ISLR | Chapter 4 Exercises

Marshall McQuillen

7/5/2018

Conceptual

1

$$f(\alpha) = Var(\alpha X + (1 - \alpha)Y)$$

Using the statistical property that $Var(X + Y) = Var(X) + Var(Y) + 2Cov(X, Y)$, the above equation can be rewritten as:

$$f(\alpha) = Var(\alpha X) + Var((1 - \alpha)Y) + 2Cov(\alpha X, (1 - \alpha)Y)$$

Then, using the statistical property that $Var(cX) = c^2Var(X)$ and $Cov(aX, bY) = abCov(X, Y)$, the equation can once again be rewritten as:

$$f(\alpha) = \alpha^2 Var(X) + (1 - \alpha)^2 Var(Y) + 2\alpha(1 - \alpha)Cov(X, Y)$$

Multiplying the $\alpha(1 - \alpha)$ comes out to:

$$f(\alpha) = \alpha^2 Var(X) + (1 - \alpha)^2 Var(Y) + 2(\alpha - \alpha^2)Cov(X, Y)$$

By then taking the partial derivative of $f(\alpha)$ with respect to α, the slope of the function at a given alpha can be obtained:

$$\frac{\partial f(\alpha)}{\partial \alpha} = 2\alpha\sigma_X^2 + 2(1 - \alpha)(-1)\sigma_Y^2 + 2(1 - 2\alpha)\sigma_{XY}$$

Divide by 2:

$$\frac{\partial f(\alpha)}{\partial \alpha} = \alpha\sigma_X^2 + (-1 + \alpha)\sigma_Y^2 + (1 - 2\alpha)\sigma_{XY}$$

Expand the second and third terms in the equation:

$$\frac{\partial f(\alpha)}{\partial \alpha} = \alpha\sigma_X^2 + -\sigma_Y^2 + \alpha\sigma_Y^2 + \sigma_{XY} - 2\alpha\sigma_{XY}$$

Factor α out of all possible terms:

$$\frac{\partial f(\alpha)}{\partial \alpha} = \alpha(\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}) - \sigma_Y^2 + \sigma_{XY}$$

Divide each term by $(\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY})$:

$$\frac{\partial f(\alpha)}{\partial \alpha} = \alpha - \frac{\sigma_Y^2 + \sigma_{XY}}{(\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY})}$$

Since the goal is to minimize the equation, setting the partial derivative to zero will return an equation that is a minimum.

$$0 \;=\; \alpha \;-\; \frac{\sigma_Y^2 \;+\; \sigma_{XY}}{(\sigma_X^2 \;+\; \sigma_Y^2 \;-\; 2\sigma_{XY})}$$

Subtract α

$$-\alpha \;= -\; \frac{\sigma_Y^2 \;+\; \sigma_{XY}}{(\sigma_X^2 \;+\; \sigma_Y^2 \;-\; 2\sigma_{XY})}$$

Multiply by -1:

$$\alpha \;=\; \frac{\sigma_Y^2 \;-\; \sigma_{XY}}{\sigma_X^2 \;+\; \sigma_Y^2 \;-\; 2\sigma_{XY}}$$

2

- **A**. Since a bootstrapped sample contains N observations of the original sample of the population, each sample being chosen at random with replacement, the probability that the first observation in a bootstrapped sample is *not* the jth observation is $\frac{n\;-\;1}{n}$.
- **B**. The probability that the second bootstrap observation is *not* the jth observation is $\left(\frac{n\;-\;1}{n}\right)^2$.
- **C**. Since a boostrapped sample contains N observations, the probability that the jth observation (x_j) is *not* in the bootstapped sample (S_b) is:

$$P(x_j \; not \; in \; S_b) \;=\; \left(\frac{n \;-\; 1}{n}\right)^n$$

  Which can be simplified to:

$$P(x_j \; not \; in \; S_b) \;=\; \left(1 \;-\; \frac{1}{n}\right)^n$$

- **D**. Since the probability that the jth observation is *not* in the boostrap sample is $\left(1 \;-\; \frac{1}{n}\right)^n$, the probability that the jth observation *is* in the bootstrap sample would be the complement, $1 \;-\; \left(1 \;-\; \frac{1}{n}\right)^n$. When $n \;=\; 5$, this comes out to $1 \;-\; \left(1 \;-\; \frac{1}{5}\right)^5 \;=\; 0.67232 \;=\; 67.23\%$
- **E**.

