Portfolio 5: Ridge Regression, LASSO and Smoothing

Kieran Morris

Task 1: Ridge and LASSO on Communities and Crime

For this task we will use the Communities and Crime dataset from the `mogavs` package, using the `glmnet` pakcage to perform LASSO regression. We begin by loading the data and creating a function which produces a training and test set for our data.

```
library(mogavs)
library(glmnet)
```

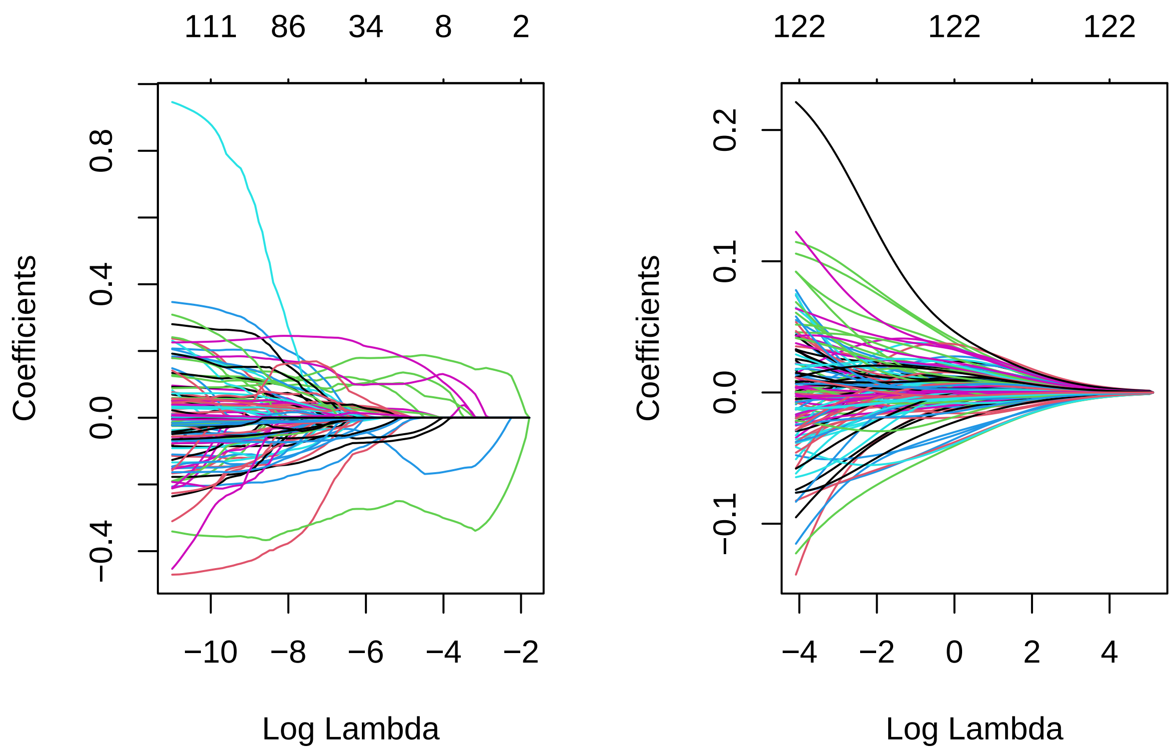
```
## Loading required package: Matrix

## Loaded glmnet 4.1-8
```

```
library(skimr)
data("crimeData")
set.seed(123)

crimedata <- scale(crimeData)
# Function to create training and test sets
createData <- function(data, trainProp = 0.7){
  n <- nrow(data)
  trainIndex <- sample(1:n, round(trainProp*n))
  trainData <- data[trainIndex,]
  testData <- data[-trainIndex,]
  return(list(trainData = trainData, testData = testData))
}

# Create training and test sets
data <- createData(crimeData)
traindata <- data$trainData
testdata <- data$testData

trainObs <- traindata[,-123]
testObs <- testdata[,-123]
trainY <- traindata[,123]
testY <- testdata[,123]
```

