Econ 573: Problem Set 4

Harvey Duperier

2022-11-01

```
library(ISLR2)
library(gbm)
```

```
## Loaded gbm 2.1.8.1
```

```
library(glmnet)
```

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

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

```
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:ISLR2':
##
##     Boston
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
```

```
## v ggplot2 3.3.6      v purrr   0.3.4
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::combine()  masks randomForest::combine()
## x tidyr::expand()   masks Matrix::expand()
```

```
## x dplyr::filter()   masks stats::filter()
## x dplyr::lag()      masks stats::lag()
## x ggplot2::margin() masks randomForest::margin()
## x tidyr::pack()     masks Matrix::pack()
## x dplyr::select()   masks MASS::select()
## x tidyr::unpack()   masks Matrix::unpack()
```

```
library(ggplot2)
library(ggthemes)
library(broom)
library(knitr)
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(splines)
library(boot)
```

```
##
## Attaching package: 'boot'
##
## The following object is masked from 'package:lattice':
##
##     melanoma
```

Part I

Question 3 Chapter 5

3a). The k-fold cross-validation is implemented by dividing the set of observations into k folds of approximately equal size. This first fold is treated as a validation set, and the method is fit on the remaining k-1 folds. The mean squared error is then computed on the observations in the held-out gold. This procedure is repeated k times; each time, a different group of observations is treated as a validation set. This process results in k estimates of the test error,

$$MSE_1$$

,

$$MSE_2$$

, ...,

$$MSE_k$$

. The k-fold CV estimate is computer by averaging these values.

3b). The advantages of k-fold CV relative to the validation set approach is that the validation set approach error rate may tend to overestimate the test error rate for the model fit on the entire data set. The disadvantages of the k-fold CV relative to the validation set approach is that the validation set approach is

conceptually simpler than k-fold and easily implemented as it only involves partitioning the existing training data into two sets. The advantages of k-fold CV relative to LOOCV is computation. Since LOOCV is a special case of k-fold CV in which k is set to equal n, it has the potential to be computationally expensive, whereas performing 10-fold CV requires fitting the learning procedure only ten times, which could be much more feasible. The disadvantages of k-fold CV relative to LOOCV is that LOOCV has a higher variance, but lower bias, than the k-fold CVs.

Question 5 Chapter 5

5a).

```
data("Default")
set.seed(1)
glm.fit <- glm(default ~ income + balance, data = Default, family = binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = default ~ income + balance, family = binomial,
##     data = Default)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4725  -0.1444  -0.0574  -0.0211   3.7245
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.154e+01  4.348e-01 -26.545  < 2e-16 ***
## income       2.081e-05  4.985e-06   4.174 2.99e-05 ***
## balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1579.0  on 9997  degrees of freedom
## AIC: 1585
##
## Number of Fisher Scoring iterations: 8
```

5bi).

```
train = sample(dim(Default)[1], dim(Default)[1] / 2)
```

5bii).

```
fit.glm = glm(default ~ income + balance, data = Default, family = "binomial", subset = train)
summary(fit.glm)
```

```
##
```

```
## Call:
## glm(formula = default ~ income + balance, family = "binomial",
##     data = Default, subset = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5830  -0.1428  -0.0573  -0.0213   3.3395
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.194e+01  6.178e-01 -19.333  < 2e-16 ***
## income       3.262e-05  7.024e-06   4.644 3.41e-06 ***
## balance      5.689e-03  3.158e-04  18.014  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1523.8  on 4999  degrees of freedom
## Residual deviance:  803.3  on 4997  degrees of freedom
## AIC: 809.3
##
## Number of Fisher Scoring iterations: 8
```

5biii).

```
glm.probs = predict(fit.glm, newdata = Default[-train, ], type="response")
glm.pred=rep("No",5000)
glm.pred[glm.probs>0.5] = "Yes"
```

5biv).

```
mean(glm.pred != Default[-train, ]$default)
```

```
## [1] 0.0254
```

5c).

```
train <- sample(dim(Default)[1], dim(Default)[1] / 2)
fit.glm <- glm(default ~ income + balance, data = Default, family = "binomial", subset = train)
probs <- predict(fit.glm, newdata = Default[-train, ], type = "response")
pred.glm <- rep("No", length(probs))
pred.glm[probs > 0.5] <- "Yes"
mean(pred.glm != Default[-train, ]$default)
```

```
## [1] 0.0274
```

```
train <- sample(dim(Default)[1], dim(Default)[1] / 2)
fit.glm <- glm(default ~ income + balance, data = Default, family = "binomial", subset = train)
probs <- predict(fit.glm, newdata = Default[-train, ], type = "response")
pred.glm <- rep("No", length(probs))
pred.glm[probs > 0.5] <- "Yes"
mean(pred.glm != Default[-train, ]$default)
```

```
## [1] 0.0244
```

```
train <- sample(dim(Default)[1], dim(Default)[1] / 2)
fit.glm <- glm(default ~ income + balance, data = Default, family = "binomial", subset = train)
probs <- predict(fit.glm, newdata = Default[-train, ], type = "response")
pred.glm <- rep("No", length(probs))
pred.glm[probs > 0.5] <- "Yes"
mean(pred.glm != Default[-train, ]$default)
```

```
## [1] 0.0244
```

From the work above, we see that the validation estimate of the Test Error can be variable, depending on the difference in which observations are included in the training and validation set.

5d).

```
train <- sample(dim(Default)[1], dim(Default)[1] / 2)
fit.glm <- glm(default ~ income + balance + student, data = Default, family = "binomial", subset = trai
pred.glm <- rep("No", length(probs))
probs <- predict(fit.glm, newdata = Default[-train, ], type = "response")
pred.glm[probs > 0.5] <- "Yes"
mean(pred.glm != Default[-train, ]$default)
```

```
## [1] 0.0278
```

Based off the results above, the student dummy variable does not lead to a reduction in the validation set estimate of the test error rate.

Question 6 Chapter 5

6a).

```
set.seed(1)
train <- sample(dim(Default)[1], dim(Default)[1] / 2)
fit.glm <- glm(default ~ income + balance, data = Default, family = "binomial", subset = train)
summary(fit.glm)
```

```
##
## Call:
## glm(formula = default ~ income + balance, family = "binomial",
##     data = Default, subset = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5830  -0.1428  -0.0573  -0.0213   3.3395
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.194e+01  6.178e-01 -19.333  < 2e-16 ***
## income       3.262e-05  7.024e-06   4.644 3.41e-06 ***
## balance      5.689e-03  3.158e-04  18.014  < 2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1523.8  on 4999  degrees of freedom
## Residual deviance:  803.3  on 4997  degrees of freedom
## AIC: 809.3
##
## Number of Fisher Scoring iterations: 8
```

The standard errors are given in the summary above from fit.glm.