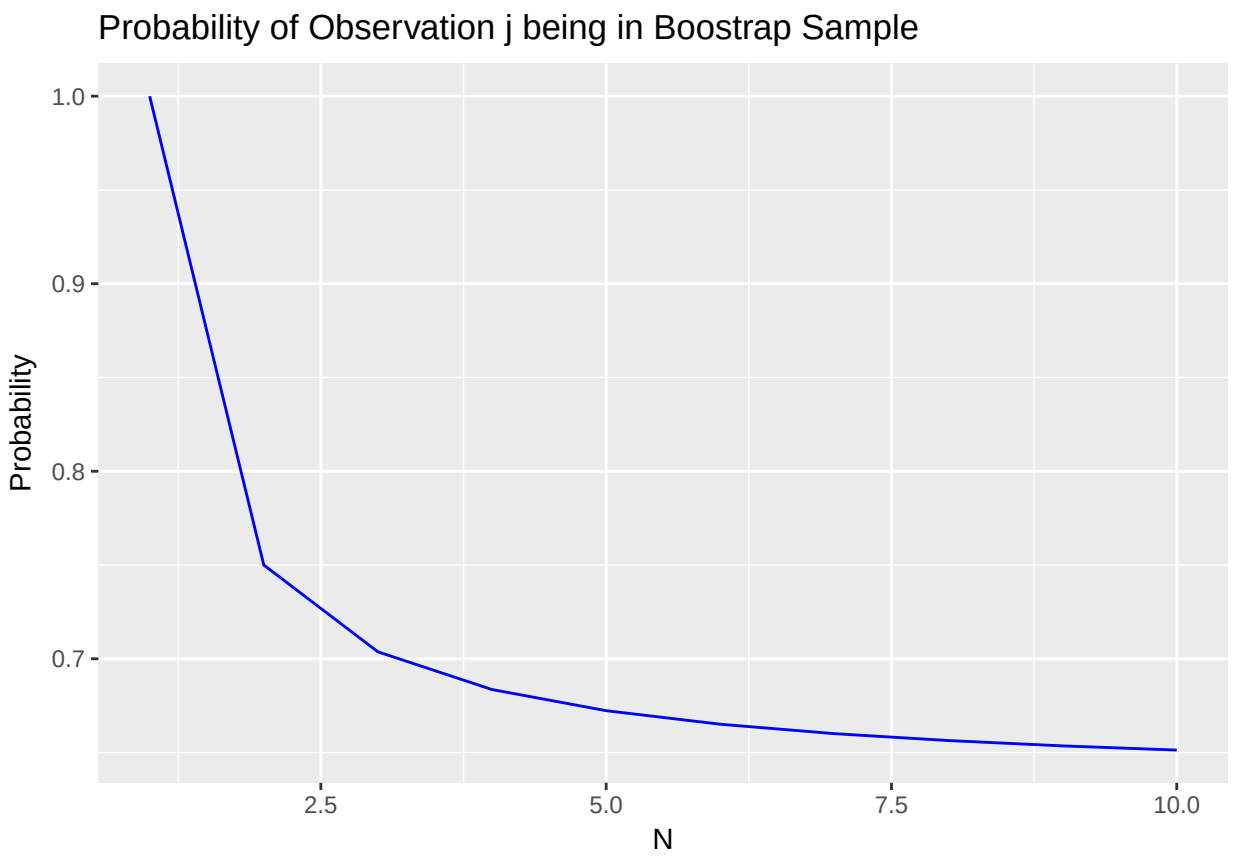
$$1 \;-\; \left(1 \;-\; \frac{1}{100}\right)^{100} \;=\; 0.6339677 \;=\; 63.40\%$$