LASSO and Ridge Regression paths

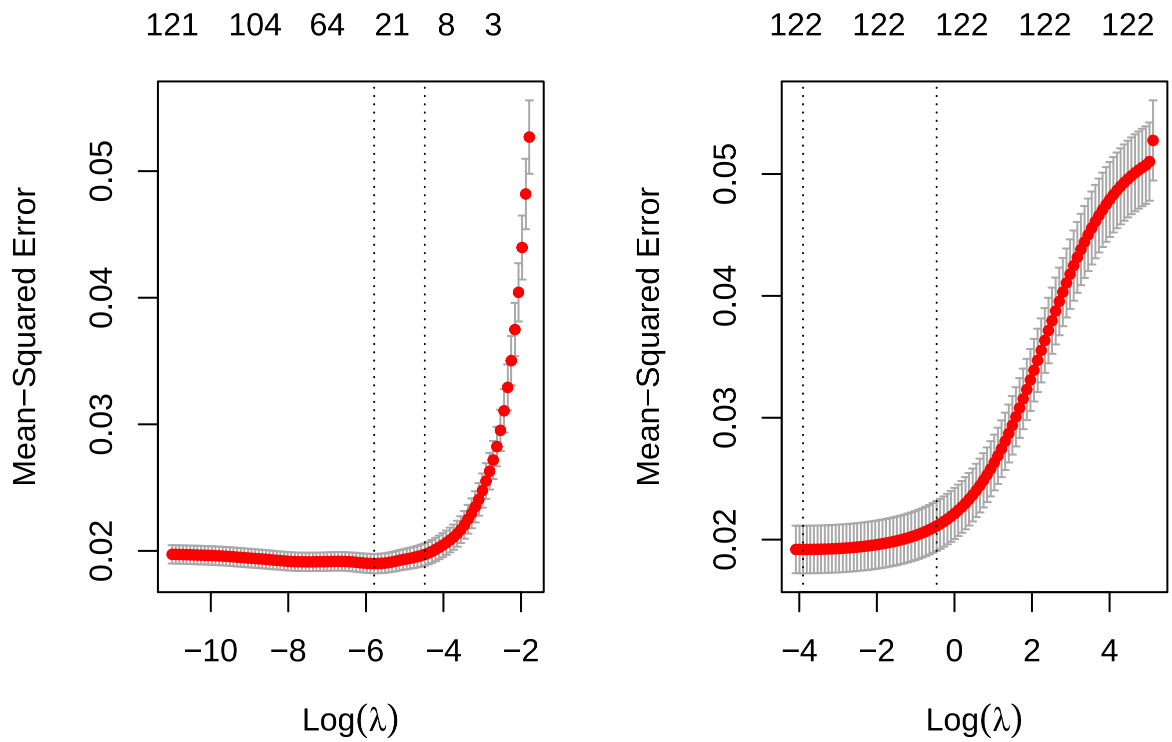
We begin with plotting the LASSO path, fortunately `glmnet` comes built in with both ridge and LASSO regression, we simply need to set `alpha = 0` and `alpha=1` respectively. We will fit our model with the training data and plot both paths.

```
library(ggplot2)

LASSO_model <- glmnet(trainObs,trainY, alpha = 1)
RIDGE_model <- glmnet(trainObs,trainY, alpha = 0)

par(mfrow=c(1,2))
plot(LASSO_model, xvar = "lambda")
plot(RIDGE_model, xvar = "lambda")
```



As expected the coefficients trail off to zero as our penalty variable λ increases. We now perform cross validation to find the optimal λ value.

```
cvLASSO <- cv.glmnet(as.matrix(trainObs), as.matrix(trainY), type.measure = "mse",alpha = 1)
cvRIDGE <- cv.glmnet(as.matrix(trainObs), as.matrix(trainY), type.measure = "mse", alpha = 0)

par(mfrow=c(1,2))
plot(cvLASSO)
plot(cvRIDGE)
```