6b).

```
boot.fn <- function(data, index) {
    fit <- glm(default ~ income + balance, data = data, family = "binomial", subset = index)
    return (coef(fit))}
```

6c).

```
boot(Default, boot.fn, 1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Default, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##          original        bias     std. error
## t1* -1.154047e+01 -3.912114e-02 4.347403e-01
## t2*  2.080898e-05  1.585717e-07 4.858722e-06
## t3*  5.647103e-03  1.856917e-05 2.300758e-04
```

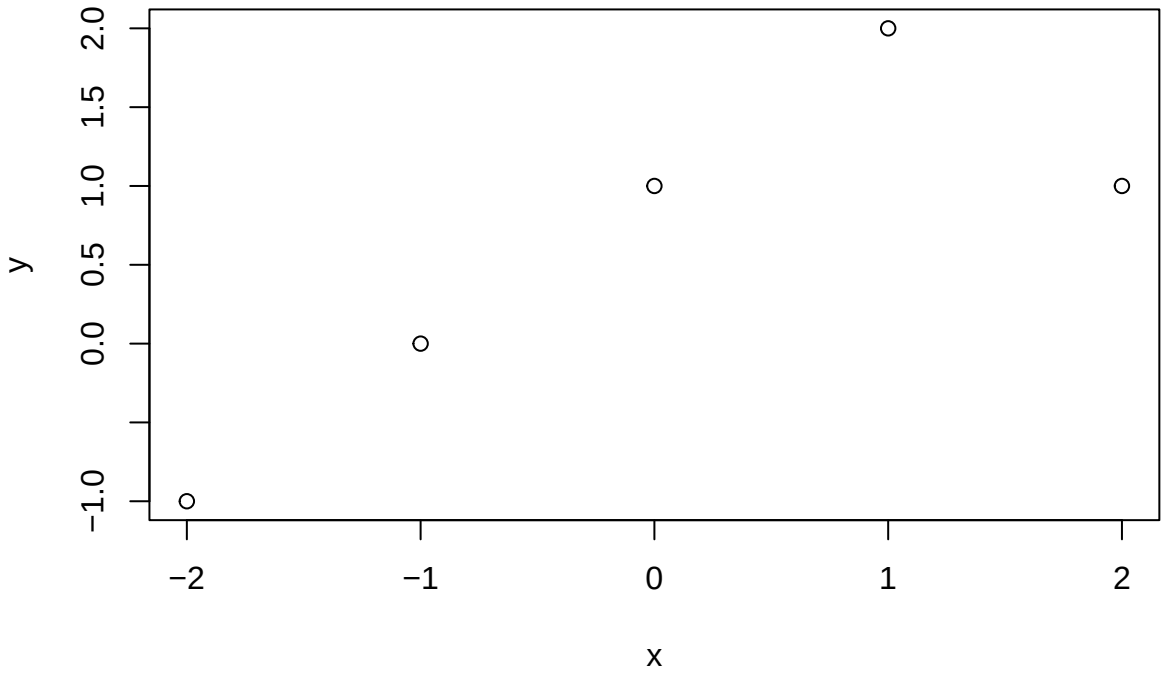
The bootstrap of the estimates of the standard errors are listed above in the summary.

6d). Overall, the estimate standard errors obtained by the two differing methods are pretty close and there wouldn't be a huge difference in using one method over the other, but I personally would chose to use the bootstrap in this case.

Part II

Question 3 Chapter 7

```
x = -2:2
y = 1 + x + -2 * (x-1)^2 * I(x>1)
plot(x, y)
```



Question 9 Chapter 7

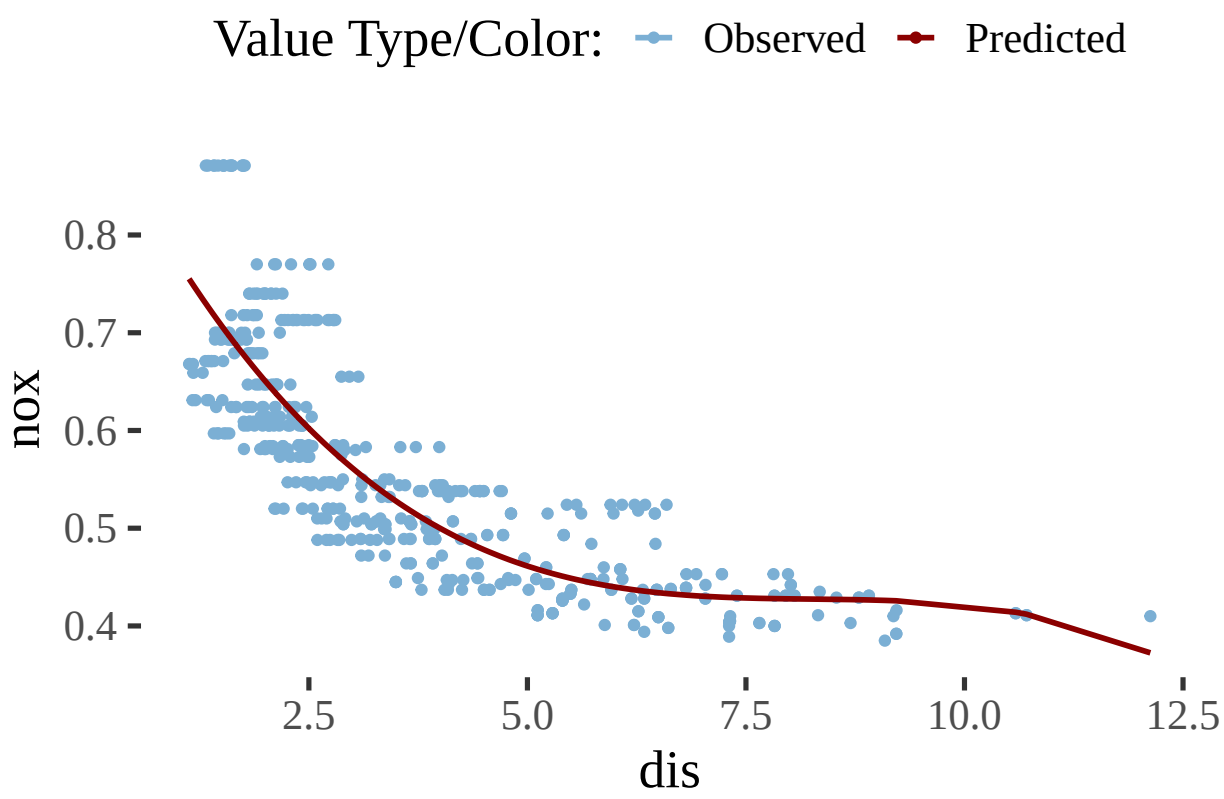
9a).

```
data("Boston")
set.seed(1)
theme_set(theme_tufte(base_size = 20) + theme(legend.position = 'top'))
data('Boston')

model <- lm(nox ~ poly(dis, 3), data = Boston)
tidy(model) %>%
  kable(digits = 3)
```

term	estimate	std.error	statistic	p.value
(Intercept)	0.555	0.003	201.021	0
poly(dis, 3)1	-2.003	0.062	-32.271	0
poly(dis, 3)2	0.856	0.062	13.796	0
poly(dis, 3)3	-0.318	0.062	-5.124	0

```
Boston %>%
  mutate(pred = predict(model, Boston)) %>%
  ggplot() +
  geom_point(aes(dis, nox, col = '1')) +
```

```
  geom_line(aes(dis, pred, col = '2'), size = 1) +
  scale_color_manual(name = 'Value Type/Color:',
                     labels = c('Observed', 'Predicted'),
                     values = c('#7BAFD4', '#8B0000'))
```



Each power of the dis coefficient is found to be statistically significant according to the model. The plot also seems to describe the data without overfitting.