- **F**.

$$1 \;-\; \left(1 \;-\; \frac{1}{100}\right)^{100} \;=\; 0.632139 \;=\; 63.21\%$$

- **G**. It is clear that as N increases the probability that the jth observation is in the bootstrap sample asymptotically approaches 0.632. The below plot illustrates this phenomenon (only displaying 1 to 10 for illustration purposes)

```
library(ggplot2)
x <- 1:100000
y <- 1 - (1 -(1/x))^x
df <- data.frame(x, y)
display_df <- df[1:10,]
ggplot(display_df, aes(x = x, y = y)) +
    geom_line(color = 'blue') +
    labs(x = "N", y = "Probability",
         title = "Probability of Observation j being in Boostrap Sample")
```



- **H**. The below code is showing mathematically what the plot above shows; that the limit of the function $1 - \left(1 - \frac{1}{x}\right)^x$ as x approaches infinity is 0.632.

```
store <- rep(NA, 10000)
for (i in 1:10000) {
    store[i] <- sum(sample(1:100, replace = TRUE)==4) > 0
}
mean(store)
```

```
## [1] 0.6342
```

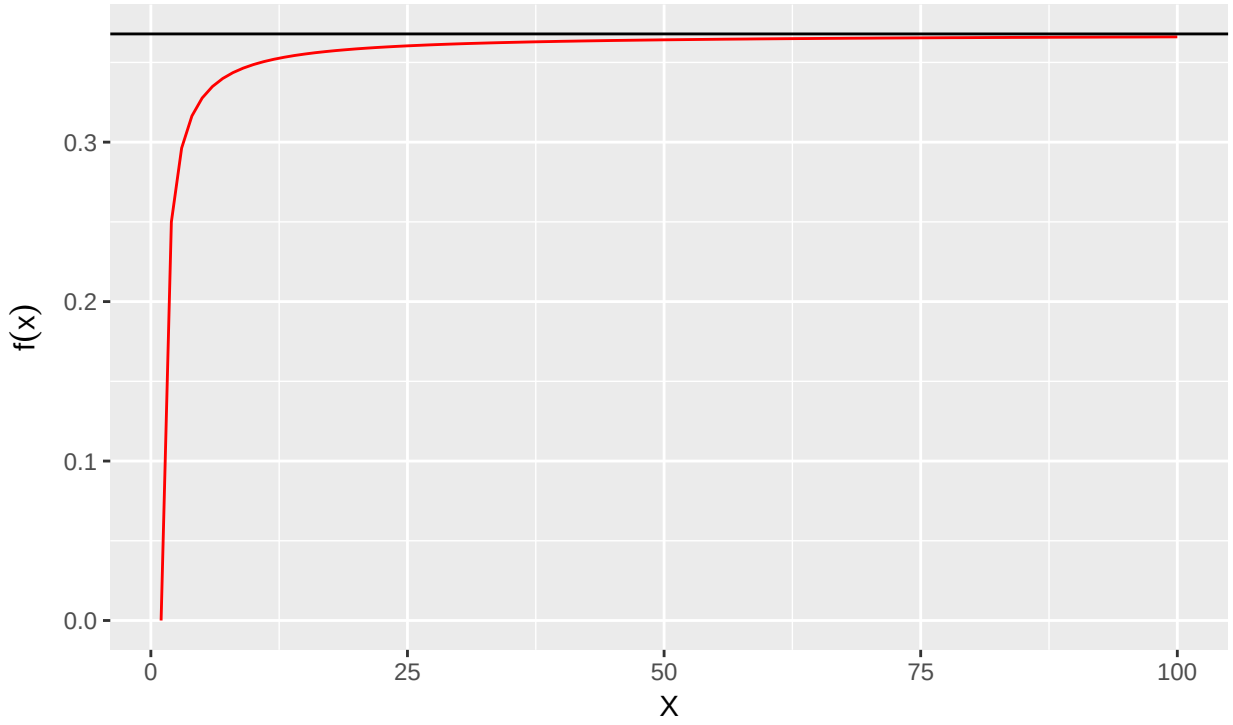
This can be written as:

$$\lim_{x \to \infty} \left(1 - \left(1 - \frac{1}{x}\right)^x\right) = 0.632$$

However, the inner part of that equation, $\lim_{x \to \infty} \left(1 - \frac{1}{x}\right)^x$, simplifies to $\frac{1}{\epsilon}$, proven by plot below:

```r
x <- 1:100
y <- (1 - (1/x))^x
asymptote <- rep(1/exp(1), 100)
df <- data.frame(x, y, asymptote)
ggplot(df, aes(x = x, y = y)) +
    geom_line(color = 'red') +
    geom_hline(aes(yintercept = asymptote)) +
    labs(x = "X", y = expression(f(x))) +
    ggtitle(expression(lim((1 - over(1, "x"))^"x", x %->% infinity) == frac(1, e)))
```

$$\lim_{x\to\infty}\left(1-\frac{1}{x}\right)^x=\frac{1}{e}$$



Therefore:

$$\lim_{x\to\infty}\left(1-\left(1-\frac{1}{x}\right)^x\right)=0.632=1-\frac{1}{\epsilon}$$

3

- **A**. K-Fold cross validation is the process of randomly dividing the entire data set into K separate subsets. A statistical model can then be trained on $K-1$ of those subsets, and the final Kth subset is used to test the model on unseen data, returning an estimate of the test error. This is performed K times, each time using a different subset as to estimate the test error. This results in K separate estimates of the testing error, which can be averaged to get the cross validated error estimate.
- **B**.

  i. There are a couple advantages of K-Fold CV over the validation set approach. First, K-Fold CV will return more than one estimate of the testing error, allowing insight into the variance of

the testing error. In addition to this, since the number of observations in the training data set using the validation set approach is less than the number of observations used in the training data set in K-Fold CV, the validation set approach will typically overestimate the testing error. This is due to the fact that a model has a better chance of modeling the true relationship within the data the more observations too which it has access.

ii. LOOCV also has a couple downsides relative to K-Fold CV. First and foremost, since a total of N models are fit to the data, there is a large increase in computation time over K-Fold CV when K is equal to the usual 5 or 10. In addition to this, since there are N total models and each of the N models consists of $N - 1$ observations, *each of the N models will be trained on nearly identical data*. This leads to the CV error estimates being highly correlated, which corresponds to high variance and low bias.

4

Bootstrapping the origin sample would allow an estimate of the standard deviation of a prediction Y for a given predictor X. By randomly drawing N observations from the data set **with replacement**, one can create Z bootstrapped data sets. The model can be fit to each of these bootstapped data sets, and then a prediction Y can recorded for a constant X with each of the models. This will result in Z estimates of the prediction Y. The standard deviation of these predictions can then be computed using the forumula:

$$\sigma_Y = \sqrt{\frac{\sum_i^z (\hat{y}_i - \bar{y})^2}{n - 1}}$$

Applied

5

- **A**.

```
suppressPackageStartupMessages(library(ISLR))
suppressPackageStartupMessages(library(caret))
log_mod <- glm(default ~ income + balance, data = Default, family = 'binomial')
```

