .
## V32           .
## V33           .
## V34           .
## V35           .
## V36           .
## V37           .
## V38           .
## V39           .
## V40           2.845545e+00
## V41           1.403102e+02
## V42           .
## V43           .
## V44           .
## V45           .
## V46           .
## V47           .
## V48           .
## V49           .
## V50          -1.018599e+01
## V51          -5.511367e+00
## V52          -4.667286e+01
## V53          -2.526771e+00
## V54           .
## V55           .
## V56           .
## V57           .
## V58           .
## V59           .
```

```
## V60          .
## V61          .
## V62          .
## V63          .
## V64          .
## V65          .
## V66          .
## V67          .
## V68          .
## V69          .
## V70          .
## V71          .
## V72          .
## V73         -8.095047e-04
## V74         -1.011876e-03
## V75         -6.571932e-04
## V76         -2.693999e-05
## V77          .
## V78          .
## V79          .
## V80          .
## V81          .
## V82          .
## V83          .
## V84          .
## V85          .
## V86          .
## V87          .
## V88          .
## V89          .
## V90          .
## V91          .
## V92          .
## V93          .
## V94          .
## V95          .
## V96          .
## V97          .
## V98          3.588762e+00
## V99          7.878137e-01
## V100         1.301520e+00
```

```
lasso.coef[lasso.coef != 0]
```

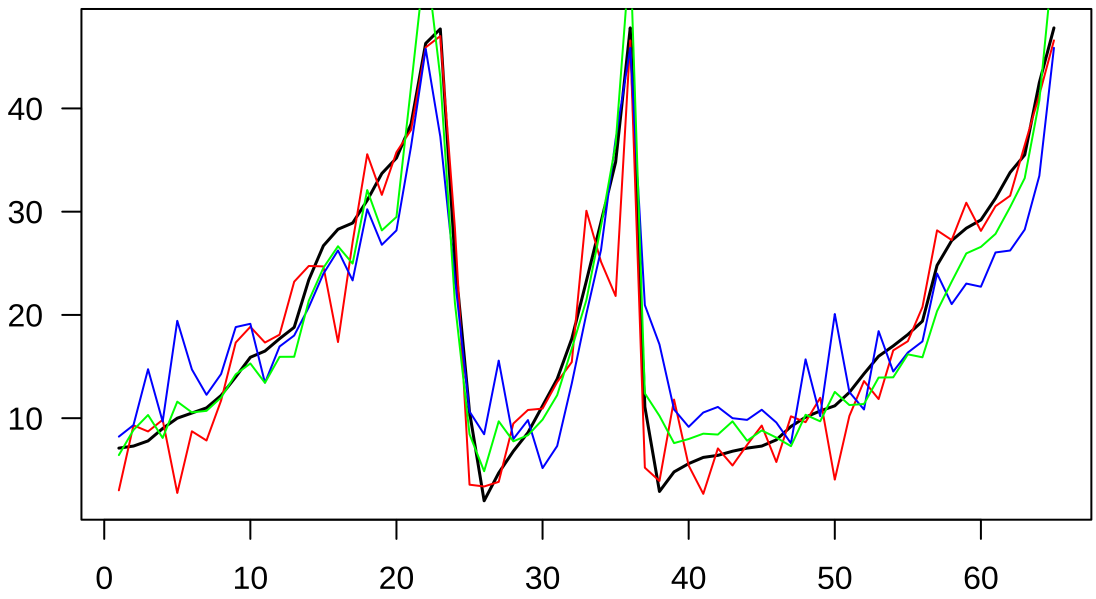
```
## <sparse>[ <logic> ]: .M.sub.i.logical() maybe inefficient
```