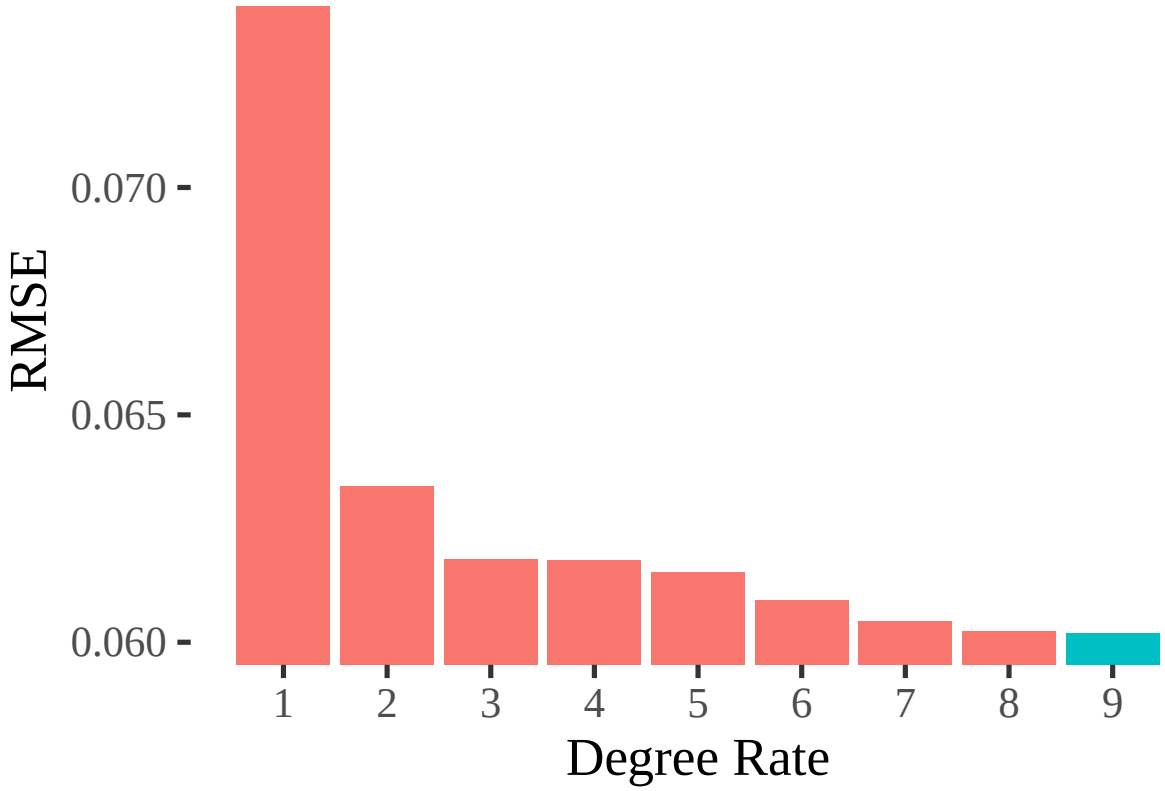
9b).

```
errors <- list()
models <- list()
pred_df <- data_frame(V1 = 1:506)
```

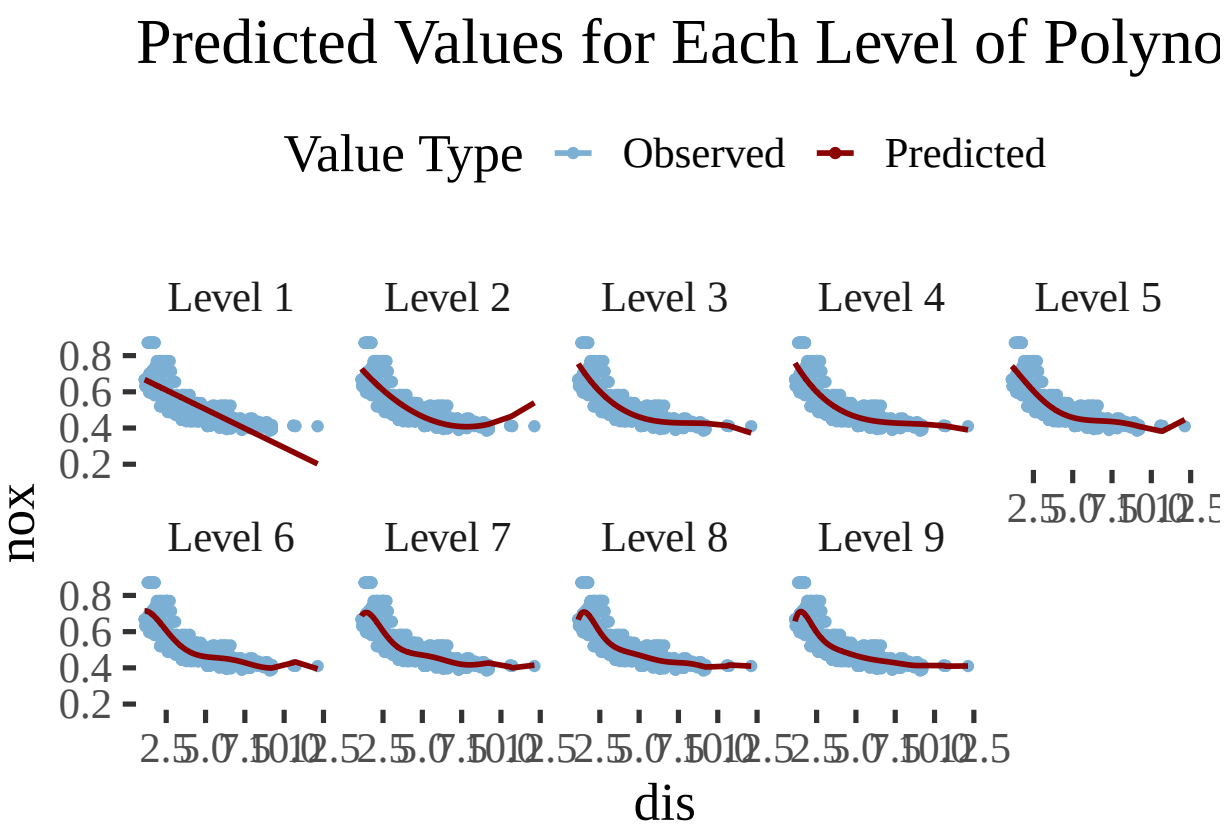
```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## Please use `tibble()` instead.
```

```
for (i in 1:9) {
  models[[i]] <- lm(nox ~ poly(dis, i), data = Boston)
  preds <- predict(models[[i]])
  pred_df[[i]] <- preds
  errors[[i]] <- sqrt(mean((Boston$nox - preds)^2))
}
errors <- unlist(errors)
names(pred_df) <- paste('Level', 1:9)
data_frame(RMSE = errors) %>%
```

```
  mutate(Poly = row_number()) %>%
  ggplot(aes(Poly, RMSE, fill = Poly == which.min(errors))) +
  geom_col() +
  guides(fill = "none") +
  scale_x_continuous(breaks = 1:9) +
  coord_cartesian(ylim = c(min(errors), max(errors))) +
  labs(x = 'Degree Rate')
```




```
Boston %>%
  cbind(pred_df) %>%
  gather(Polynomial, prediction, -(1:14)) %>%
  mutate(Polynomial = factor(Polynomial,
                             levels = unique(as.character(Polynomial)))) %>%
  ggplot() +
  ggtitle('Predicted Values for Each Level of Polynomial') +
  geom_point(aes(dis, nox, col = '1')) +
  geom_line(aes(dis, prediction, col = '2'), size = 1) +
  scale_color_manual(name = 'Value Type',
                     labels = c('Observed', 'Predicted'),
                     values = c('#7BAFD4', '#8B0000')) +
  facet_wrap(~ Polynomial, nrow = 2)
```

