Non-Parametric Regression and Shrinkage Methods

Blake Pappas

December 17, 2023

Non-Parametric Regression: Motorcycle Accident Simulation Data

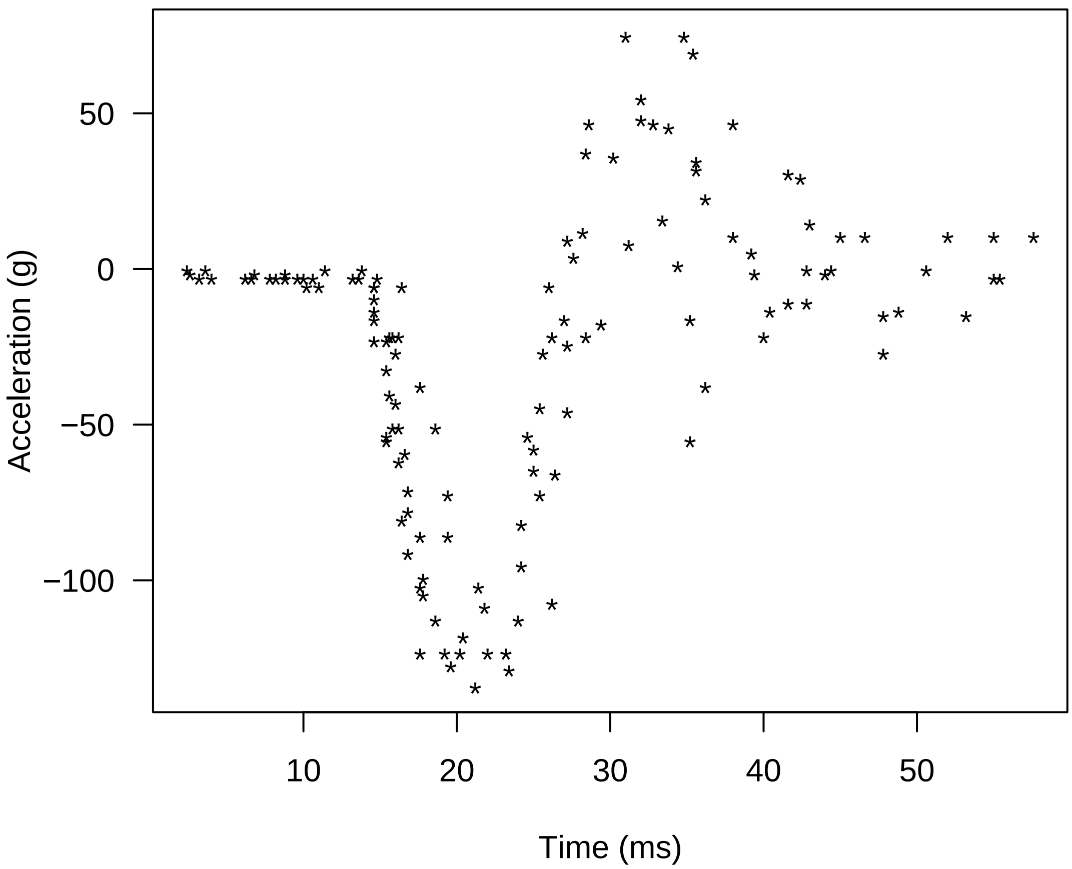
A data frame giving a series of measurements of head acceleration in a simulated motorcycle accident, used to test crash helmets.

- `times`: time (in milliseconds) after impact
- `accel`: head acceleration in g

Data Source: Silverman, B. W. (1985) Some aspects of the spline smoothing approach to non-parametric curve fitting. *Journal of the Royal Statistical Society series B* 47, 1–52.

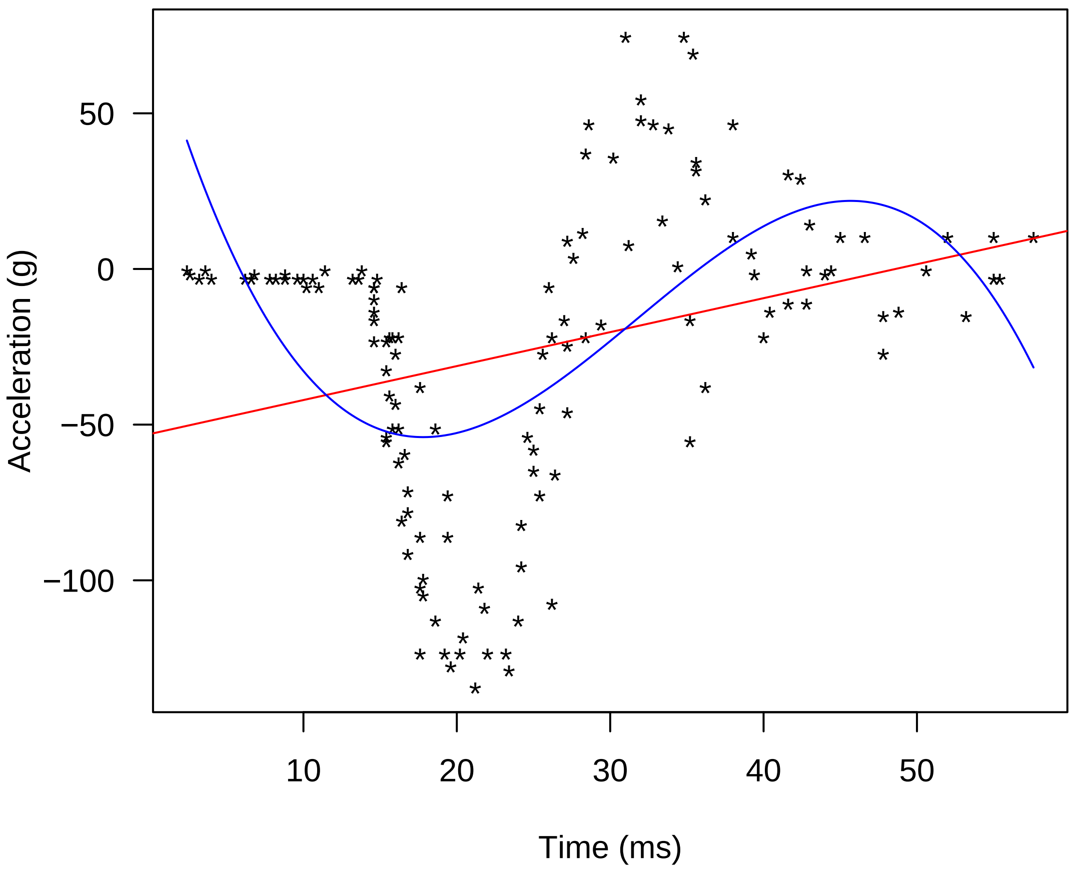
Load and Plot the Data

```
library(MASS) # Load the library
data(mcycle) # Load the data set
attach(mcycle)
plot(accel ~ times, pch = "*", cex = 1, las = 1,
     xlab = "Time (ms)", ylab = "Acceleration (g)")
```