We can see that the optimal $\log(\lambda)$ value for LASSO is around about -3, and for Ridge it is around 2 (when use a mean square error as our measure). Fortunately `cv.glmnet` auto generates the minimum value for us to print.

```
print(cvLASSO$lambda.min)
```

```
## [1] 0.003069441
```

```
print(cvRIDGE$lambda.min)
```

```
## [1] 0.02019487
```

We now see the values of β with the optimal λ values for both LASSO and Ridge.

```
LASSO_beta <- coef(cvLASSO, s = cvLASSO$lambda.min)
RIDGE_beta <- coef(cvRIDGE, s = cvRIDGE$lambda.min)

print(head(LASSO_beta))
```

```
## 6 x 1 sparse Matrix of class "dgCMatrix"
##                      s1
## (Intercept)  0.39200512
## x.V6         .
## x.V7         .
```

```
## x.V8         0.10590384
## x.V9        -0.05459534
## x.V10        .
```

```r
print(head(RIDGE_beta))
```

```
## 6 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  0.396078054
## x.V6        -0.020301189
## x.V7        -0.003593695
## x.V8         0.104087703
## x.V9        -0.075372205
## x.V10       -0.032028209
```

As we can see there are considerably many more zero entires in the `LASSO_beta` which is a result of the penalty term, let's find our the mean square error for both of these β values.

```r
LASSO_pred <- predict(cvLASSO, newx = as.matrix(testObs), s = cvLASSO$lambda.min)
RIDGE_pred <- predict(cvRIDGE, newx = as.matrix(testObs), s = cvRIDGE$lambda.min)

LASSO_MSE <- mean((testY - LASSO_pred)^2)
RIDGE_MSE <- mean((testY - RIDGE_pred)^2)

print(LASSO_MSE)
```

```
## [1] 0.04678155
```

```r
print(RIDGE_MSE)
```

```
## [1] 0.1010746
```

The LASSO regression is around half as big as the ridge regression, considering that the solution is also sparse, this leads us to believe that in this context LASSO is much more beneficial. As having sparse matrices can never be a bad thing, especially in the high dimensional case.

Task 2: Smoothing on mtcars

For this task we use ther `mtcars` dataset, an old favourite if you ask me. We will use the `gam` function from the `mgcv` package to perform the smoothing on the variable `mpg`, as it varies with the variable `hp`, we can use many types of splines, although for natural cubic splines we will set `bs = "cr"` in `gam`. Firstly we import the dataset and take our necessary variables.

```r
library(mgcv)
```