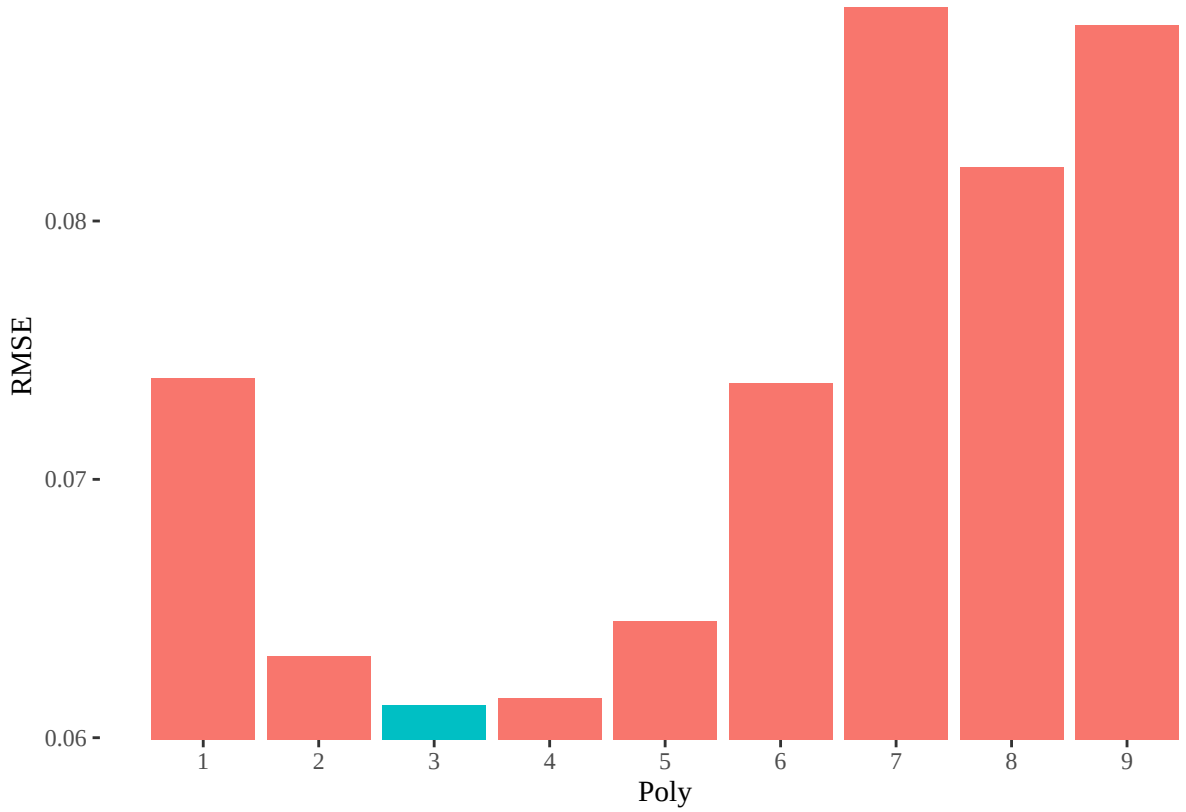


From this we see that the model with the highest polynomial degree has the lowest RSS when fitted and tested on the same data.

9c).

```
errors <- list()
folds <- sample(1:10, 506, replace = TRUE)
errors <- matrix(NA, 10, 9)
for (k in 1:10) {
  for (i in 1:9) {
    model <- lm(nox ~ poly(dis, i), data = Boston[folds != k,])
    pred <- predict(model, Boston[folds == k,])
    errors[k, i] <- sqrt(mean((Boston$nox[folds == k] - pred)^2))
  }
}
errors <- apply(errors, 2, mean)
data_frame(RMSE = errors) %>%
  mutate(Poly = row_number()) %>%
  ggplot(aes(Poly, RMSE, fill = Poly == which.min(errors))) +
  geom_col() + theme_tufte() + guides(fill = FALSE) +
  scale_x_continuous(breaks = 1:9) +
  coord_cartesian(ylim = range(errors))
```

```
## Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> =
## "none")` instead.
```



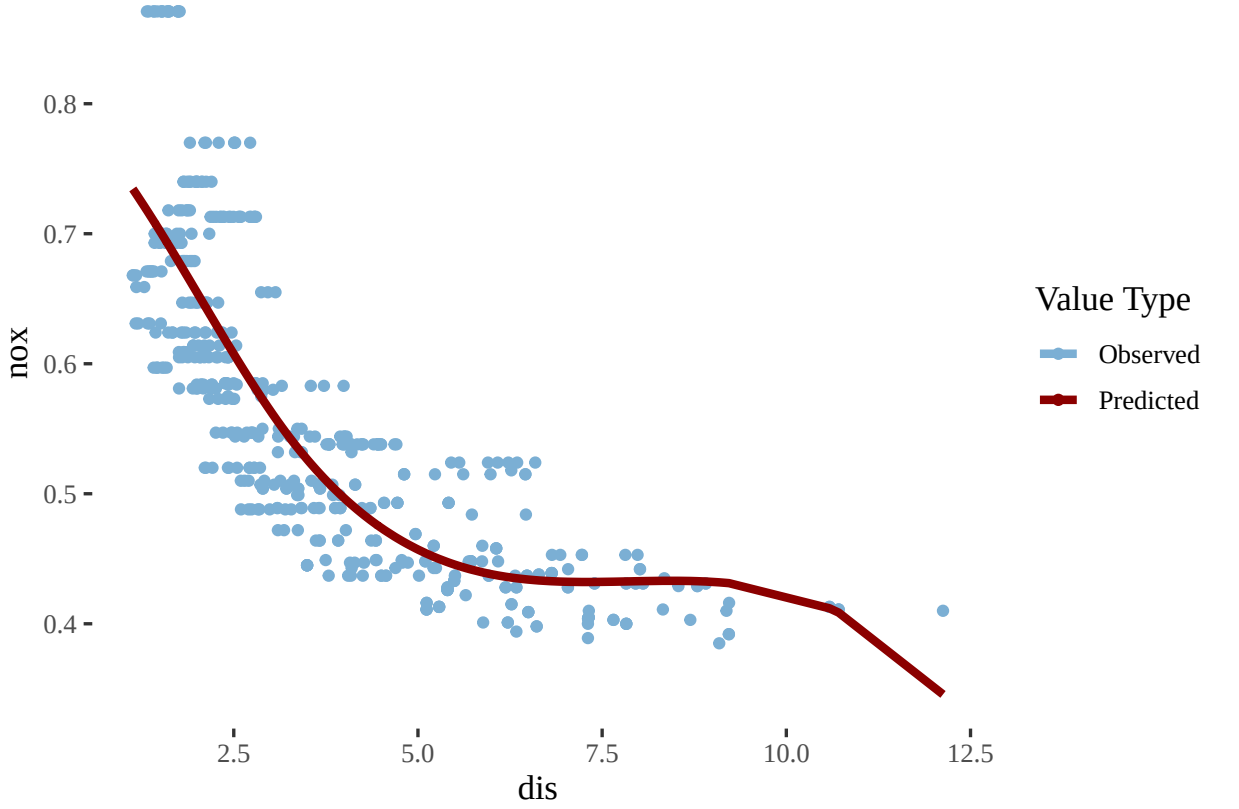
The model with a polynomial degree of 4 is chosen when tested on out-of-sample data. This is the highest polynomial degree that does not show signs of overfitting like 5 through 9 do.

9d).

```
model <- lm(nox ~ bs(dis, df = 4), data = Boston)
kable(tidy(model), digits = 3)
```

term	estimate	std.error	statistic	p.value
(Intercept)	0.734	0.015	50.306	0.000
bs(dis, df = 4)1	-0.058	0.022	-2.658	0.008
bs(dis, df = 4)2	-0.464	0.024	-19.596	0.000
bs(dis, df = 4)3	-0.200	0.043	-4.634	0.000
bs(dis, df = 4)4	-0.389	0.046	-8.544	0.000

```
Boston %>%
  mutate(pred = predict(model)) %>%
  ggplot() +
  geom_point(aes(dis, nox, col = '1')) +
  geom_line(aes(dis, pred, col = '2'), size = 1.5) +
  scale_color_manual(name = 'Value Type',
                     labels = c('Observed', 'Predicted'),
                     values = c('#7BAFD4', '#8B0000')) +
  theme_tufte(base_size = 13)
```



This model finds all the different bases to be statistically significant and the pred line seems to fit data well without overfitting.