- **B**.

```
# i
set.seed(5)
sample_size <- floor(0.5*nrow(Default))
train_idx <- sample(seq_len(nrow(Default)), size = sample_size)
train <- Default[train_idx,]
test <- Default[-train_idx,]

# ii
lm_fit <- glm(default ~ income + balance, data = train, family = 'binomial')

# iii
y_hat <- predict(lm_fit, newdata = test, type = 'response')
preds <- rep('No', nrow(test))
preds[y_hat > 0.5] <- 'Yes'

# iv
```

```
cm <- confusionMatrix(as.factor(preds), as.factor(test$default), 'Yes')
cm$overall[1]
```

```
## Accuracy
##   0.9754
```

- **C**. Testing three different random seeds leads to the model performing pretty well, with the Accuracy floating from 97% - 98% and therefore a validation error rate of 2% - 3%.

```
results <- c()
for (seed in sample(100, 3)) {
    # train/validation split
    set.seed(seed = seed)
    sample_size <- floor(0.5*nrow(Default))
    train_idx <- sample(seq_len(nrow(Default)), size = sample_size)
    train <- Default[train_idx,]
    test <- Default[-train_idx,]

    # logistic regression modeling
    lm_fit <- glm(default ~ income + balance, data = train, family = 'binomial')

    # prediction
    y_hat <- predict(lm_fit, newdata = test, type = 'response')
    preds <- rep('No', nrow(test))
    preds[y_hat > 0.5] <- 'Yes'

    # model evaluation
    cm <- confusionMatrix(as.factor(preds), as.factor(test$default), 'Yes')
    acc <- cm$overall[1]
    results <- c(results, acc)
}
results
```

```
## Accuracy Accuracy Accuracy
##   0.9738   0.9760   0.9732
```

- **D**. It doesn't seem as though adding the one hot encoded *Student* variable adds any predictive power to the model. The validation error of 2.68% is within the range of the error rates with *Student* excluded from the model.

```
# train/validation split
set.seed(seed = 200)
sample_size <- floor(0.5*nrow(Default))
train_idx <- sample(seq_len(nrow(Default)), size = sample_size)
train <- Default[train_idx,]
test <- Default[-train_idx,]

# logistic regression modeling
lm_fit <- glm(default ~ ., data = train, family = 'binomial')

# prediction
y_hat <- predict(lm_fit, newdata = test, type = 'response')
preds <- rep('No', nrow(test))
preds[y_hat > 0.5] <- 'Yes'

# model evaluation
```

```
cm <- confusionMatrix(as.factor(preds), as.factor(test$default), 'Yes')
acc <- cm$overall[1]

# validation error
1 - acc
```

```
## Accuracy
##   0.0268
```

6

- **A**.

```
set.seed(5)
log_mod <- glm(default ~ income + balance, Default, family = 'binomial')
knitr::kable(summary(log_mod)$coef)
```

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-11.5404684	0.4347564	-26.544680	0.00e+00
income	0.0000208	0.0000050	4.174178	2.99e-05
balance	0.0056471	0.0002274	24.836280	0.00e+00

- **B**.

```
boot.fn <- function(data, idx_array) {
        fit <- glm(default ~ income + balance,
                   data = data,
                   family = 'binomial',
                   subset = idx_array)
        return(coef(fit)[2:3])

}
```

- **C**.

```
suppressPackageStartupMessages(library(boot))
estimates <- boot(Default, boot.fn, 100)
estimates
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Default, statistic = boot.fn, R = 100)
##
##
## Bootstrap Statistics :
##         original        bias     std. error
## t1* 2.080898e-05 -6.607158e-07 4.237847e-06
## t2* 5.647103e-03 -2.975715e-05 2.429335e-04
```

- **D**. With 100 bootstrapped data sets fit using the same formula, the coefficient estimates are the exact same and the standard errors for the coefficients are the same to 5th and 3rd significant digit.