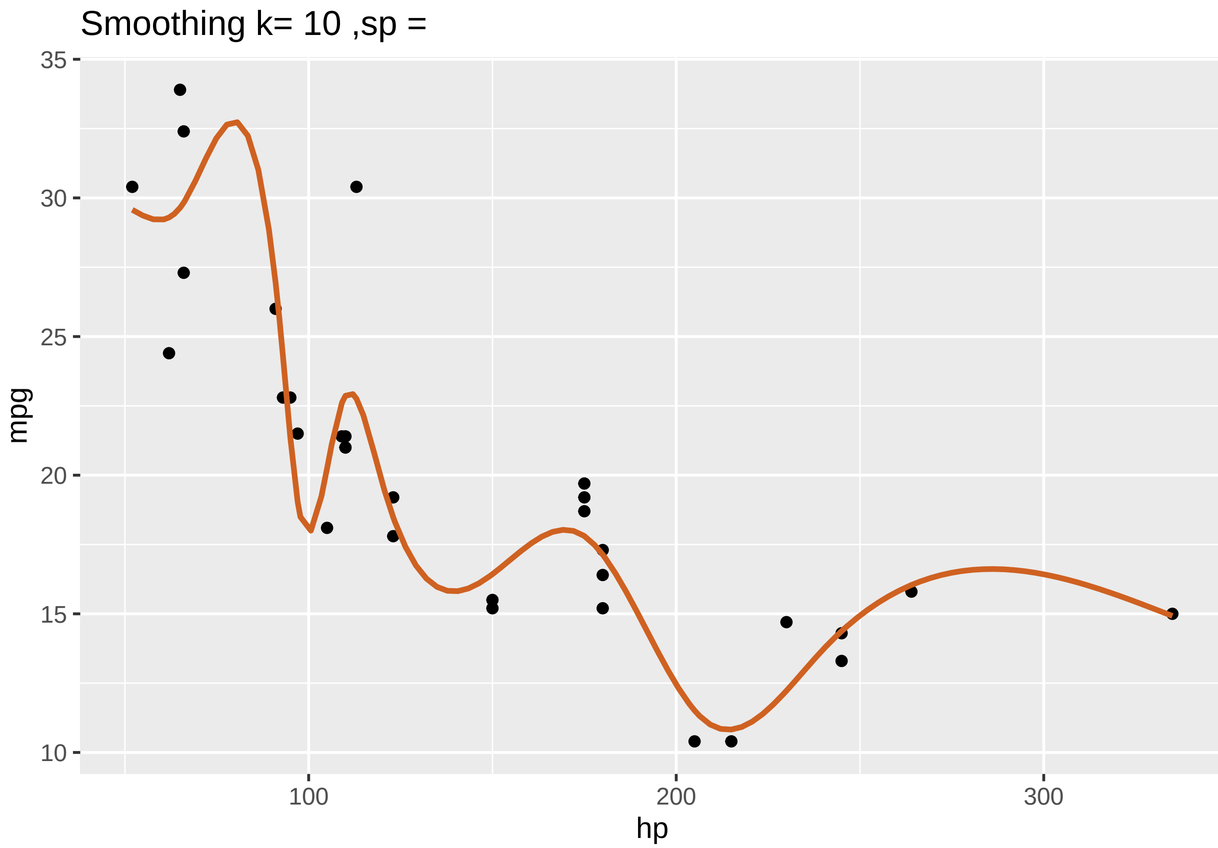
```
## Loading required package: nlme
```

```
## This is mgcv 1.9-1. For overview type 'help("mgcv-package")'.
```

```r
data(mtcars)
# Create training and test sets

dataY <- mtcars$mpg
dataX <- mtcars$hp
```

We now fit the model using the `gam` function, and plot the smoothing function over the original data using `ggplot` with `k= 10` and `sp = 0` (no penalty) for an example.