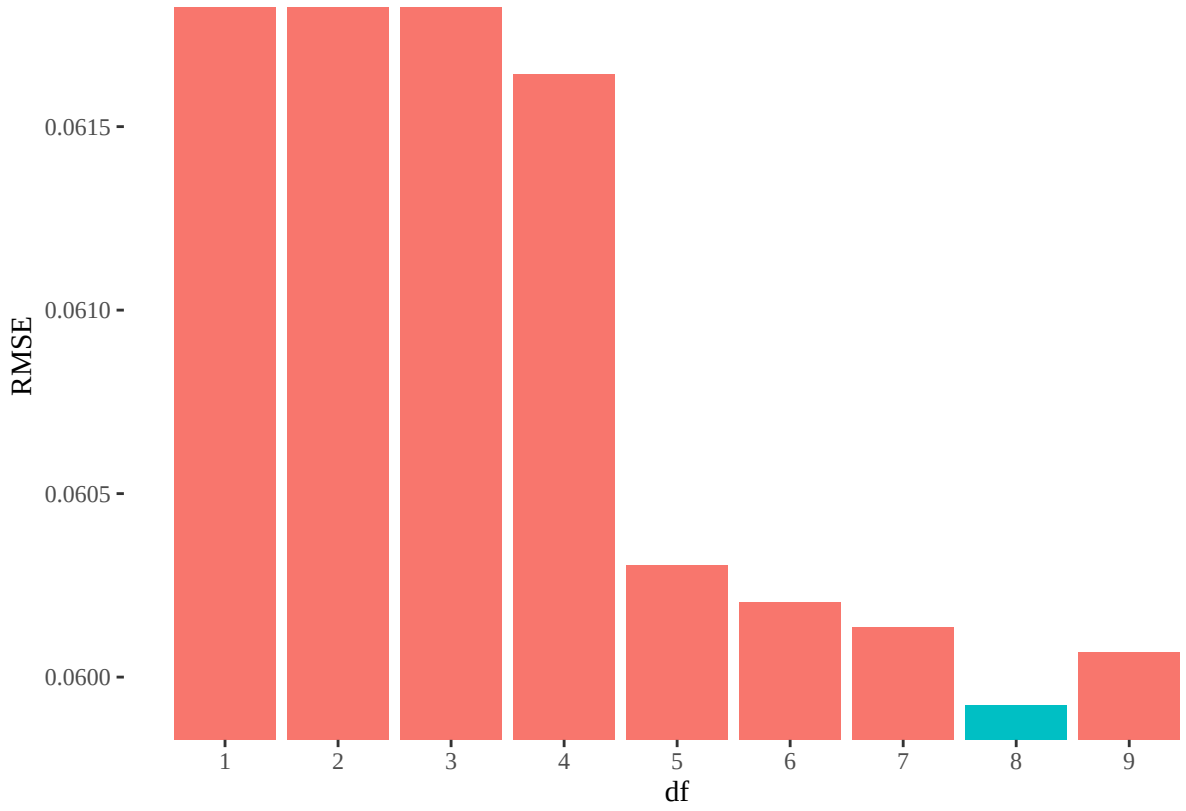
9e).

```
errors <- list()
models <- list()
pred_df <- data_frame(V1 = 1:506)
for (i in 1:9) {
  models[[i]] <- lm(nox ~ bs(dis, df = i), data = Boston)
  preds <- predict(models[[i]])
  pred_df[[i]] <- preds
  errors[[i]] <- sqrt(mean((Boston$nox - preds)^2))
}
```

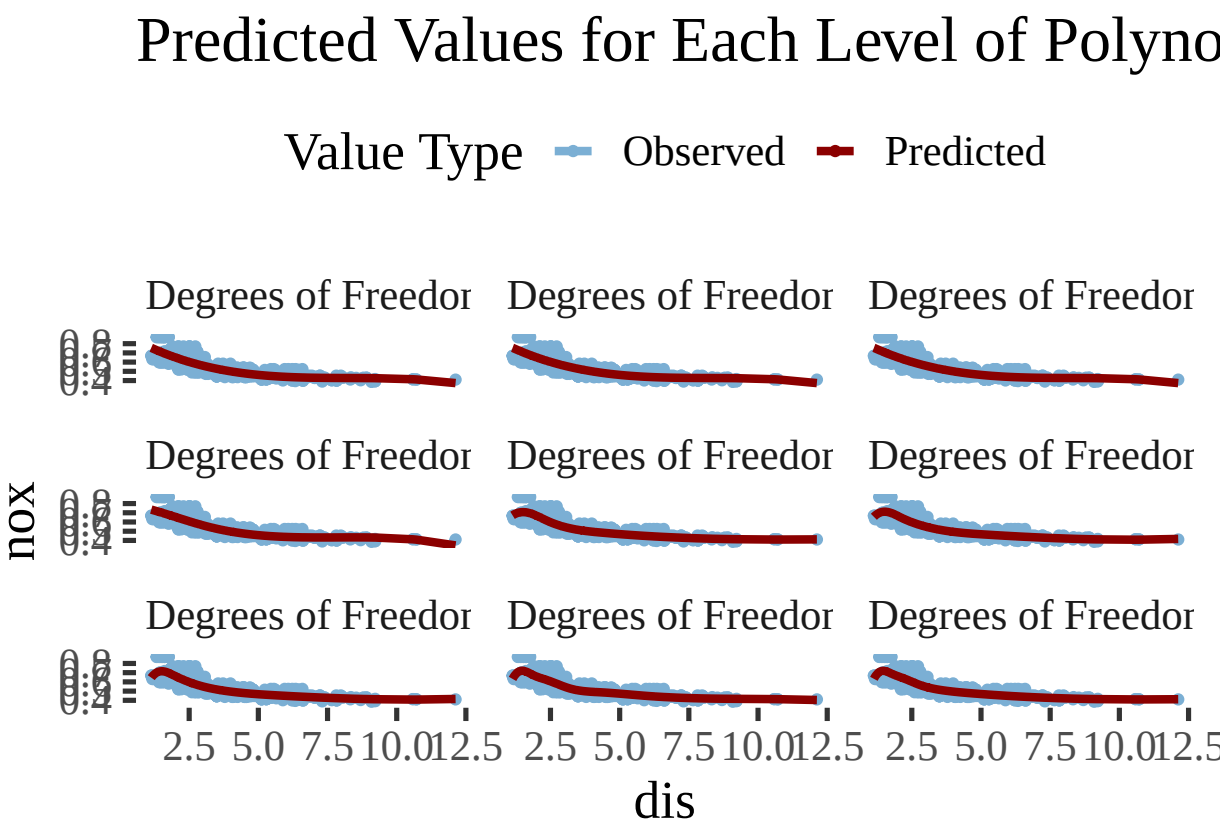
```
## Warning in bs(dis, df = i): 'df' was too small; have used 3

## Warning in bs(dis, df = i): 'df' was too small; have used 3
```

```
names(pred_df) <- paste(1:9, 'Degrees of Freedom')
data_frame(RMSE = unlist(errors)) %>%
  mutate(df = row_number()) %>%
  ggplot(aes(df, RMSE, fill = df == which.min(errors))) +
  geom_col() + guides(fill = FALSE) + theme_tufte() +
  scale_x_continuous(breaks = 1:9) +
  coord_cartesian(ylim = range(errors))
```

```
## Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> =
## "none")` instead.
```




```
Boston %>%
  cbind(pred_df) %>%
  gather(df, prediction, -(1:14)) %>%
  mutate(df = factor(df, levels = unique(as.character(df)))) %>%
  ggplot() + ggtitle('Predicted Values for Each Level of Polynomial') +
  geom_point(aes(dis, nox, col = '1')) +
  geom_line(aes(dis, prediction, col = '2'), size = 1.5) +
  scale_color_manual(name = 'Value Type',
                     labels = c('Observed', 'Predicted'),
                     values = c('#7BAFD4', '#8B0000')) +
  facet_wrap(~ df, nrow = 3)
```



When trained and tested on the same data, the higher complexity models are deemed the best as shown on the plots above.