Linear and Polynomial Regression Fits

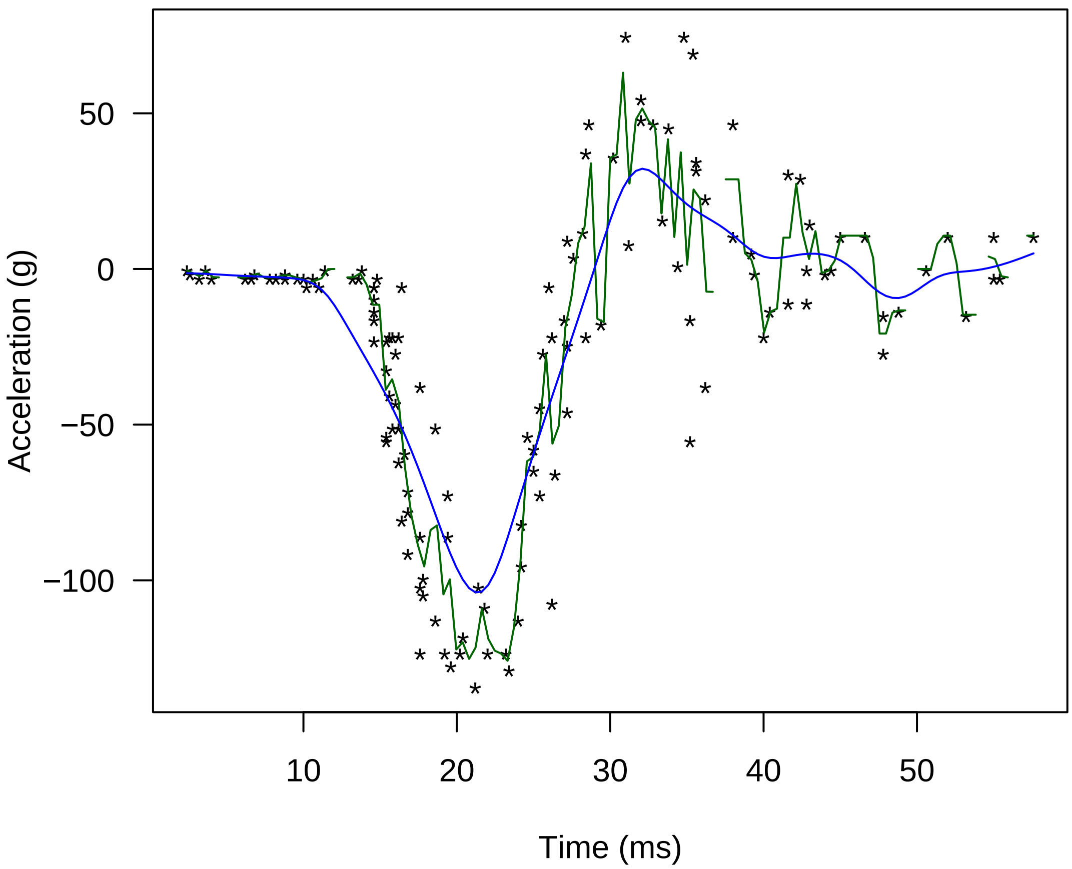
```r
# This Will Show How Bad a Linear Model Is for This Data Set
rg <- range(times)
xg = seq(rg[1], rg[2], 0.1) # Prediction grids

plot(times, accel, pch = "*", cex = 1, las = 1,
     xlab = "Time (ms)", ylab = "Acceleration (g)")
lmFit <- lm(accel ~ times, data = mcycle)
abline(lmFit, col = "red")
Cub.polyFit <- lm(accel ~ poly(times, 3), data = mcycle) # Polynomial Regression
Cub.polyPred <- predict(Cub.polyFit, data.frame(times = xg))
lines(xg, Cub.polyPred, col = "blue")
```



Kernel Regression

$\hat{f}(x) = \hat{\mathbb{E}}(Y|X = x) = \frac{\sum_{i=1}^{n} K_h(x-x_i)y_i}{\sum_{i=1}^{n} K_h(x-x_i)}$, where K_h is a kernel with a bandwidth h.

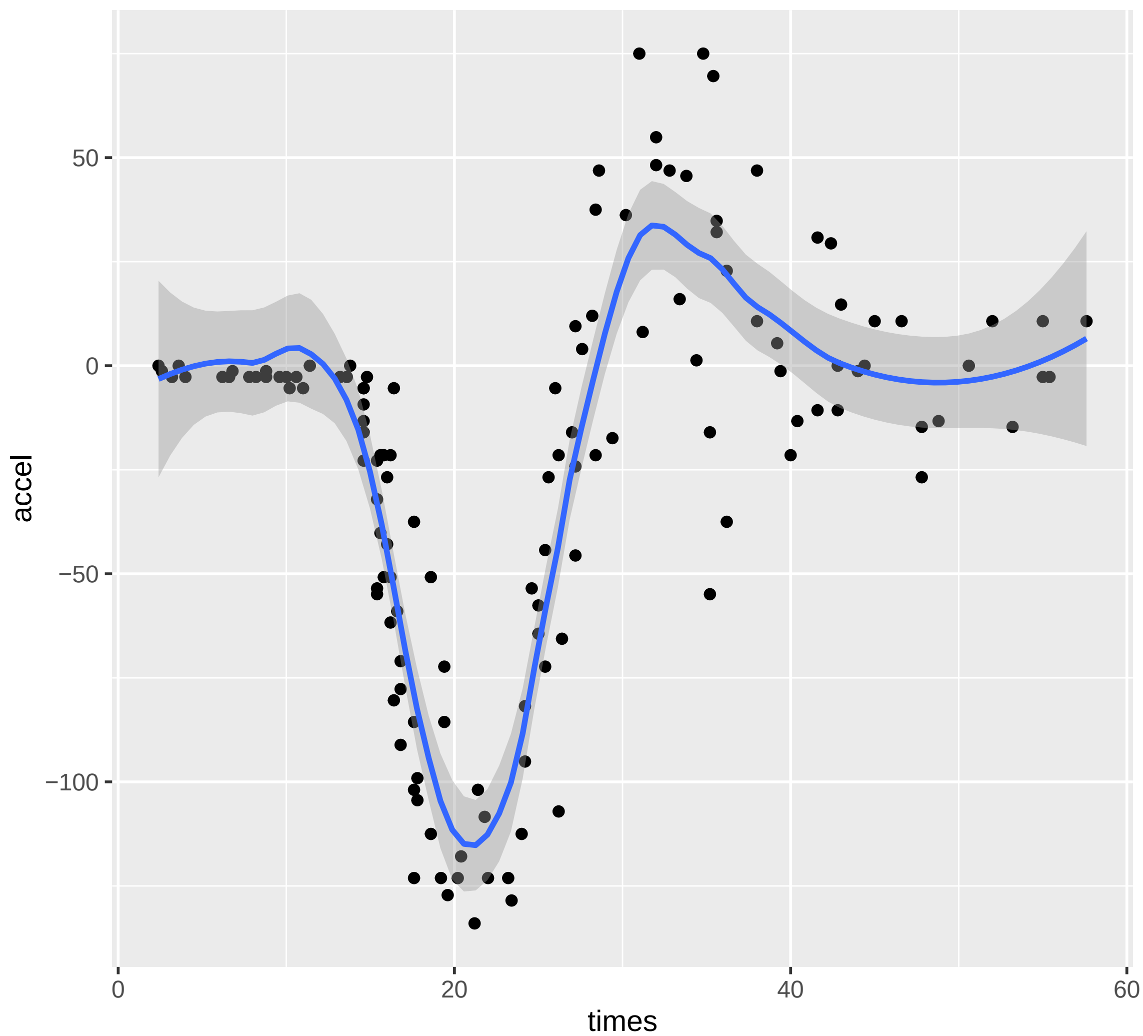
```
KernFit <- with(mcycle, ksmooth(times, accel, kernel = "normal", bandwidth = 0.5))
KernFit2 <- with(mcycle, ksmooth(times, accel, kernel = "normal", bandwidth = 5))

plot(times, accel, pch = "*", cex = 1, las = 1,
     xlab = "Time (ms)", ylab = "Acceleration (g)")
lines(KernFit$x, KernFit$y, col = "darkgreen")
lines(KernFit2$x, KernFit2$y, col = "blue")
```




```
# Green line --> smaller bandwidths
# Blue line --> larger bandwidths
# Blue line is better and smoother predictor
```