7

- **A**.

```
attach(Weekly)
log.fit <- glm(Direction ~ Lag1 + Lag2,
               family = 'binomial')
summary(log.fit)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2, family = "binomial")
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.623  -1.261   1.001   1.083   1.506
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.22122    0.06147   3.599 0.000319 ***
## Lag1        -0.03872    0.02622  -1.477 0.139672
## Lag2         0.06025    0.02655   2.270 0.023232 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1496.2  on 1088  degrees of freedom
## Residual deviance: 1488.2  on 1086  degrees of freedom
## AIC: 1494.2
##
## Number of Fisher Scoring iterations: 4
```

- **B**.

```
log.fit <- glm(Direction ~ Lag1 + Lag2,
               data = Weekly[-1,],
               family = 'binomial')
summary(log.fit)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2, family = "binomial", data = Weekly[-1,
##     ])
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6258  -1.2617   0.9999   1.0819   1.5071
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.22324    0.06150   3.630 0.000283 ***
## Lag1        -0.03843    0.02622  -1.466 0.142683
## Lag2         0.06085    0.02656   2.291 0.021971 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1494.6  on 1087  degrees of freedom
## Residual deviance: 1486.5  on 1085  degrees of freedom
## AIC: 1492.5
##
## Number of Fisher Scoring iterations: 4
```

- **C**. The first observation was *not* predicted correctly; with a predicted probability of 57.14% for Direction going up, this observation would be misclassified.

```
y_hat <- predict(log.fit, newdata = Weekly[1,], type = 'response')
y_hat
```

```
##         1
## 0.5713923
```

- **D**.

```
errors <- c()
for (i in 1:nrow(Weekly)) {
    fit <- glm(Direction ~ Lag1 + Lag2,
               data = Weekly[-i,],
               family = 'binomial')
    y_hat <- predict(fit, newdata = Weekly[i,], type = 'response')
    pred <- ifelse(y_hat > 0.5, 'Up','Down')
    error <- ifelse(pred == Weekly[i, 'Direction'], 0, 1)
    errors <- c(errors, error)
}
```

- **E**. With 490 errors the error rate comes out to 45%, potentially enough to make a profit over time, however I'm not sure I would put my own money on the line.

```
mean(errors)
```

```
## [1] 0.4499541
```

