STATS 202: Data Mining and Analysis

Instructor: Linh Tran

HOMEWORK # 3

Due date: August 6, 2021

Stanford University

Introduction

Homework problems are selected from the course textbook: *An Introduction to Statistical Learning*.

Problem 1 (7 points)

Chapter 6, Exercise 3 (p. 260).

As s increases, the regression model becomes more flexible, since the set $\{\beta : \sum_{j=1}^{p} |\beta_j| \leq s\}$ gets bigger. Therefore, the bias-variance tradeoff indicates that we'd expect:

(a) iv

(b) ii

(c) iii

(d) iv

(e) v

Problem 2 (7 points)

Chapter 6, Exercise 4 (p. 260).

The objective function can be re-written as follows:

$$\min \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{i=1}^{p} \beta_j x_{ij} \right)$$
$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 \leq s_\lambda,$$

where s_λ is decreasing in λ, though the exact relationship (in λ) is unclear. Hence, the similar argument for Problem 1 can be applied to this problem. Thus, we get

(a) iii

(b) ii

(c) iv

(d) iii

(e) v

Problem 3 (7 points)

Chapter 6, Exercise 9 (p. 263). Don't do parts (e), (f), and (g).

```
library(glmnet)
data(College)
```

(a)

```
set.seed(1)
n <- nrow(College)
training.idx <- sample(n, floor(0.8*n))
data.train <- College[training.idx,]
data.test <- College[-training.idx,]
X <- model.matrix(~ -1 + ., data.train[,-which(names(College) == "Apps")])
newX <- model.matrix(~ -1 + ., data.test[,-which(names(College) == "Apps")])
```

Note that the results can differ noticeably depending on (i) the seed that is set for the splits, and (ii) the proportion of the data set aside as the test set.

(b)

```
lm.fit <- lm(Apps ~ ., data = data.train)
y.hat <- predict(lm.fit, newdata=data.test)
lm.mse <- mean((data.test$Apps - y.hat)^2)
cat(sprintf('Test error: %d\n', round(lm.mse)))

## Test error: 1567324
```

The test error for the linear model is 1567324.

(c)

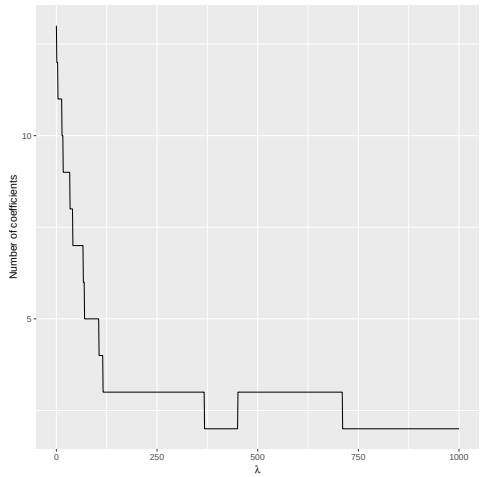
```
ridge.cv.fits <- cv.glmnet(X, data.train$Apps, alpha=0)
y.hat <- predict(ridge.cv.fits, newx = newX, type = "response",
                 "lambda.min")
ridge.mse <- mean((data.test$Apps - y.hat)^2)
cat(sprintf('Test error: %d\n', round(ridge.mse)))

## Test error: 1435788
```

The test error for the ridge model is 1435788.

(d)

```
lasso.cv.fits <- cv.glmnet(X, data.train$Apps, alpha=1,
                           lambda=seq(0, 1000, 1))
y.hat <- predict(lasso.cv.fits, newx = newX, type = "response",
                 "lambda.min")
lasso.mse <- mean((data.test$Apps - y.hat)^2)
cat(sprintf('Test error: %d\n', round(lasso.mse)))

## Test error: 1564303

coef <- predict(lasso.cv.fits, s=lasso.cv.fits$lambda.min,
                type='coefficients')
cat(sprintf('Number of non-zero coef: %d\n', sum(coef!=0)))

## Number of non-zero coef: 19
```

The test error for the Lasso model is 1564303. There are 13 coefficients greater than 0 (including the intercept). The figure below shows how the number of non-zero coefficients change over the various values of λ.



Problem 4 (7 points)

Chapter 7, Exercise 1 (p. 297).

(a) $\forall x \leq \xi$,

$$\begin{aligned} f_1(x) &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 \\ &= a_1 + b_1 x + c_1 x^2 + d_1 x^3. \end{aligned}$$

Hence, $\beta_0 = a_1, \beta_1 = b_1, \beta_2 = c_1, \beta_3 = d_1$.