Local Polynomial Regression Fitting (*loess*)

```
library(ggplot2)
plot <- ggplot(aes(x = times, y = accel), data = mcycle)
plot <- plot + geom_point()
(plot <- plot + geom_smooth(method = "loess", degree = 2, span = 0.4, se = TRUE)) # Span is synonymous
```



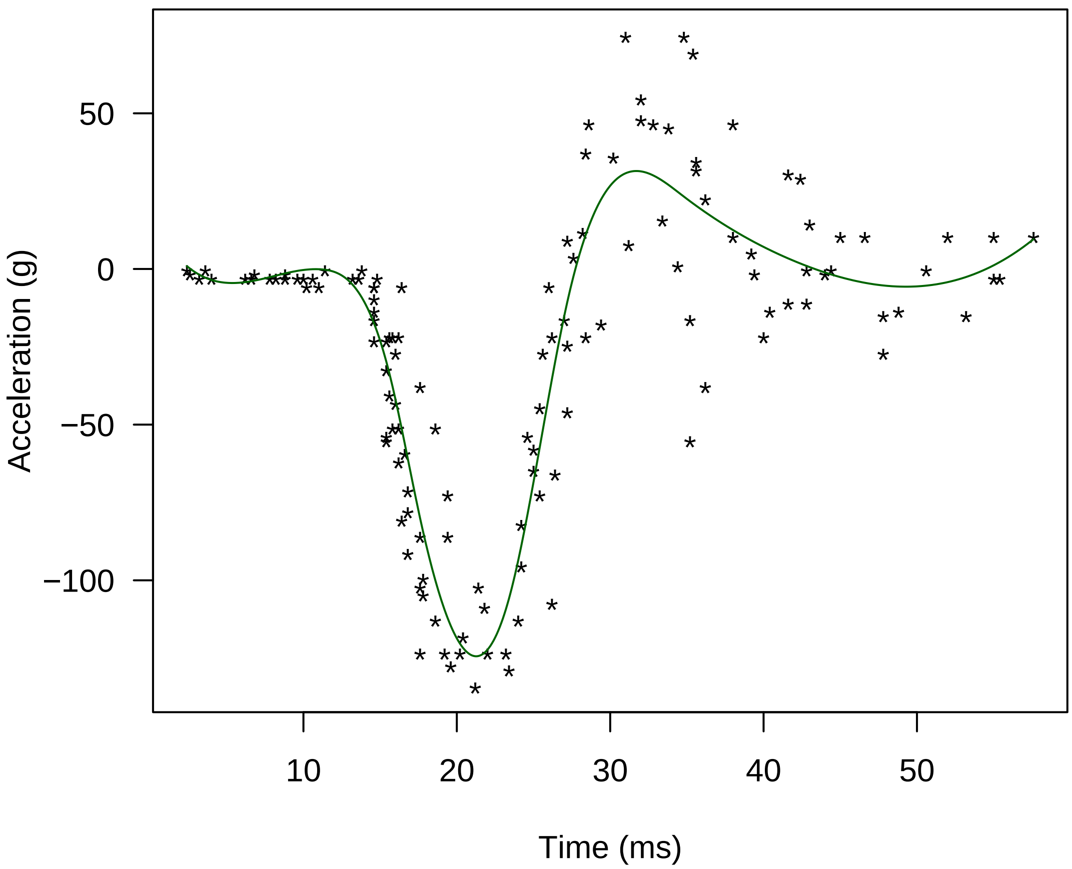
Regression Splines

```
# install.packages("splines")
library(splines)
RegSplineFit <- lm(accel ~ bs(times, df = 10), data = mcycle)
summary(RegSplineFit)
```

```
##
## Call:
## lm(formula = accel ~ bs(times, df = 10), data = mcycle)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -76.673 -12.362  -0.557  13.139  51.740
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)             0.9312    14.4492   0.064  0.94872
```

```
## bs(times, df = 10)1   -12.2008    37.5144  -0.325  0.74556    
## bs(times, df = 10)2     6.2223    23.6415   0.263  0.79284    
## bs(times, df = 10)3    -7.3726    18.2652  -0.404  0.68718    
## bs(times, df = 10)4  -118.7497    17.9975  -6.598 1.13e-09 ***
## bs(times, df = 10)5  -152.4486    20.0955  -7.586 7.25e-12 ***
## bs(times, df = 10)6    50.0827    18.7966   2.664  0.00875 ** 
## bs(times, df = 10)7    19.4271    19.3827   1.002  0.31819    
## bs(times, df = 10)8    -8.1814    23.9354  -0.342  0.73308    
## bs(times, df = 10)9   -11.1443    29.2202  -0.381  0.70358    
## bs(times, df = 10)10    8.6378    23.6119   0.366  0.71513    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 22.68 on 122 degrees of freedom
## Multiple R-squared:  0.7964, Adjusted R-squared:  0.7797 
## F-statistic: 47.72 on 10 and 122 DF,  p-value: < 2.2e-16
```

```
RegSplinePred <- predict(RegSplineFit, data.frame(times = xg))

plot(times, accel, pch = "*", cex = 1, las = 1,
     xlab = "Time (ms)", ylab = "Acceleration (g)")
lines(xg, RegSplinePred, col = "darkgreen")
```



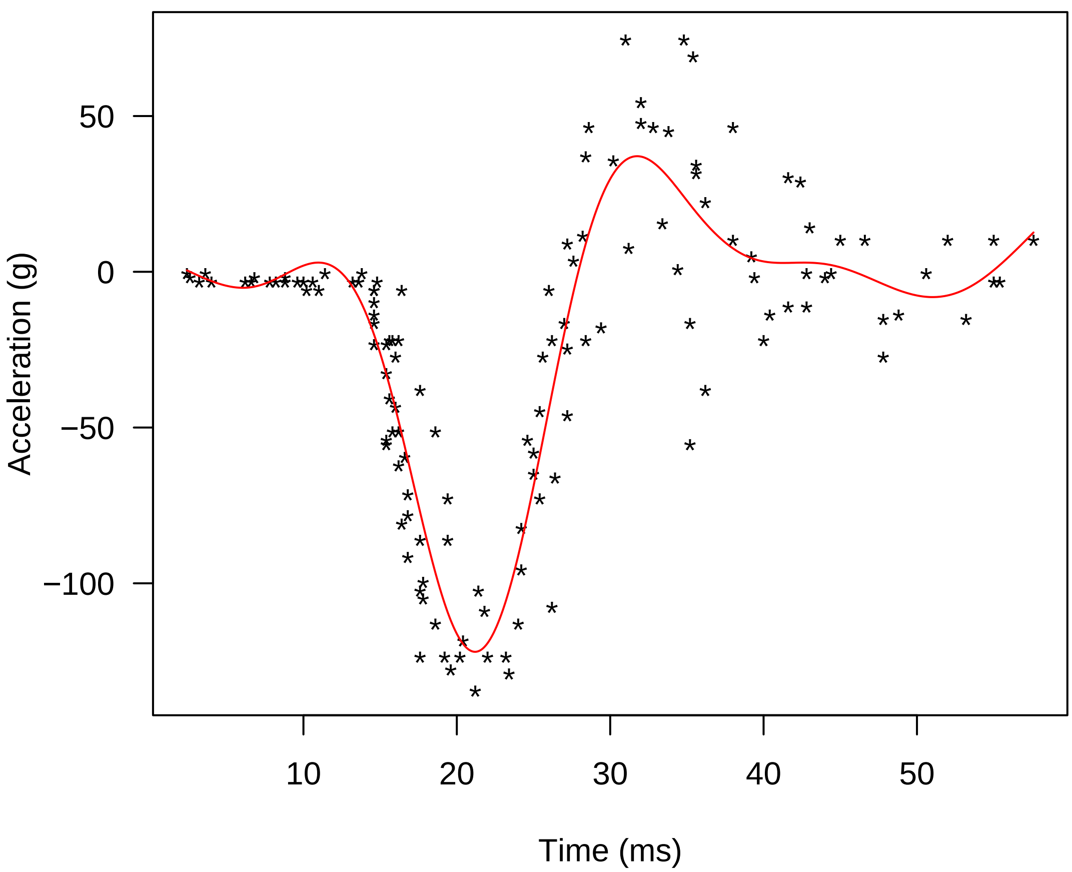
Generalized Additive Models

```
# install.packages("mgcv")
library(mgcv)
GAMFit <- gam(accel ~ s(times), data = mcycle)
summary(GAMFit)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## accel ~ s(times)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -25.546      1.951   -13.1   <2e-16 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##             edf Ref.df     F p-value
## s(times) 8.693  8.972 53.52  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.783   Deviance explained = 79.8%
## GCV = 545.78  Scale est. = 506       n = 133
```

```r
GAMpred <- predict(GAMFit, data.frame(times = xg))

plot(times, accel, pch = "*", cex = 1, las = 1,
     xlab = "Time (ms)", ylab = "Acceleration (g)")
lines(xg, GAMpred, col = "red")
```