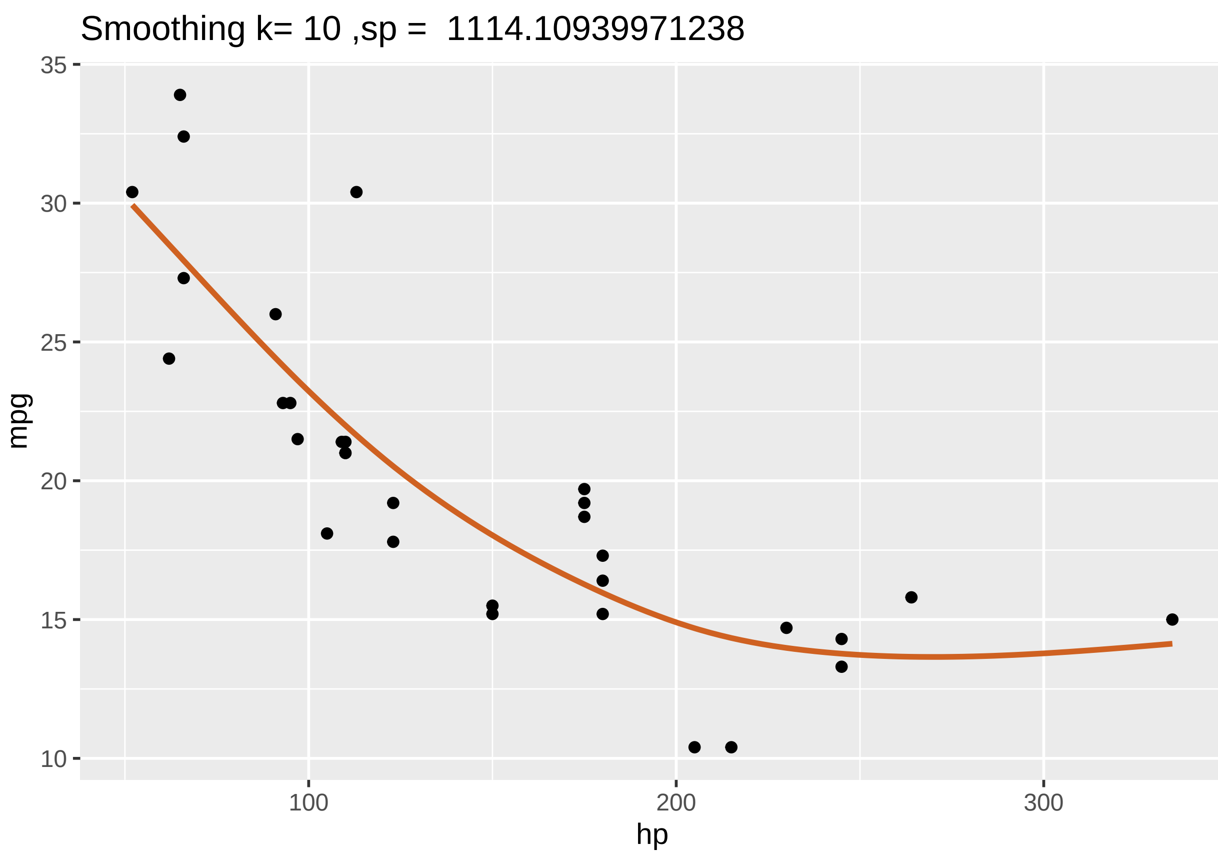
```r
plot_spline_model <- function(k,sp){
  if (sp==""){
    spline_model <- gam(dataY ~ s(dataX, bs = "cr",k=k))
  } else {
    spline_model <- gam(dataY ~ s(dataX, bs = "cr",k=k,sp=sp))
  }
# Load the ggplot2 package
library(ggplot2)

# Create a sequence of values within the range of dataX
new_dataX <- seq(min(dataX), max(dataX), length.out = 100)

# Predict the response for these values
predicted_dataY <- predict(spline_model, newdata = data.frame(dataX = new_dataX))

# Create a data frame for the original data and the predicted values
df <- data.frame(hp = c(dataX, new_dataX), mpg = c(dataY, rep(NA, length(new_dataX))), fitted = c(fitte

# Plot the data points and the fitted spline model
 plt <- ggplot(df, aes(x = hp)) +
  geom_point(aes(y = mpg), color = "#000000") +
  geom_line(aes(y = fitted), color = "#cf6121", size = 1, na.rm = TRUE) +
  labs(title = paste("Smoothing k=", k, ",sp = ",spline_model$sp), x = "hp", y = "mpg")
  return(plt)
}

plot_spline_model(10,0)
```



This does look nice, but perhaps a little to squiggly, so we can add a larger penalty term to smooth it out a bit. Fortunately if the argument `sd = NULL` in the `gam` function, it will automatically find the optimal `sp` via generalised cross validation.