(b) $\forall x > \xi$,

$$\begin{aligned} f_2(x) &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x - \xi)^3 \\ &= (\beta_0 + \xi^3) + (\beta_1 + 3\xi^2)x + (\beta_2 + 3\xi)x^2 + (\beta_3 + \beta_4)x^3 \\ &= a_2 + b_2 x + c_2 x^2 + d_2 x^3. \end{aligned}$$

Hence, $\beta_0 + \beta_4 \xi^3 = a_2, \beta_1 + 3\beta_4 \xi^2 = b_2, \beta_2 + 3\beta_4 \xi = c_2, \beta_3 + \beta_4 = d_2$. We conclude that $f(x)$ is a piecewise cubic polynomial.

(c) Evaluating $f(x)$ at ξ on each side of the knot

$$\begin{aligned} f(\xi+) &= \beta_0 + \beta_1 \xi + \beta_2 \xi^2 + \beta_3 \xi^3 + \beta_4 (\xi - \xi)^3 \\ &= \beta_0 + \beta_1 \xi + \beta_2 \xi^2 + \beta_3 \xi^3 \\ &= f(\xi-). \end{aligned}$$

Therefore $f(x)$ is continuous at ξ.

(d) Taking the derivatives on each side of the knot, and evaluating at ξ, leads to the same function of the coefficients:

$$f'(\xi+) = \beta_1 + 2\beta_2 \xi + 3\beta_3 \xi^2 + 3\beta_4 (\xi - \xi)^2 = \beta_1 + 2\beta_2 \xi + 3\beta_3 \xi^2 = f'(\xi-).$$

(e) Taking the second derivatives on each side of the knot, and evaluating at ξ leads to the same function of the coefficients:

$$f''(\xi+) = 2\beta_2 + 6\beta_3 \xi + 6\beta_4 (\xi - \xi) = 2\beta_2 + 6\beta_3 \xi = f''(\xi-).$$

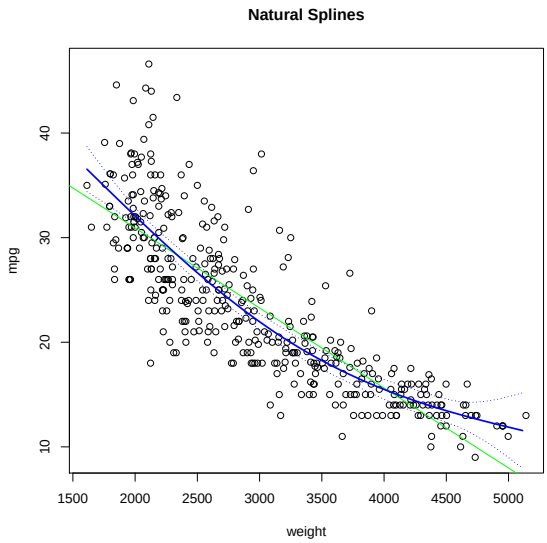
In parts (d) and (e) we showed that the first and second derivatives are continuous at the knot. We conclude that the function $f(x)$ is a cubic spline.

Problem 5 (7 points)

Chapter 7, Exercise 8 (p. 299). Find at least one non-linear estimate which does better than linear regression, and justify this using a t-test or by showing an improvement in the cross-validation error with respect to a linear model. You must also produce a plot of the predictor X vs. the non-linear estimate $\hat{f}(X)$.

Fitting a natural spline:

```
library(ISLR)
library(splines)
auto.lm.fit = lm(mpg ~ weight, data = Auto)
auto.ns.fit = lm(mpg ~ ns(weight, knots = c(2400, 3200, 4500)), data = Auto)

# Plot
wt.grid = seq(min(Auto$weight), max(Auto$weight), 50)
plot(mpg ~ weight, data = Auto, main = "Natural Splines")
abline(auto.lm.fit, col = "green")
ns.pred = predict(auto.ns.fit, newdata = list(weight = wt.grid), se = T)
se.bands = cbind(ns.pred$fit + 2 * ns.pred$se.fit, ns.pred$fit - 2 * ns.pred$se.fit)
lines(wt.grid, ns.pred$fit, lwd = 2, col = "blue")
matlines(wt.grid, se.bands, lwd = 1, col = "blue", lty = 3)
```