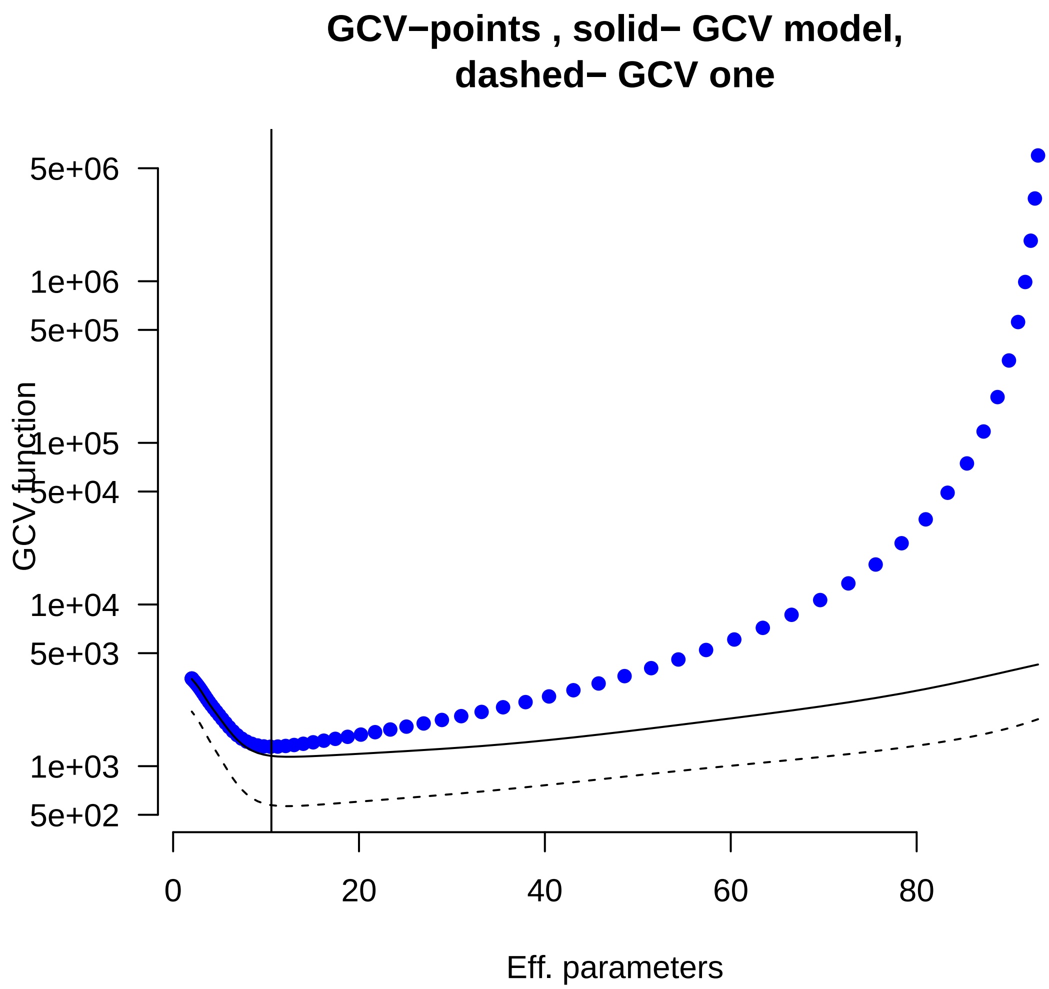


Smoothing Splines

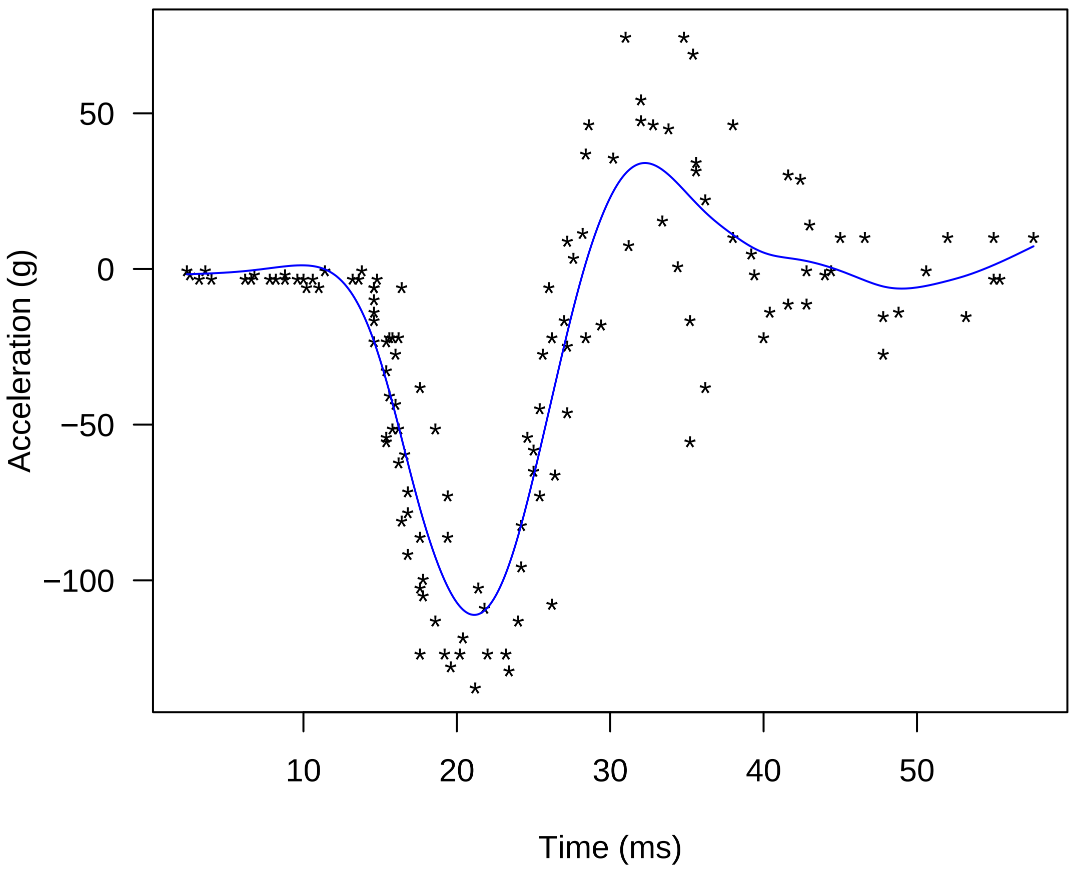
```r
# install.packages("fields")
library(fields)
SpFit <- sreg(times, accel)
summary(SpFit)
```

```
## CALL:
## sreg(x = times, y = accel)
##
##  Number of Observations:              133
##  Number of unique points:             133
##  Eff. degrees of freedom for spline: 10.6
##  Residual degrees of freedom:        122.4
##  GCV est. tau                         22.97
##  Pure error tau                       24.49
##  lambda                               0.3826
##
## RESIDUAL SUMMARY:
##      min    1st Q   median    3rd Q      max
## -78.1500 -13.8800  -0.7238  13.6300  49.6300
##
## DETAILS ON SMOOTHING PARAMETER:
##  Method used:      Cost:
##    lambda       trA       GCV   GCV.one GCV.model    tauHat
##    0.3826   10.5726 1318.0646  573.4152 1156.4850   22.9746
##
##  Summary of estimates for lambda
##            lambda    trA    GCV tauHat converge
## GCV        0.3826 10.573 1318.1  22.97       13
## GCV.model  0.1835 12.467 1142.5  22.64       12
## GCV.one    0.1981 12.253  565.5  22.66       12
## pure error 1.1041  8.375 1380.7  24.49       NA
```

```r
plot(SpFit, which = 3, col = "blue", pch = 16, las = 1)
```