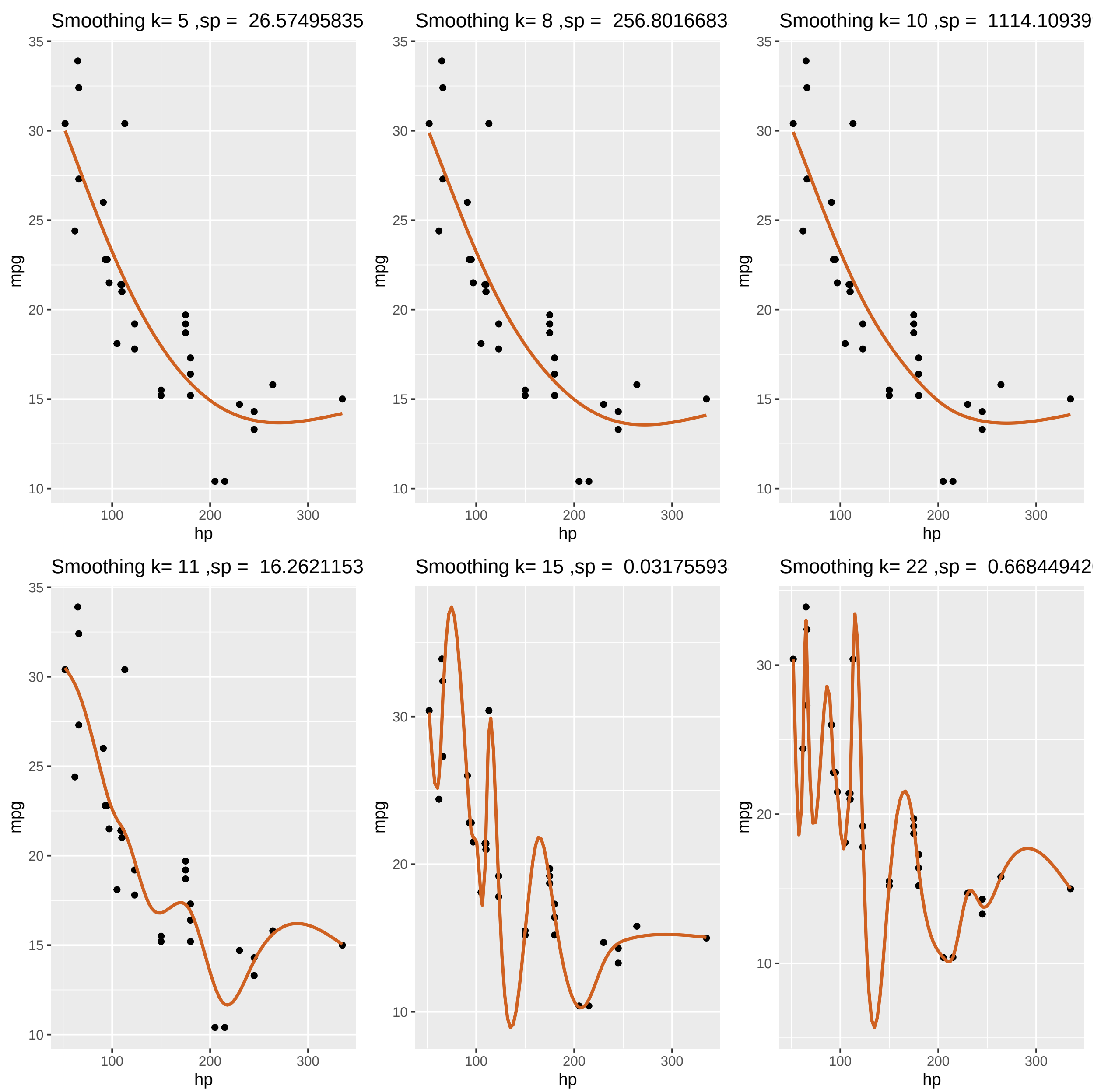
```
plot_spline_model(10,"")
```

```
## Warning: Removed 100 rows containing missing values (‘geom_point()‘).
```



Which is much better, lets see if we can do better with different values of `k`.

```
library(ggpubr)

plt1 <- plot_spline_model(5,"")
plt2 <- plot_spline_model(8,"")
plt3 <- plot_spline_model(10,"")
plt4 <- plot_spline_model(11,"")
plt5 <- plot_spline_model(15,"")
plt6 <- plot_spline_model(22,"")

ggarrange(plt1, plt2, plt3,plt4,plt5,plt6, ncol = 3,nrow = 2)
```



We can see that we reach an appropriate amount of smoothing at `k=10` and things kind of go downhill from there, or perhaps thats a bad analogy since we are constantly going up and downhill from there onwards. Even with an optimised `sd` penalty we suffer form overfitting.