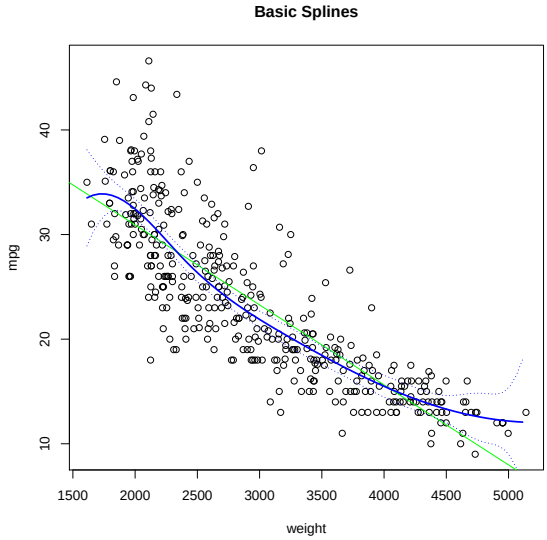
Comparing the model fits:

```
  anova(auto.lm.fit, auto.ns.fit)

## Analysis of Variance Table
##
## Model 1: mpg ~ weight
## Model 2: mpg ~ ns(weight, knots = c(2400, 3200, 4500))
##   Res.Df    RSS Df Sum of Sq      F      Pr(>F)
## 1    390 7321.2
## 2    387 6778.7  3     542.5 10.324 0.000001497 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The p-value comparing the linear model to the natural splines model is $\approx 10^{-6}$ indicating that a linear model is not justified in this case. The non-linear relation is also evident from the plot.

Fitting a basis spline:

```
auto.bs.fit = lm(mpg ~ bs(weight, knots = c(2400, 3200, 4500)), data = Auto)

# Plot
plot(mpg ~ weight, data = Auto, main = "Basic Splines")
abline(auto.lm.fit, col = "green")
bs.pred = predict(auto.bs.fit, newdata = list(weight = wt.grid), se = T)
se.bands = cbind(bs.pred$fit + 2 * bs.pred$se.fit, bs.pred$fit - 2 * bs.pred$se.fit)
lines(wt.grid, bs.pred$fit, lwd = 2, col = "blue")
matlines(wt.grid, se.bands, lwd = 1, col = "blue", lty = 3)
```



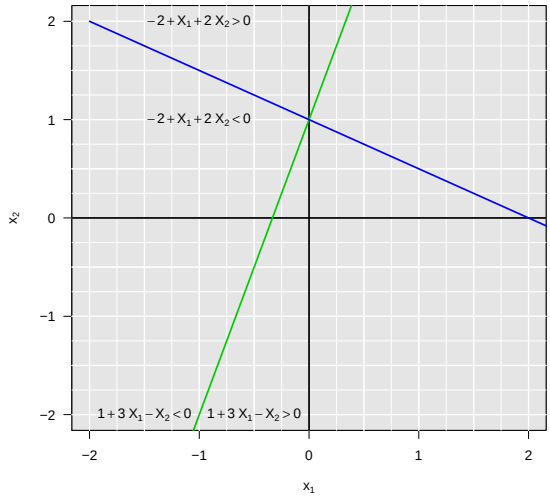
Comparing the model fits:

```
  anova(auto.lm.fit, auto.bs.fit)

## Analysis of Variance Table
##
## Model 1: mpg ~ weight
## Model 2: mpg ~ bs(weight, knots = c(2400, 3200, 4500))
##   Res.Df    RSS Df Sum of Sq      F     Pr(>F)
## 1    390 7321.2
## 2    385 6738.8  5    582.42 6.6549 0.00000584 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The p-value comparing the linear model to the natural splines model is $< 10^{-5}$ indicating that a linear model is not justified in this case. The non-linear relation is also evident from the plot.

Problem 6 (7 points)

Chapter 9, Exercise 1 (p. 368).



The green line represents the hyperplane $1 + 3X_1 - X_2 = 0$, while the blue line represents the hyperplane $-2 + X_1 + 2X_2 = 0$.

Problem 7 (8 points)

Chapter 9, Exercise 8 (p. 371).

```
library(ISLR)
library(e1071)
data(OJ)
```

(a)
```
set.seed(1)
n <- nrow(OJ)
training.idx <- sample(n, 800)
data.train <- OJ[training.idx,]
data.test <- OJ[-training.idx,]
```

Note that the results can differ noticeably depending on the seed that is set for the splits.