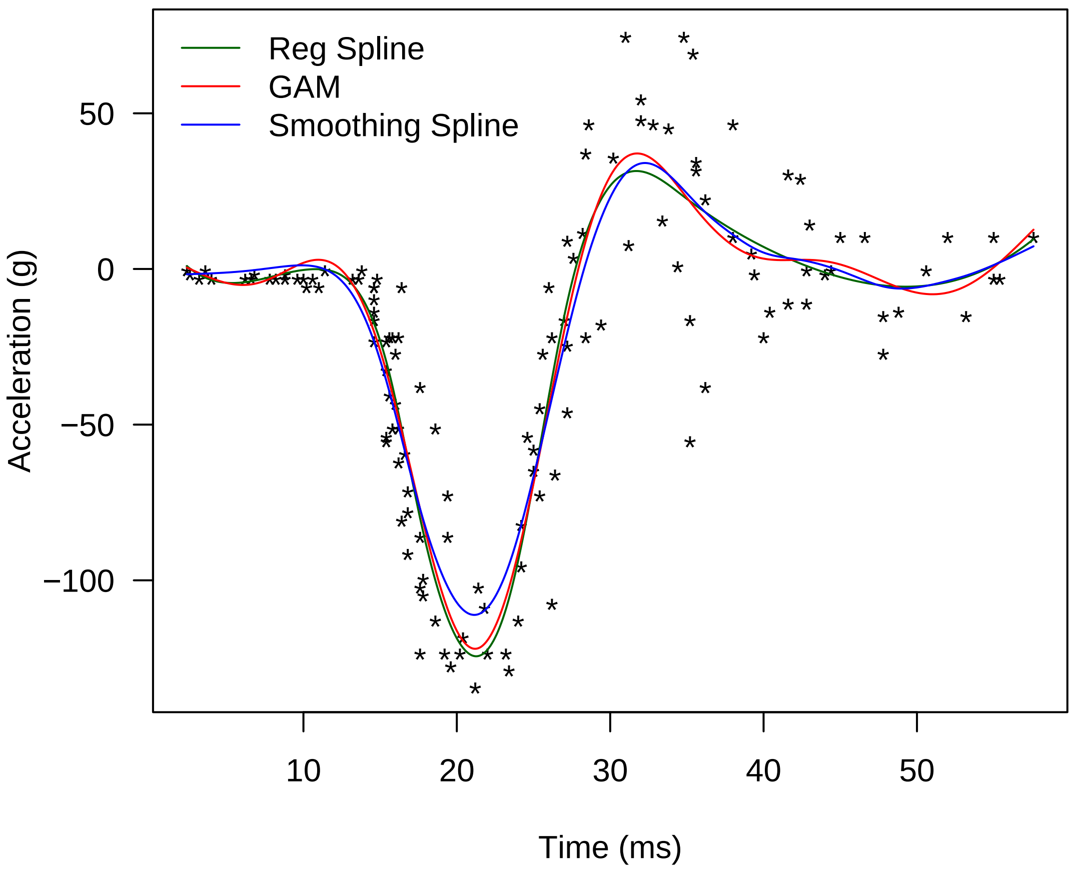



```
SpPred <- predict(SpFit, xg)

plot(times, accel, pch = "*", cex = 1, las = 1,
     xlab = "Time (ms)", ylab = "Acceleration (g)")
lines(xg, SpPred , col = "blue")
```



Comparing Regression Spline/GAM/Smoothing Spline Fits

```
plot(times, accel, pch = "*", cex = 1, las = 1,
     xlab = "Time (ms)", ylab = "Acceleration (g)")
lines(xg, RegSplinePred, col = "darkgreen")
lines(xg, GAMpred, col = "red")
lines(xg, SpPred, col = "blue")
legend("topleft", legend = c("Reg Spline", "GAM", "Smoothing Spline"),
       col = c("darkgreen", "red", "blue"), lty = 1, bty = "n")
```



Shrinkage Methods

The rest of this R session is largely based on the R lab: Ridge Regression and the Lasso of the book "Introduction to Statistical Learning with Applications in R" by *Gareth James*, *Daniela Witten*, *Trevor Hastie* and *Robert Tibshirani*.

The `glmnet` package will be used to perform ridge regression and the Lasso package will be used to predict `Salary` on the `Hitters` data.