9f).

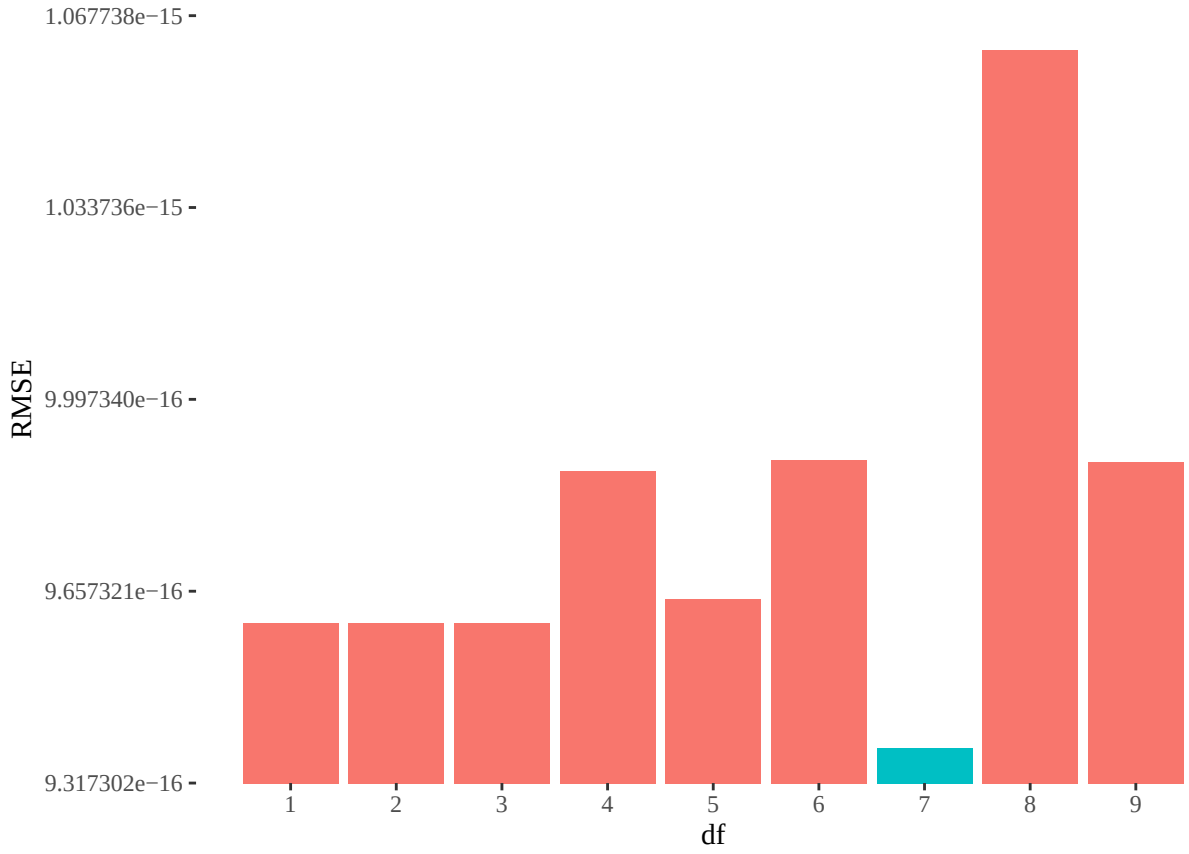
```
folds <- sample(1:10, size = 506, replace = TRUE)
errors <- matrix(NA, 10, 9)
models <- list()
for (k in 1:10) {
  for (i in 1:9) {
    models[[i]] <- lm(nox ~ bs(nox, df = i), data = Boston[folds != k,])
    pred <- predict(models[[i]], Boston[folds == k,])
    errors[k, i] <- sqrt(mean((Boston$nox[folds == k] - pred)^2))
  }
}
```

```
## Warning in bs(nox, df = i): 'df' was too small; have used 3

## Warning in bs(nox, df = i): 'df' was too small; have used 3

## Warning in bs(nox, df = i): 'df' was too small; have used 3

## Warning in bs(nox, df = i): 'df' was too small; have used 3

## Warning in bs(nox, df = i): 'df' was too small; have used 3

## Warning in bs(nox, df = i): 'df' was too small; have used 3
```

```
## Warning in bs(nox, df = i): 'df' was too small; have used 3

## Warning in bs(nox, degree = 3L, knots = numeric(0), Boundary.knots = c(0.389, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(nox, df = i): 'df' was too small; have used 3

## Warning in bs(nox, degree = 3L, knots = numeric(0), Boundary.knots = c(0.389, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(nox, degree = 3L, knots = numeric(0), Boundary.knots = c(0.389, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(nox, degree = 3L, knots = c(`50%` = 0.538), Boundary.knots =
## c(0.389, : some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(nox, degree = 3L, knots = c(`33.33333%` = 0.489, `66.66667%` =
## 0.597: some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(nox, degree = 3L, knots = c(`25%` = 0.449, `50%` = 0.538, : some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(nox, degree = 3L, knots = c(`20%` = 0.442, `40%` = 0.507, : some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(nox, degree = 3L, knots = c(`16.66667%` = 0.437, `33.33333%` =
## 0.489, : some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(nox, degree = 3L, knots = c(`14.28571%` = 0.431, `28.57143%` =
## 0.464, : some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(nox, df = i): 'df' was too small; have used 3

## Warning in bs(nox, df = i): 'df' was too small; have used 3

## Warning in bs(nox, df = i): 'df' was too small; have used 3

## Warning in bs(nox, df = i): 'df' was too small; have used 3

## Warning in bs(nox, df = i): 'df' was too small; have used 3

## Warning in bs(nox, df = i): 'df' was too small; have used 3

## Warning in bs(nox, df = i): 'df' was too small; have used 3

## Warning in bs(nox, df = i): 'df' was too small; have used 3

## Warning in bs(nox, df = i): 'df' was too small; have used 3

## Warning in bs(nox, df = i): 'df' was too small; have used 3

## Warning in bs(nox, df = i): 'df' was too small; have used 3

## Warning in bs(nox, df = i): 'df' was too small; have used 3
```

```
errors <- apply(errors, 2, mean)
data_frame(RMSE = errors) %>%
  mutate(df = row_number()) %>%
  ggplot(aes(df, RMSE, fill = df == which.min(errors))) +
  geom_col() + theme_tufte() + guides(fill = FALSE) +
  scale_x_continuous(breaks = 1:9) +
  coord_cartesian(ylim = range(errors))
```

```
## Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> =
## "none")` instead.
```