(b)
```
svm.linear.fit <- svm(Purchase ~., data = data.train, kernel = "linear",
                      cost = 0.01)
summary(svm.linear.fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = data.train, kernel = "linear",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  435
```

```
##
##  ( 219 216 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

(c)
```
  error.linear.train <- mean(svm.linear.fit$fitted != data.train$Purchase)
  y.hat <- predict(svm.linear.fit, data.test)
  error.linear.test <- mean(y.hat != data.test$Purchase)
  cat(sprintf('Train error: %0.3f\n', error.linear.train))
```

```
## Train error: 0.175
```

```
  cat(sprintf('Test error: %0.3f\n', error.linear.test))
```

```
## Test error: 0.178
```

Using the 0-1 loss, the training and test errors are 0.175 and 0.178, respectively.

(d)
```
  cost.range = list(cost = 10^seq(-2, 1, by = 0.15))
  tune.linear.fit = tune(svm, Purchase ~ ., data= data.train,
                         kernel="linear", ranges= cost.range)
  summary(tune.linear.fit)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      cost
##  7.079458
##
## - best performance: 0.17125
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.17375 0.03884174
## 2   0.01412538 0.17375 0.03747684
## 3   0.01995262 0.17375 0.03747684
## 4   0.02818383 0.17500 0.03726780
## 5   0.03981072 0.17625 0.03606033
## 6   0.05623413 0.18000 0.03073181
## 7   0.07943282 0.18000 0.03129164
## 8   0.11220185 0.18125 0.03240906
## 9   0.15848932 0.18000 0.03593976
## 10  0.22387211 0.18000 0.03291403
## 11  0.31622777 0.17875 0.03438447
## 12  0.44668359 0.17875 0.03438447
## 13  0.63095734 0.17625 0.03197764
## 14  0.89125094 0.17625 0.03197764
## 15  1.25892541 0.17500 0.03061862
```

```
## 16  1.77827941 0.17375 0.02972676
## 17  2.51188643 0.17250 0.03374743
## 18  3.54813389 0.17250 0.03322900
## 19  5.01187234 0.17250 0.03322900
## 20  7.07945784 0.17125 0.03438447
## 21 10.00000000 0.17125 0.03488573
```

```
  # Best fit
  tune.linear.fit$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = data.train,
##     ranges = cost.range, kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  7.079458
##
## Number of Support Vectors:  328
```

This indicates that the optimal cost is 7.079458.

(e)

```
  error.linear.train.tune <- mean(tune.linear.fit$best.model$fitted != data.train$Purchase)
  y.hat <- predict(tune.linear.fit$best.model, data.test)
  error.linear.test.tune <- mean(y.hat != data.test$Purchase)
  cat(sprintf('Train error: %0.3f\n', error.linear.train.tune))
```

```
## Train error: 0.163
```

```
  cat(sprintf('Test error: %0.3f\n', error.linear.test.tune))
```

```
## Test error: 0.152
```

Using the 0-1 loss, the training and test errors are 0.162 and 0.152, respectively.

(f)

```
  svm.radial.fit <- svm(Purchase ~., data = data.train, kernel = "radial",
                        cost = 0.01)
  summary(svm.radial.fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = data.train, kernel = "radial",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##
## Number of Support Vectors:  634
##
```

```
##  ( 319 315 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
  error.radial.train <- mean(svm.radial.fit$fitted != data.train$Purchase)
  y.hat <- predict(svm.radial.fit, data.test)
  error.radial.test <- mean(y.hat != data.test$Purchase)
  cat(sprintf('Train error: %0.3f\n', error.radial.train))
```

```
## Train error: 0.394
```

```
  cat(sprintf('Test error: %0.3f\n', error.radial.test))
```

```
## Test error: 0.378
```

```
  cost.range = list(cost = 10^seq(-2, 1, by = 0.15))
  tune.radial.fit = tune(svm, Purchase ~ ., data= data.train,
                         kernel="radial", ranges= cost.range)
  summary(tune.radial.fit)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##       cost
##  0.6309573
##
## - best performance: 0.1675
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.39375 0.06568284
## 2   0.01412538 0.39375 0.06568284
## 3   0.01995262 0.39375 0.06568284
## 4   0.02818383 0.38500 0.07378648
## 5   0.03981072 0.24250 0.07434903
## 6   0.05623413 0.19500 0.05109903
## 7   0.07943282 0.18250 0.05210833
## 8   0.11220185 0.18125 0.04611655
## 9   0.15848932 0.17750 0.04556741
## 10  0.22387211 0.17375 0.04656611
## 11  0.31622777 0.17250 0.04518481
## 12  0.44668359 0.17625 0.04619178
## 13  0.63095734 0.16750 0.04495368
## 14  0.89125094 0.17250 0.04031129
## 15  1.25892541 0.17250 0.04031129
## 16  1.77827941 0.17750 0.03622844
## 17  2.51188643 0.18000 0.04048319
## 18  3.54813389 0.18125 0.04379958
## 19  5.01187234 0.18125 0.04299952
## 20  7.07945784 0.18125 0.04135299
## 21 10.00000000 0.18125 0.04340139
```