Ridge Regression

1. Data Setup

```
# Predict the Salary of MLB Baseball Players

# install.packages("ISLR")
library(ISLR)
data(Hitters)
```

```r
Hitters = na.omit(Hitters) # Omits missing values
head(Hitters)
```

```
##                   AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun
## -Alan Ashby         315   81     7   24  38    39    14   3449   835     69
## -Alvin Davis        479  130    18   66  72    76     3   1624   457     63
## -Andre Dawson       496  141    20   65  78    37    11   5628  1575    225
## -Andres Galarraga   321   87    10   39  42    30     2    396   101     12
## -Alfredo Griffin    594  169     4   74  51    35    11   4408  1133     19
## -Al Newman          185   37     1   23   8    21     2    214    42      1
##                   CRuns CRBI CWalks League Division PutOuts Assists Errors
## -Alan Ashby         321  414    375      N        W     632      43     10
## -Alvin Davis        224  266    263      A        W     880      82     14
## -Andre Dawson       828  838    354      N        E     200      11      3
## -Andres Galarraga    48   46     33      N        E     805      40      4
## -Alfredo Griffin    501  336    194      A        W     282     421     25
## -Al Newman           30    9     24      N        E      76     127      7
##                   Salary NewLeague
## -Alan Ashby        475.0         N
## -Alvin Davis       480.0         A
## -Andre Dawson      500.0         N
## -Andres Galarraga   91.5         N
## -Alfredo Griffin   750.0         A
## -Al Newman          70.0         A
```

```r
summary(Hitters)
```

```
##      AtBat            Hits           HmRun            Runs
##  Min.   : 19.0   Min.   :  1.0   Min.   : 0.00   Min.   :  0.00
##  1st Qu.:282.5   1st Qu.: 71.5   1st Qu.: 5.00   1st Qu.: 33.50
##  Median :413.0   Median :103.0   Median : 9.00   Median : 52.00
##  Mean   :403.6   Mean   :107.8   Mean   :11.62   Mean   : 54.75
##  3rd Qu.:526.0   3rd Qu.:141.5   3rd Qu.:18.00   3rd Qu.: 73.00
##  Max.   :687.0   Max.   :238.0   Max.   :40.00   Max.   :130.00
##       RBI             Walks            Years            CAtBat
##  Min.   :  0.00   Min.   :  0.00   Min.   : 1.000   Min.   :   19.0
##  1st Qu.: 30.00   1st Qu.: 23.00   1st Qu.: 4.000   1st Qu.:  842.5
##  Median : 47.00   Median : 37.00   Median : 6.000   Median : 1931.0
##  Mean   : 51.49   Mean   : 41.11   Mean   : 7.312   Mean   : 2657.5
##  3rd Qu.: 71.00   3rd Qu.: 57.00   3rd Qu.:10.000   3rd Qu.: 3890.5
##  Max.   :121.00   Max.   :105.00   Max.   :24.000   Max.   :14053.0
##      CHits            CHmRun           CRuns             CRBI
##  Min.   :   4.0   Min.   :  0.00   Min.   :   2.0   Min.   :   3.0
##  1st Qu.: 212.0   1st Qu.: 15.00   1st Qu.: 105.5   1st Qu.:  95.0
##  Median : 516.0   Median : 40.00   Median : 250.0   Median : 230.0
##  Mean   : 722.2   Mean   : 69.24   Mean   : 361.2   Mean   : 330.4
##  3rd Qu.:1054.0   3rd Qu.: 92.50   3rd Qu.: 497.5   3rd Qu.: 424.5
##  Max.   :4256.0   Max.   :548.00   Max.   :2165.0   Max.   :1659.0
##      CWalks        League  Division    PutOuts          Assists
##  Min.   :   1.0   A:139   E:129    Min.   :   0.0   Min.   :  0.0
##  1st Qu.:  71.0   N:124   W:134    1st Qu.: 113.5   1st Qu.:  8.0
##  Median : 174.0                    Median : 224.0   Median : 45.0
##  Mean   : 260.3                    Mean   : 290.7   Mean   :118.8
```

```
##  3rd Qu.: 328.5                    3rd Qu.: 322.5   3rd Qu.:192.0
##  Max.   :1566.0                    Max.   :1377.0   Max.   :492.0
##      Errors           Salary       NewLeague
##  Min.   : 0.000   Min.   :  67.5   A:141
##  1st Qu.: 3.000   1st Qu.: 190.0   N:122
##  Median : 7.000   Median : 425.0
##  Mean   : 8.593   Mean   : 535.9
##  3rd Qu.:13.000   3rd Qu.: 750.0
##  Max.   :32.000   Max.   :2460.0
```

```
# install.packages("glmnet")
library(glmnet)
X <- model.matrix(Salary ~ ., data = Hitters)[, -1] # Creates a predictor variables
y <- Hitters$Salary # Creates a response variable
```

The `glmnet()` function has an alpha argument that determines what type of model is fit. If `alpha = 0`, then a ridge regression model is fit, and if `alpha = 1`, then a Lasso model is fit. We first fit a ridge regression model, which minimizes

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} beta_j^2,$$

where $\lambda \geq 0$ is a *tuning parameter* to be determined.

2. Fit a ridge regression over a grid of λ values.

```
grid <- 10^seq(10, -2, length = 100)
ridge.mod <- glmnet(X, y, alpha = 0, lambda = grid)
```

3. Ridge Regression Coefficients

```
dim(coef(ridge.mod))
```

```
## [1]  20 100
```

We expect the coefficient estimates to be much smaller, in terms of ℓ_2 norm, when a large value of λ is used.

```
ridge.mod$lambda[50] # Displays 50th lambda value
```

```
## [1] 11497.57
```

```
coef(ridge.mod)[, 50] # Displays coefficients associated with 50th lambda value
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
## 407.356050200   0.036957182   0.138180344   0.524629976   0.230701523
##           RBI         Walks         Years        CAtBat         CHits
##   0.239841459   0.289618741   1.107702929   0.003131815   0.011653637
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##   0.087545670   0.023379882   0.024138320   0.025015421   0.085028114
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
##  -6.215440973   0.016482577   0.002612988  -0.020502690   0.301433531
```

```r
sqrt(sum(coef(ridge.mod)[-1, 50]^2)) # Calculates l2 norm
```

```
## [1] 6.360612
```

In contrast, here are the coefficients when $\lambda = 705$, along with their ℓ_2 norm. Note the much larger l_2 norm of the coefficients associated with this smaller value of λ.

```r
ridge.mod$lambda[60] # Displays 60th lambda value
```

```
## [1] 705.4802
```

```r
coef(ridge.mod)[, 60] # Displays coefficients associated with 60th lambda value
```

```
##  (Intercept)        AtBat         Hits        HmRun         Runs          RBI 
##  54.32519950   0.11211115   0.65622409   1.17980910   0.93769713   0.84718546 
##        Walks        Years       CAtBat        CHits       CHmRun        CRuns 
##   1.31987948   2.59640425   0.01083413   0.04674557   0.33777318   0.09355528 
##         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists 
##   0.09780402   0.07189612  13.68370191 -54.65877750   0.11852289   0.01606037 
##       Errors   NewLeagueN 
##  -0.70358655   8.61181213
```

```r
sqrt(sum(coef(ridge.mod)[-1, 60]^2)) # Calculate l2 norm
```

```
## [1] 57.11001
```

We can use the `predict()` function for a number of purposes. For instance, we can obtain the ridge regression coefficients for a new value of λ, say 50:

```r
predict(ridge.mod, s = 50, type = "coefficients")[1:20, ]
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs 
##  4.876610e+01 -3.580999e-01  1.969359e+00 -1.278248e+00  1.145892e+00 
##           RBI         Walks         Years        CAtBat         CHits 
##  8.038292e-01  2.716186e+00 -6.218319e+00  5.447837e-03  1.064895e-01 
##        CHmRun         CRuns          CRBI        CWalks       LeagueN 
##  6.244860e-01  2.214985e-01  2.186914e-01 -1.500245e-01  4.592589e+01 
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN 
## -1.182011e+02  2.502322e-01  1.215665e-01 -3.278600e+00 -9.496680e+00
```

4. Training/Testing

We now split the samples into a training set and a test set in order to estimate the test error of ridge regression and later on the Lasso.

```r
set.seed(1)
train <- sample(1:nrow(X), nrow(X) / 2)
test <- (-train)
y.test <- y[test]

# Fit Ridge Regression to the Training Data
ridge.mod <- glmnet(X[train,], y[train], alpha = 0, lambda = grid, thresh = 1e-12)

# Predict the Salary to the Testing Data with Lambda = 4
ridge.pred <- predict(ridge.mod, s = 4, newx = X[test, ])

# Calculate the Root Mean Square Error (RMSE)
sqrt(mean((ridge.pred - y.test)^2))
```

```
## [1] 377.093
```

```r
# Compute the RMSE for the Intercept-Only Model
sqrt(mean((mean(y[train]) - y.test)^2))
```

```
## [1] 473.9936
```

```r
# Change to a Much Larger Lambda
ridge.pred <- predict(ridge.mod, s = 1e10, newx = X[test, ])
sqrt(mean((ridge.pred - y.test)^2))
```

```
## [1] 473.9935
```

```r
# Change Lambda to 0
ridge.pred <- predict(ridge.mod, s = 0, newx = X[test, ])
sqrt(mean((ridge.pred - y.test)^2))
```

```
## [1] 409.6215
```

```r
lm(y ~ X, subset = train)
```

```
##
## Call:
## lm(formula = y ~ X, subset = train)
##
## Coefficients:
## (Intercept)       XAtBat        XHits       XHmRun        XRuns         XRBI
##    274.0145      -0.3521      -1.6377       5.8145       1.5424       1.1243
##      XWalks       XYears      XCAtBat       XCHits      XCHmRun       XCRuns
##      3.7287     -16.3773      -0.6412       3.1632       3.4008      -0.9739
##       XCRBI      XCWalks     XLeagueN   XDivisionW     XPutOuts     XAssists
##     -0.6005       0.3379     119.1486    -144.0831       0.1976       0.6804
##     XErrors  XNewLeagueN
##     -4.7128     -71.0951
```

```r
predict(ridge.mod, s = 0, type = "coefficients")[1:20, ]
```

```
##   (Intercept)        AtBat         Hits        HmRun         Runs          RBI 
##   274.2089049   -0.3699455   -1.5370022    5.9129307    1.4811980    1.0772844 
##         Walks        Years       CAtBat        CHits       CHmRun        CRuns 
##     3.7577989  -16.5600387   -0.6313336    3.1115575    3.3297885   -0.9496641 
##          CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists 
##    -0.5694414    0.3300136  118.4000592 -144.2867510    0.1971770    0.6775088 
##        Errors   NewLeagueN 
##    -4.6833775  -70.1616132
```