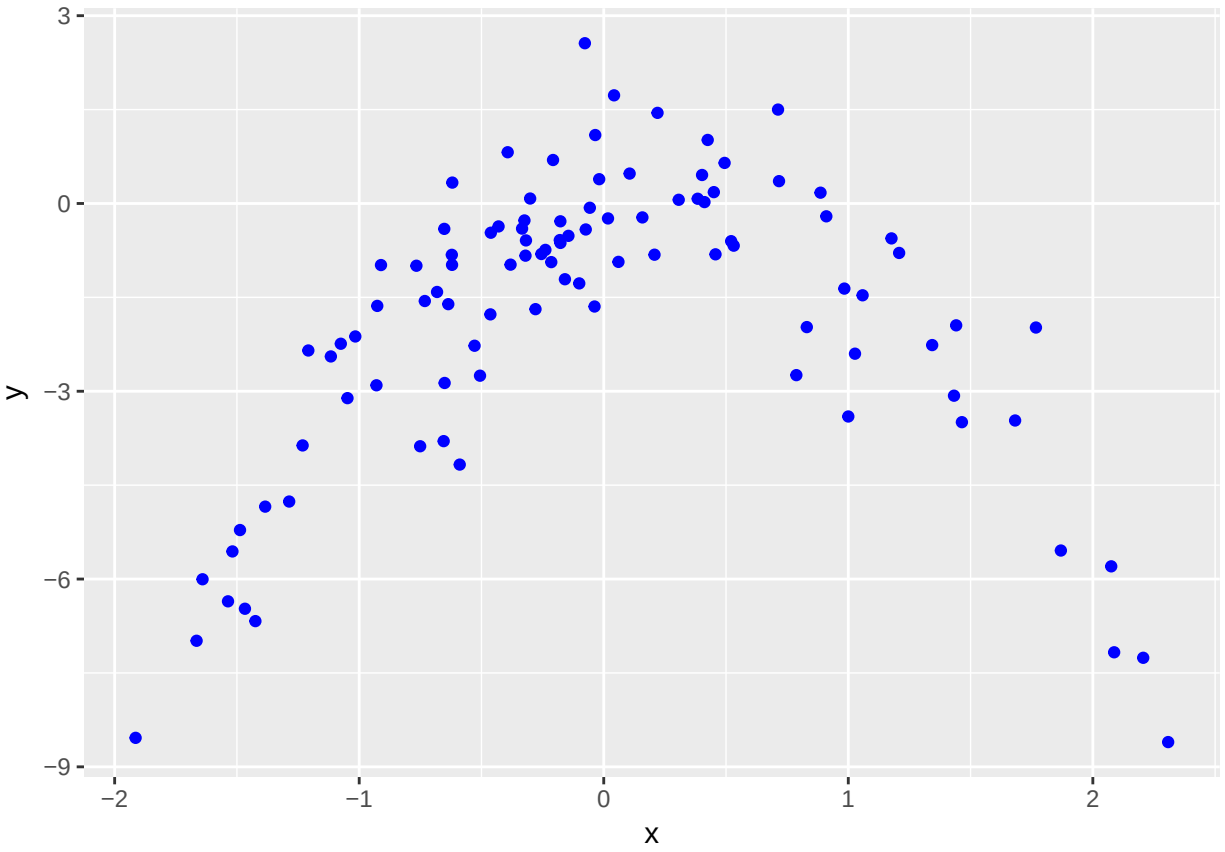
8

- **A**. $N = 100$ and $P = 2$ (two since $2X^2$ is a transformation of X) and the equation is $Y = X - 2X^2 + \epsilon$

```
set.seed(1)
y <- rnorm(100)
x <- rnorm(100)
y <- x - 2*x^2 + rnorm(100)
```

- **B**. As expected, there is a downward facing parabola and most of the noise in the relationship seems to be centered in the interval $-1 \leq X \leq 1$.

```
df <- data.frame(x, y)
ggplot(df, aes(x = x, y = y)) +
    geom_point(color = 'blue')
```



- **C**.

```
set.seed(5)
for (i in 1:4) {
    glm.fit <- glm(y ~ poly(x, i), df, family = 'gaussian')
    cv.error <- cv.glm(df, glm.fit)
    print(cv.error$delta)
}
```

```
## [1] 5.890979 5.888812
## [1] 1.086596 1.086326
## [1] 1.102585 1.102227
## [1] 1.114772 1.114334
```

- **D**. The results are the same for for both random seeds because there is no *randomness* associated with LOOCV; N models are trained on $N-1$ observations and then evaluated on the final observation. All N evaluations are then averaged together to get the final cross validated estimate of the test error. Changing the seed would have an affect only when using K-Fold cross validation where $K \neq N$.

```
set.seed(100)
for (i in 1:4) {
    glm.fit <- glm(y ~ poly(x, i), df, family = 'gaussian')
    cv.error <- cv.glm(df, glm.fit)
    print(cv.error$delta)
}
```

```
## [1] 5.890979 5.888812
## [1] 1.086596 1.086326
## [1] 1.102585 1.102227
```

```
## [1] 1.114772 1.114334
```