```
##  [1]  2.234885e+01  5.378785e+01 -8.071676e+00 -4.031423e+01 -1.627120e+00
##  [6] -3.564300e+01 -2.560770e+01 -1.435523e+01 -4.764256e+00 -3.494832e+00
## [11] -2.287864e+00 -3.518089e+00 -7.456287e-01 -1.613391e-01  2.845545e+00
## [16]  1.403102e+02 -1.018599e+01 -5.511367e+00 -4.667286e+01 -2.526771e+00
## [21] -8.095047e-04 -1.011876e-03 -6.571932e-04 -2.693999e-05  3.588762e+00
## [26]  7.878137e-01  1.301520e+00
```

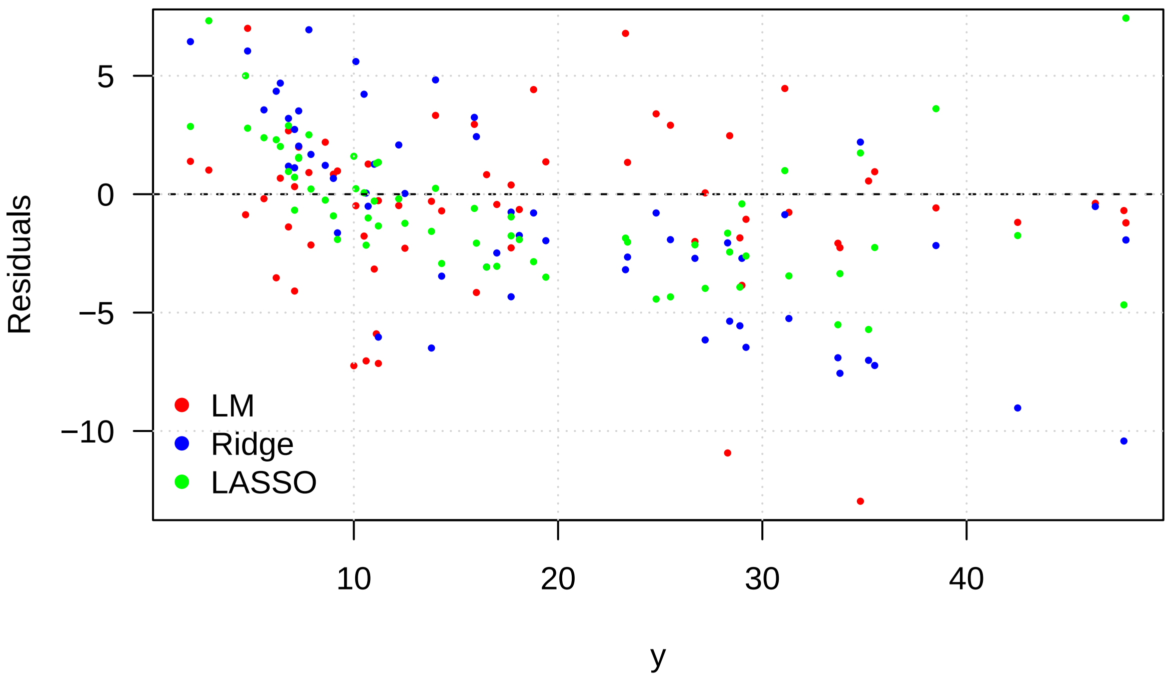
Answer:

The number of non-zero regression coefficients is 26.

```
plot(1:65, y[test], type = "l", las = 1, xlab = "", ylab = "", lwd = 1.5)
lines(1:65, predict(lmFit, testmeat), col = "red")
lines(1:65, ridge.pred, col = "blue")
lines(1:65, LASSO.pred, col = "green")
```




```
plot(y[test], predict(lmFit, testmeat) - y[test], pch = 16, cex = 0.5, col = "red",
     las = 1, xlab = "y", ylab = "Residuals")
points(y[test], ridge.pred - y[test], pch = 16, cex = 0.5, col = "blue")
points(y[test], LASSO.pred - y[test], pch = 16, cex = 0.5, col = "green")
abline(h = 0, lty = 2)
grid()
legend("bottomleft", legend = c("LM", "Ridge", "LASSO"),
       pch = 16, col = c("red", "blue", "green"),
       bty = "n")
```


5
0
−5
−10
Residuals
LM
Ridge
LASSO
10
20
30
40
y