Instead of arbitrarily choosing $\lambda = 4$, it would be better to use cross-validation (CV) to choose the tuning parameter λ. We can do this using the built-in cross-validation function, `cv.glmnet()`. By default, the function performs 10-fold cross-validation, though this can be changed using the argument `folds`.

5. Cross-Validation (CV)

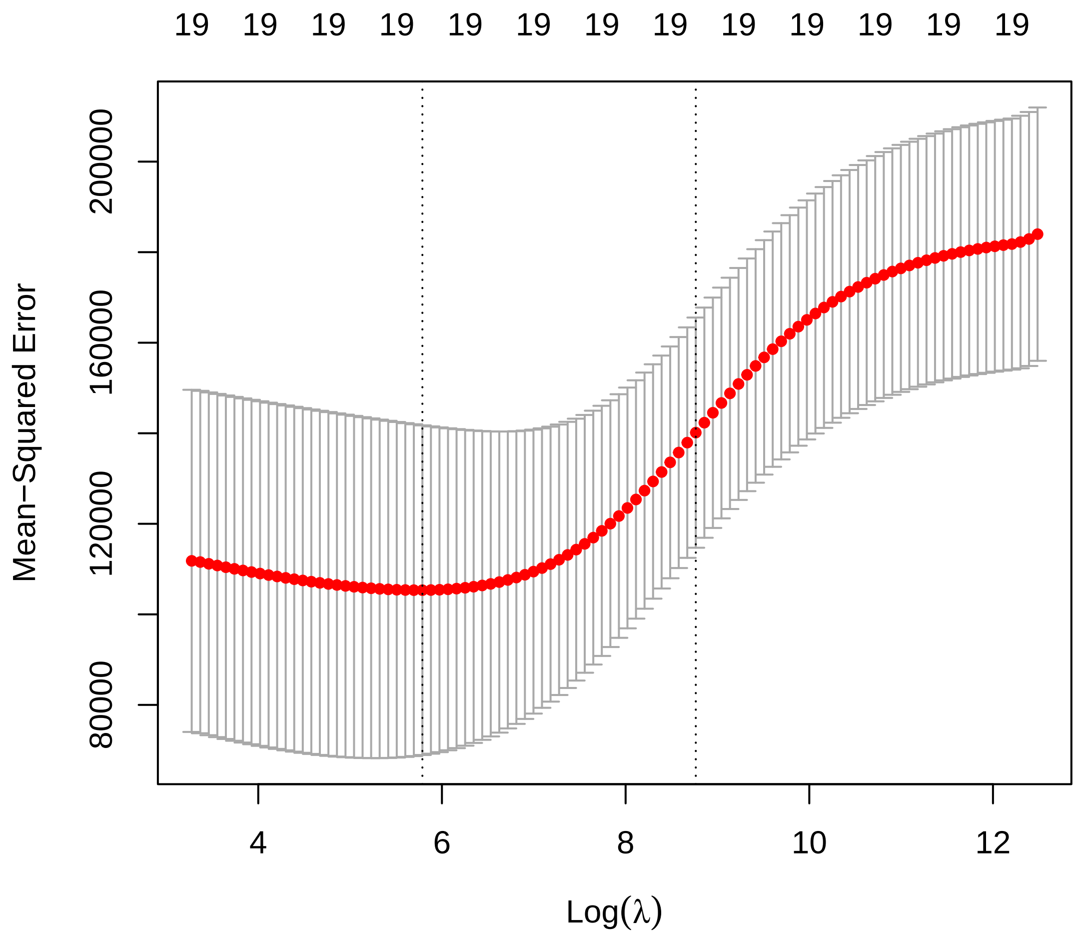
```r
set.seed(1)

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

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

```
## [1] 326.0828
```

```r
ridge.pred <- predict(ridge.mod, s = bestLambda, newx = X[test, ])
sqrt(mean((ridge.pred - y.test)^2))
```

```
## [1] 373.9741
```

```r
plot(cv.out) # Draw plot of training MSE as a function of lambda
```



Finally, we refit our ridge regression model on the full data set, using the value of λ chosen by cross-validation, and examine the coefficient estimates.

```r
# Fit Ridge Regression Model on Full Dataset
out <- glmnet(X, y, alpha = 0)