- **E**. The best model as determined by the cross validated test estimates is the model with the second degree polynomial. This holds to reason since this model is the closest to the true relationship between X and Y.
- **F**. Looking at the output below, it is clear that the estimate for the second degree polynomial coefficient is highly statistically significant throughout all models. However, as one would expect, the estimates for the third and fourth degree polynomial coefficients are not statistically significant, perfectly aligning with the output of the LOOCV results.

```
set.seed(100)
for (i in 1:4) {
    glm.fit <- glm(y ~ poly(x, i), df, family = 'gaussian')
    print(summary(glm.fit)$coef)
}
```

```
##              Estimate Std. Error    t value     Pr(>|t|)
## (Intercept) -1.827707  0.2362206 -7.7372898 9.181461e-12
## poly(x, i)   2.316401  2.3622062  0.9806091 3.292002e-01
##               Estimate Std. Error   t value     Pr(>|t|)
## (Intercept)  -1.827707  0.1032351 -17.70431 3.804657e-32
## poly(x, i)1   2.316401  1.0323515   2.24381 2.711854e-02
## poly(x, i)2 -21.058587  1.0323515 -20.39866 7.333860e-37
##                Estimate Std. Error     t value     Pr(>|t|)
## (Intercept)  -1.8277074  0.1037248 -17.6207390 7.610579e-32
## poly(x, i)1   2.3164010  1.0372479   2.2332183 2.785714e-02
## poly(x, i)2 -21.0585869  1.0372479 -20.3023667 1.636959e-36
## poly(x, i)3  -0.3048398  1.0372479  -0.2938929 7.694742e-01
##                Estimate Std. Error     t value     Pr(>|t|)
## (Intercept)  -1.8277074  0.1041467 -17.5493533 1.444977e-31
## poly(x, i)1   2.3164010  1.0414671   2.2241711 2.850549e-02
## poly(x, i)2 -21.0585869  1.0414671 -20.2201171 3.457023e-36
## poly(x, i)3  -0.3048398  1.0414671  -0.2927023 7.703881e-01
## poly(x, i)4  -0.4926249  1.0414671  -0.4730105 6.372907e-01
```

9

- **A**.

```
library(MASS)
mu_hat <- mean(Boston$medv)
mu_hat
```

```
## [1] 22.53281
```

- **B**.

```
stand_error <- sd(Boston$medv) / sqrt(nrow(Boston))
stand_error
```

```
## [1] 0.4088611
```

- **C**. 10,000 bootstrapped estimates of the mean of *medv* comes out to equal the calculated mean and the bootstrapped standard error estimate is only off by roughly 0.002.

```
boot.mean <- function(data, idx_array) {
    return(mean(data[idx_array, 'medv']))
```

```
}
set.seed(5)
boot_est <- boot(Boston, boot.mean, 10000)
boot_est
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston, statistic = boot.mean, R = 10000)
##
##
## Bootstrap Statistics :
##     original       bias    std. error
## t1* 22.53281 -0.000891996   0.4100943
```

- **D**. The bootstrapped 95% confidence interval for the mean of *medv* is (21.71262, 23.35299).

```
lower <- boot_est$t0 - 2*sd(boot_est$t)
upper <- boot_est$t0 + 2*sd(boot_est$t)
lower
```

```
## [1] 21.71262
```

```
upper
```

```
## [1] 23.35299
```

- **E**.

```
med_hat <- median(Boston$medv)
med_hat
```

```
## [1] 21.2
```

- **F**. With 10,000 bootstrapped data sets, the estimate for the median comes out to be the same as the measured median and the standard error is only 0.38.

```
boot.med <- function(data, idx_array) {
    return(median(data[idx_array, 'medv']))
}
set.seed(5)
boot_est <- boot(Boston, boot.med, 10000)
boot_est
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston, statistic = boot.med, R = 10000)
##
##
## Bootstrap Statistics :
##     original   bias    std. error
## t1*     21.2 -0.01332   0.3798335
```

- **G**.

```r
perc_10 <- quantile(Boston$medv, 0.1)
perc_10
```

```
##   10%
## 12.75
```

- **H**. The bootstrapped estimate of the 10th percentile of *medv* is the same as the measured 10th percentile using the entire data set, however the standard error is noticably higher than the other two statistics.

```r
boot.perc_10 <- function(data, idx_array) {
    return(quantile(data[idx_array, 'medv'], 0.1))
}
set.seed(5)
boot_est <- boot(Boston, boot.perc_10, 10000)
boot_est
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston, statistic = boot.perc_10, R = 10000)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*    12.75 0.01072   0.5012576
```