```
  # Best fit
  tune.radial.fit$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = data.train,
##     ranges = cost.range, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.6309573
##
## Number of Support Vectors:  391
```

```
  error.radial.train.tune <- mean(tune.radial.fit$best.model$fitted != data.train$Purchase)
  y.hat <- predict(tune.radial.fit$best.model, data.test)
  error.radial.test.tune <- mean(y.hat != data.test$Purchase)
  cat(sprintf('Train error: %0.3f\n', error.radial.train.tune))
```

```
## Train error: 0.149
```

```
  cat(sprintf('Test error: %0.3f\n', error.radial.test.tune))
```

```
## Test error: 0.181
```

Using the 0-1 loss, the training and test errors for the untuned fit are 0.394 and 0.378, respectively.
Using `tune`, the optimal cost is 0.6309573.
Using the 0-1 loss, the training and test errors for the tuned fit are 0.149 and 0.181, respectively.

(g)
```
  svm.poly.fit <- svm(Purchase ~., data = data.train, kernel = "poly",
                      cost = 0.01, degree=2)
  summary(svm.poly.fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = data.train, kernel = "poly",
##     cost = 0.01, degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  636
##
##  ( 321 315 )
##
##
## Number of Classes:  2
##
```

```
## Levels:
##  CH MM
```

```
  error.poly.train <- mean(svm.poly.fit$fitted != data.train$Purchase)
  y.hat <- predict(svm.poly.fit, data.test)
  error.poly.test <- mean(y.hat != data.test$Purchase)
  cat(sprintf('Train error: %0.3f\n', error.poly.train))
```

```
## Train error: 0.372
```

```
  cat(sprintf('Test error: %0.3f\n', error.poly.test))
```

```
## Test error: 0.367
```

```
  cost.range = list(cost = 10^seq(-2, 1, by = 0.15))
  tune.poly.fit = tune(svm, Purchase ~ ., data= data.train,
                       kernel="poly", degree=2, ranges= cost.range)
  summary(tune.poly.fit)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      cost
##  7.079458
##
## - best performance: 0.1825
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.39000 0.08287373
## 2   0.01412538 0.37125 0.07337357
## 3   0.01995262 0.37000 0.07269609
## 4   0.02818383 0.36625 0.06509875
## 5   0.03981072 0.36500 0.05614960
## 6   0.05623413 0.34625 0.05434266
## 7   0.07943282 0.33000 0.06513874
## 8   0.11220185 0.31125 0.06958458
## 9   0.15848932 0.26250 0.07383352
## 10  0.22387211 0.22500 0.06972167
## 11  0.31622777 0.20875 0.05894029
## 12  0.44668359 0.20750 0.05688683
## 13  0.63095734 0.20625 0.05376453
## 14  0.89125094 0.20500 0.05627314
## 15  1.25892541 0.20125 0.05478810
## 16  1.77827941 0.19500 0.05809475
## 17  2.51188643 0.18625 0.05964304
## 18  3.54813389 0.18625 0.05816941
## 19  5.01187234 0.18375 0.05104804
## 20  7.07945784 0.18250 0.05041494
## 21 10.00000000 0.18625 0.05185785
```

```
  # Best fit
  tune.poly.fit$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = data.train,
##     ranges = cost.range, kernel = "poly", degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  7.079458
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  358
```

```
  error.poly.train.tune <- mean(tune.poly.fit$best.model$fitted != data.train$Purchase)
  y.hat <- predict(tune.poly.fit$best.model, data.test)
  error.poly.test.tune <- mean(y.hat != data.test$Purchase)
  cat(sprintf('Train error: %0.3f\n', error.poly.train.tune))
```

```
## Train error: 0.150
```

```
  cat(sprintf('Test error: %0.3f\n', error.poly.test.tune))
```

```
## Test error: 0.189
```

Using the 0-1 loss, the training and test errors for the untuned fit are 0.372 and 0.367, respectively.

Using `tune`, the optimal cost is 7.079458.

Using the 0-1 loss, the training and test errors for the tuned fit are 0.15 and 0.189, respectively.

(h) It appears the linear kernel gives the best test error (0.152), while the radial kernel gives the best training error (0.149).