# Display Coefficients Using Lambda Chosen by CV
predict(out, type = "coefficients", s = bestLambda)[1:20, ]
```

```
##  (Intercept)        AtBat         Hits        HmRun         Runs          RBI
##  15.44383120   0.07715547   0.85911582   0.60103106   1.06369007   0.87936105
##        Walks        Years       CAtBat        CHits       CHmRun        CRuns
##   1.62444617   1.35254778   0.01134999   0.05746654   0.40680157   0.11456224
##         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists
##   0.12116504   0.05299202  22.09143197 -79.04032656   0.16619903   0.02941950
##       Errors   NewLeagueN
##  -1.36092945   9.12487765
```

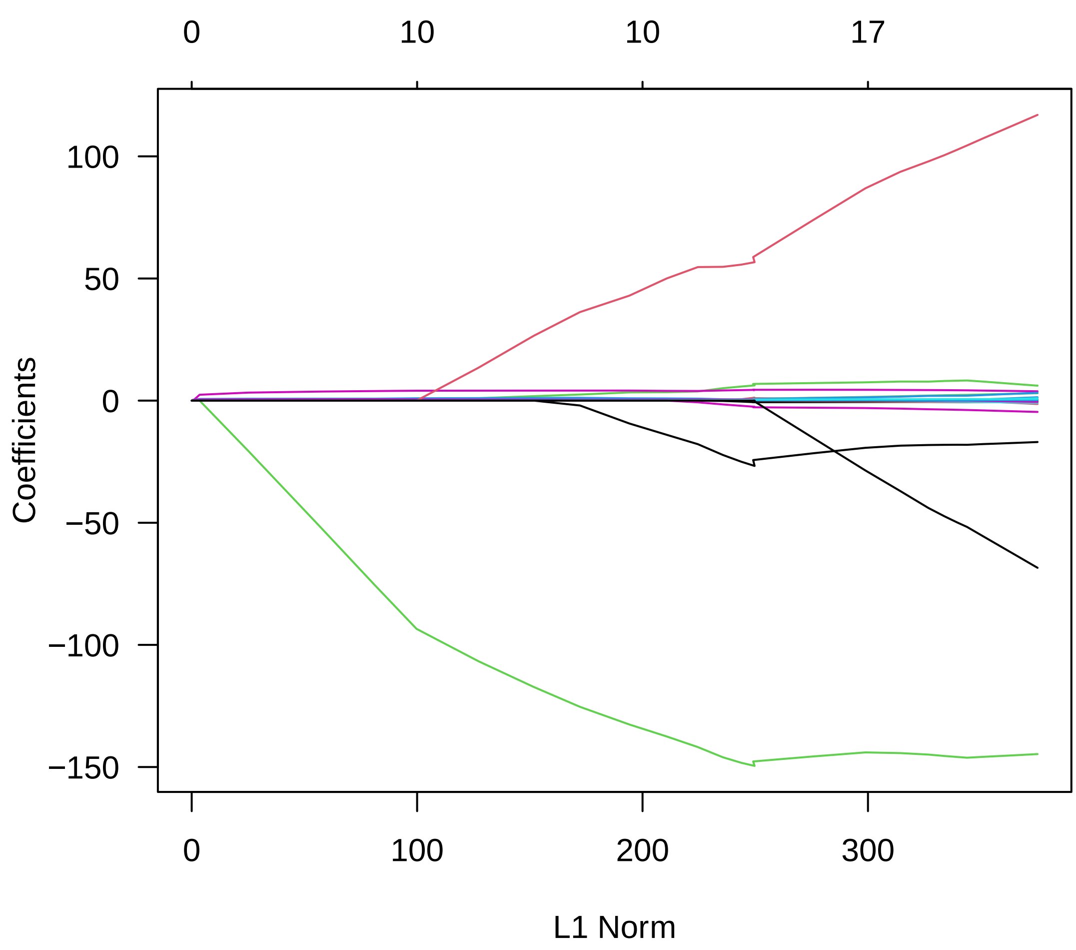
The Lasso

We see that the ridge regression with a wise choice of λ can outperform least squares as well as the null model on the Hitters data set. We now ask whether the Lasso, which minimizes

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

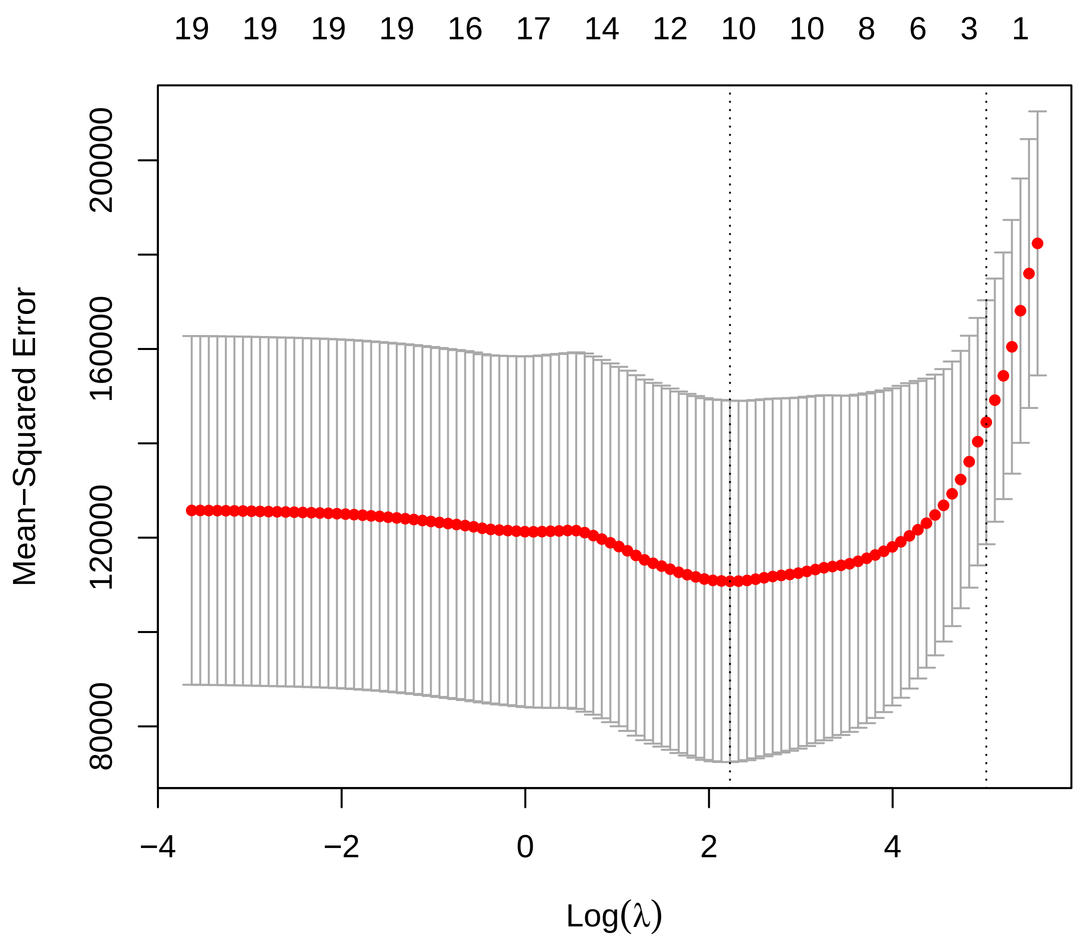
can yield either a more accurate or a more interpretable model than the ridge regression. In order to fit a Lasso model, we once again use the `glmnet()` function. However, this time we use the argument `alpha = 1`.

```
# Fit Lasso Model on Training Data
lasso.mod <- glmnet(X[train, ], y[train], alpha = 1, lambda = grid)

# Draw Plot of Coefficients
plot(lasso.mod, las = 1)
```



Notice that in the coefficient plot that, depending on the choice of tuning parameter, some of the coefficients are exactly equal to zero. We now perform cross-validation and compute the associated test error:

```r
set.seed(1)

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

# Draw Plot of Training MSE as a Function of Lambda
plot(cv.out)
```




```r
# Select Lambda That Minimizes Training MSE
bestLambda <- cv.out$lambda.min

# Use Best Lambda to Predict Test Data
lasso.pred <- predict(lasso.mod, s = bestLambda, newx = X[test, ])

# Calculate Test RMSE
sqrt(mean((lasso.pred - y[test])^2))
```

```
## [1] 379.043
```

This is substantially lower than the test set RMSE of the null model and of least squares, and very similar to the test RMSE of the ridge regression with λ chosen by cross-validation.

However, the Lasso has a substantial advantage over the ridge regression in that the resulting coefficient estimates are sparse. Here we see that 8 of the 19 coefficient estimates are exactly zero:

```r
# Fit Lasso Model on Full Dataset
out <- glmnet(X, y, alpha = 1, lambda = grid)

# Display Coefficients Using Lambda Chosen by CV
(lasso.coef <- predict(out, type = "coefficients", s = bestLambda)[1:20, ])
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
##    1.27479059   -0.05497143    2.18034583    0.00000000    0.00000000
##           RBI         Walks         Years        CAtBat         CHits
##    0.00000000    2.29192406   -0.33806109    0.00000000    0.00000000
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##    0.02825013    0.21628385    0.41712537    0.00000000   20.28615023
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
## -116.16755870    0.23752385    0.00000000   -0.85629148    0.00000000
```

```r
lasso.coef[lasso.coef != 0] # Display only non-zero coefficients
```

```
##   (Intercept)         AtBat          Hits         Walks         Years
##    1.27479059   -0.05497143    2.18034583    2.29192406   -0.33806109
##        CHmRun         CRuns          CRBI       LeagueN     DivisionW
##    0.02825013    0.21628385    0.41712537   20.28615023 -116.16755870
##       PutOuts        Errors
##    0.23752385   -0.85629148
```