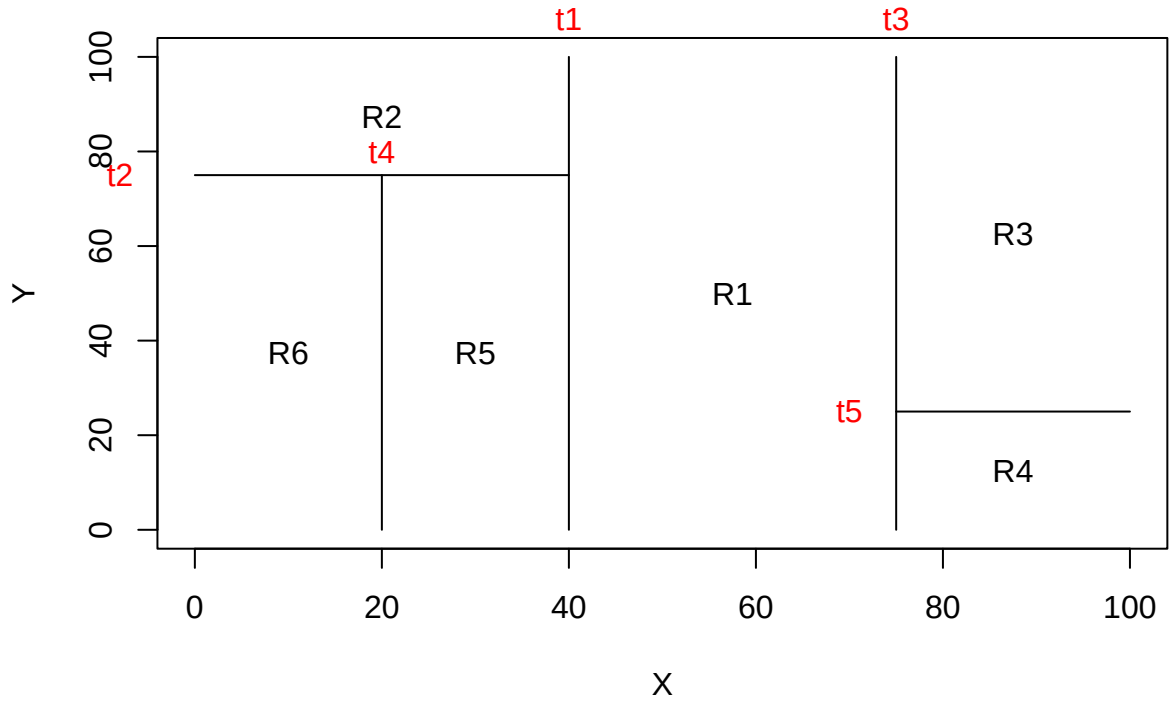
These were validated on out-of-sample data. Due to this, a simpler model is chosen, with df of 4. This is similar to polynomial validation, in that this is the most complex model that does not begin to show signs of overfitting like 5 through 9 do.

Part III

Question 1 Chapter 8

```
par(xpd=NA)
plot(NA, NA, type="n", xlim=c(0,100), ylim=c(0,100), xlab="X", ylab="Y")
# t1: x = 40; (40, 0) (40, 100)
lines(x=c(40,40),y=c(0,100))
```

```
text(x=40, y=108, labels=c("t1"), col="red")
# t2: y = 75; (0, 75) (40, 75)
lines(x=c(0,40), y=c(75,75))
text(x=-8, y=75, labels=c("t2"), col="red")
# t3: x = 75; (75,0) (75, 100)
lines(x=c(75,75),y=c(0,100))
text(x=75, y=108, labels=c("t3"), col="red")
# t4: x = 20; (20,0) (20, 75)
lines(x=c(20,20),y=c(0,75))
text(x=20, y=80, labels=c("t4"), col="red")
# t5: y=25; (75,25) (100,25)
lines(x=c(75,100),y=c(25,25))
text(x=70, y=25, labels=c("t5"), col="red")
text(x=(40+75)/2, y=50, labels=c("R1"))
text(x=20, y=(100+75)/2, labels=c("R2"))
text(x=(75+100)/2, y=(100+25)/2, labels=c("R3"))
text(x=(75+100)/2, y=25/2, labels=c("R4"))
text(x=30, y=75/2, labels=c("R5"))
text(x=10, y=75/2, labels=c("R6"))
```




```
        "[  X<40 ]
        |       |
    [Y<75]     [X<75]
    |   |      |    |
 [X<20] R2     R1   [Y<25]
```

```
 |    |            |    |
 R6   R5           R4   R3"
```

```
## [1] "[  X<40 ] \n        |       |\n    [Y<75]    [X<75]\n    |   |     |    |\n [X<20] R2    R1    [
```

Question 5 Chapter 8

In the majority approach, the final classification is red since the number of red predictions is greater than the number of green predictions. In the second approach, the final classification is green since the average of the probabilities is less than the threshold of 50%.

Question 10 Chapter 8

10a).

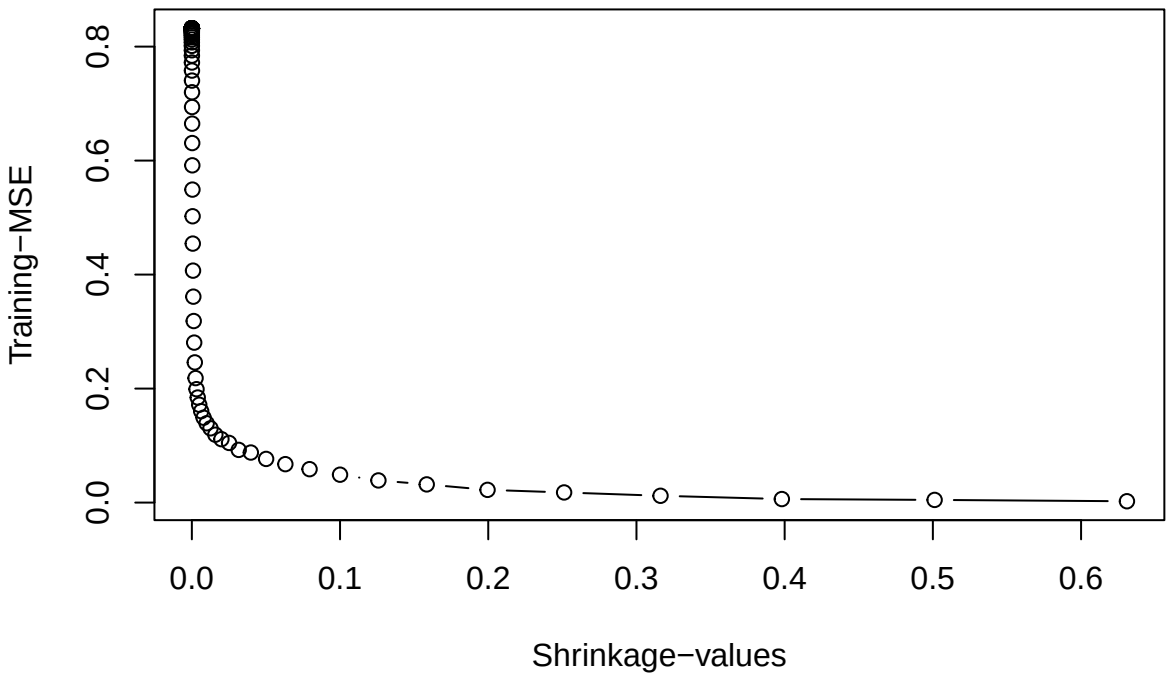
```
data("Hitters")
Hitters = na.omit(Hitters)
Hitters$Salary = log(Hitters$Salary)
```

10b).

```
train = 1:200
hitters.train = Hitters[train,]
hitters.test = Hitters[-train,]
```

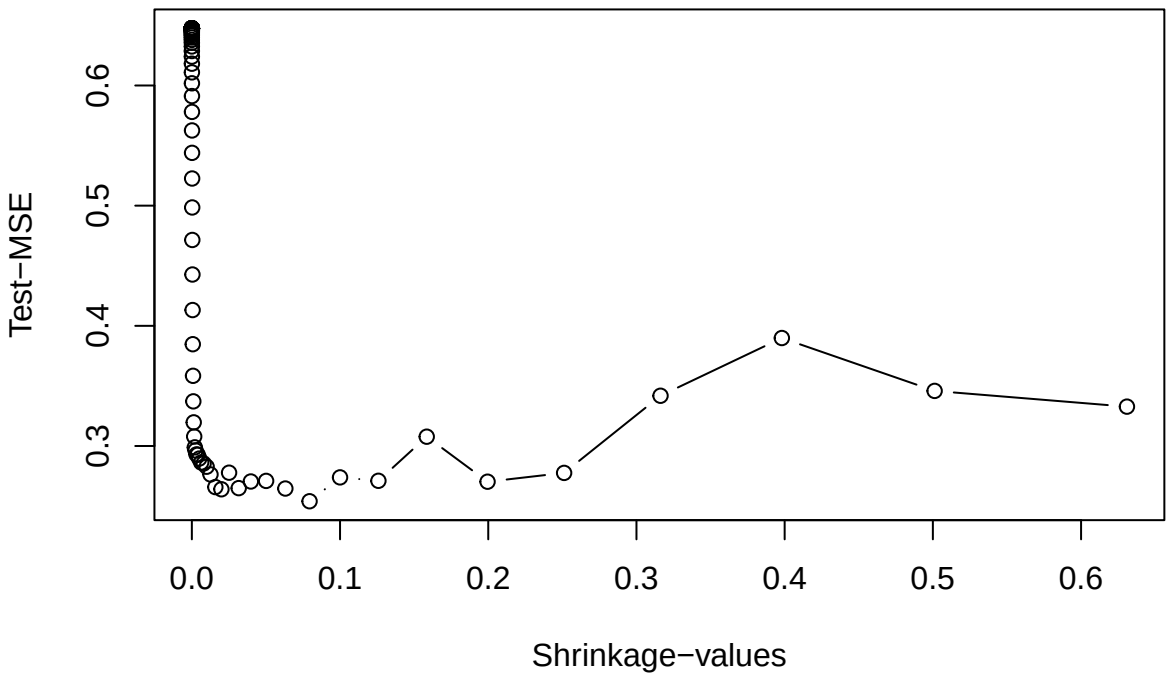
10c).

```
set.seed(1)
pows = seq(-10, -0.2, by = 0.1)
lambdas = 10^pows
train.err = rep(NA, length(lambdas))
for (i in 1:length(lambdas)) {
  boost.hitters = gbm(Salary ~ ., data = hitters.train, distribution = "gaussian", n.trees = 1000, shri
  pred.train = predict(boost.hitters, hitters.train, n.trees = 1000)
  train.err[i] = mean((pred.train - hitters.train$Salary)^2)
}
plot(lambdas, train.err, type = "b", xlab = "Shrinkage-values", ylab = "Training-MSE")
```



10d).

```
set.seed(1)
test.err <- rep(NA, length(lambdas))
for (i in 1:length(lambdas)) {
  boost.hitters = gbm(Salary ~ ., data = hitters.train, distribution = "gaussian", n.trees = 1000, shri
  yhat = predict(boost.hitters, hitters.test, n.trees = 1000)
  test.err[i] = mean((yhat - hitters.test$Salary)^2)
}
plot(lambdas, test.err, type = "b", xlab = "Shrinkage-values", ylab = "Test-MSE")
```




```
min(test.err)
```

```
## [1] 0.2540265
```

```
lambdas[which.min(test.err)]
```

```
## [1] 0.07943282
```

10e).

```
fitFirst = lm(Salary ~ ., data = hitters.train)
predFirst = predict(fitFirst, hitters.test)
mean((predFirst - hitters.test$Salary)^2)
```

```
## [1] 0.4917959
```

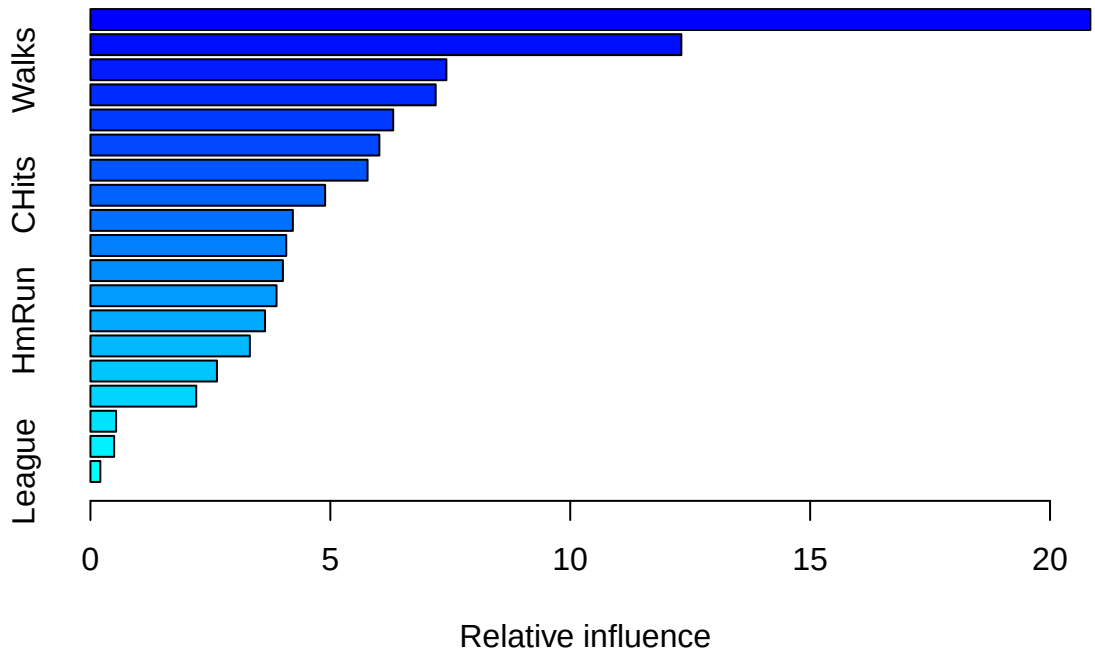
```
x = model.matrix(Salary ~ ., data = hitters.train)
x.test = model.matrix(Salary ~ ., data = hitters.test)
y = hitters.train$Salary
fitSnd = glmnet(x, y, alpha = 0)
predSnd = predict(fitSnd, s = 0.01, newx = x.test)
mean((predSnd - hitters.test$Salary)^2)
```

```
## [1] 0.4570283
```

The test MSE for boosting is lower than for linear and ridge regression shown by the data above.

10f).

```
boost.hitters <- gbm(Salary ~ ., data = hitters.train, distribution = "gaussian", n.trees = 1000, shrink
summary(boost.hitters)
```




```
##                var    rel.inf
## CAtBat       CAtBat 20.8404970
## CRBI           CRBI 12.3158959
## Walks         Walks  7.4186037
## PutOuts     PutOuts  7.1958539
## Years         Years  6.3104535
## CWalks       CWalks  6.0221656
## CHmRun       CHmRun  5.7759763
## CHits         CHits  4.8914360
## AtBat         AtBat  4.2187460
## RBI             RBI  4.0812410
## Hits           Hits  4.0117255
## Assists     Assists  3.8786634
## HmRun         HmRun  3.6386178
## CRuns         CRuns  3.3230296
## Errors       Errors  2.6369128
## Runs           Runs  2.2048386
## Division   Division  0.5347342
## NewLeague NewLeague  0.4943540
```

```
## League       League  0.2062551
```

The variable 'CAtBat' is the most important predictor in the boosted model.

10g).

```
set.seed(1)
bag.hitters <- randomForest(Salary ~ ., data = hitters.train, mtry = 19, ntree = 500)
yhat.bag <- predict(bag.hitters, newdata = hitters.test)
mean((yhat.bag - hitters.test$Salary)^2)
```

```
## [1] 0.2299324
```

The test set MSE for this approach is 0.2299324 which is slightly lower than the test MSE for boosting.