Stat 495 - Midterm Practice Questions - Solution

Add 1

A random variable X is a member of an exponential family if its density $f(x|\theta)$ can be written as:

$$f(x|\theta) = a(\theta)b(x)\exp\left[c(\theta)d(x)\right],$$

where $a()$ and $c()$ are functions only of the parameter θ, and $b()$ and $d()$ are functions of x.

Verify that the Binomial distribution is an exponential family. Note, X $Bin(n,p)$ has pmf given by:

$$P(X = x|p) = (n \text{ choose } x)p^x(1-p)^{n-x}.$$

SOLUTION

This is very similar to the Bernoulli example on the chapter 8 handout.

We start from the pmf and do some rearranging.

$$P(X = x|p) = (n \text{ choose } x)p^x(1-p)^{n-x}.$$

$$P(X = x|p) = (1-p)^n \ (n \text{ choose } x)\left(\frac{p}{1-p}\right)^x.$$

We can see the a() and b() parts already, but need to get the last part into an exp to finish.

$$P(X = x|p) = (1-p)^n \ (n \text{ choose } x)\exp\left[x\log\left(\frac{p}{1-p}\right)\right].$$

So, $a(p) = (1-p)^n$, $b(x) = (n \text{ choose } x)$, $c(p) = log(p/(1-p))$ and $d(x) = x$. Thus, the Binomial distribution is an exponential family.

Add 2

Describe how to obtain a bootstrap distribution for a chosen statistic and obtain a (say) 95 percent confidence interval for its related parameter of interest.

SOLUTION

We start with a data set of n observations with p variables. A bootstrap sample is a sample of n observations selected WITH replacement from the set of n observations in our data set. All p variables are retained. Some of the original observations will be repeated and others will be left out. We compute our statistic on the original data set, and then compute the statistic again on B bootstrapped samples. B is usually 1000 or more. This gives us a bootstrap distribution for the chosen statistic - a distribution of values for the statistic from "similar" samples.

Once obtained, what are some of the uses of a bootstrap distribution? Well, one use is to get a confidence interval for the related parameter (whatever the statistic is estimating). Assuming the bootstrap distribution of the statistic is not too skewed, we can just take appropriate quantiles. I.E. for a 95 percent CI, take the 2.5 and 97.5 quantiles of the bootstrap distribution.

Alternatives based on normal-shaped or skewed distributions were explored in Stat 230. But the quantile idea is a basic approach that is widely applicable.

Add 3

Explain the plug-in principle in relation to obtaining the standard error of a sample proportion.

SOLUTION

Working from properties of the binomial distribution for a RV X $Bin(n, p)$, we can find that $\hat{p} = X/n$ has a standard deviation of $\sqrt{p(1-p)/n}$. We do not usually know the value of p though, hence why we are interested in $\hat{p}$ to begin with. The plug-in principle says that we can use $\hat{p}$ in place of p in the formula and obtain the standard error of $\hat{p}$, which is our best estimate of the standard deviation of $\hat{p}$.

Hence, we find that $SE(\hat{p}) = \sqrt{\hat{p}(1-\hat{p})/n}$.

Add 4

Explain what a Bayesian conjugate family is and what its benefits are.

SOLUTION

Bayesian conjugate families describe situations where the distributions for the prior and data are "nice" interacting with each other such that the posterior is from the same family as the prior, but with altered parameters (basically, updated based on the data).

Examples: We derived a Gamma-Poisson (prior is Gamma, data is Poisson) result and you had a Beta-Binomial (prior is Beta, data observed was Binomial (or independent Bernoullis)) on your homework. There is also a Normal-Normal (Normal prior for the mean of the normal model for the data), and others.

The major benefit of working in these families is the easy (tractable) math.

Add 5

Using the Chapter 8 Lab data set for context, explain how to perform a permutation test to see whether the mean age differs for individuals with monthly income > 7500 versus individuals with less than that monthly income.

```
credit <- read.csv("https://awagaman.people.amherst.edu/stat495/creditsample.csv", header = T)
```

SOLUTION

We have a group variable that indicates whether individuals have a monthly income > 7500 versus not. (Equal to 7500 should go in one group, so putting in less than group.)

```
credit <- mutate(credit, group = ifelse(MonthlyIncome > 7500, 1, 0))
tally(~ group, data = credit)
```

```
## group
##     0     1  <NA>
## 16862  7183  5955
```

We have 16862 individuals with income <= 7500 and 7183 with income > 7500. We ignore NAs here (you can argue that you want to include them in the shuffling as well).

We take the 24045 individuals with income recorded and record their average age by group.

```
favstats(age ~ group, data = credit)
```

```
##   group min Q1 median Q3 max    mean      sd     n missing
## 1     0  21 39     50 62 102 50.5875 15.2925 16862       0
## 2     1  24 45     53 61  97 53.1694 11.4434  7183       0
```

We see that the higher income group has a slightly higher mean age. But is this statistically significant? The permutation test helps us address this question.

With an appropriate seed set for reproducibility, we take the same individuals and shuffle their group labels, recomputing the average ages per group. We can save some summary measure of that difference - perhaps the t-test statistic that would result or the difference in means itself. We repeat this process. This generates a distribution of summary measures that would be possible under the null hypothesis of no relationship of age with income group. Once we have an appropriate distribution built up, we assess how unusual our observed measure on the original sample was in comparison to the distribution. If we have a value in the tails, it suggests that our results would not be possible under the null hypothesis, and we would have a significant result.

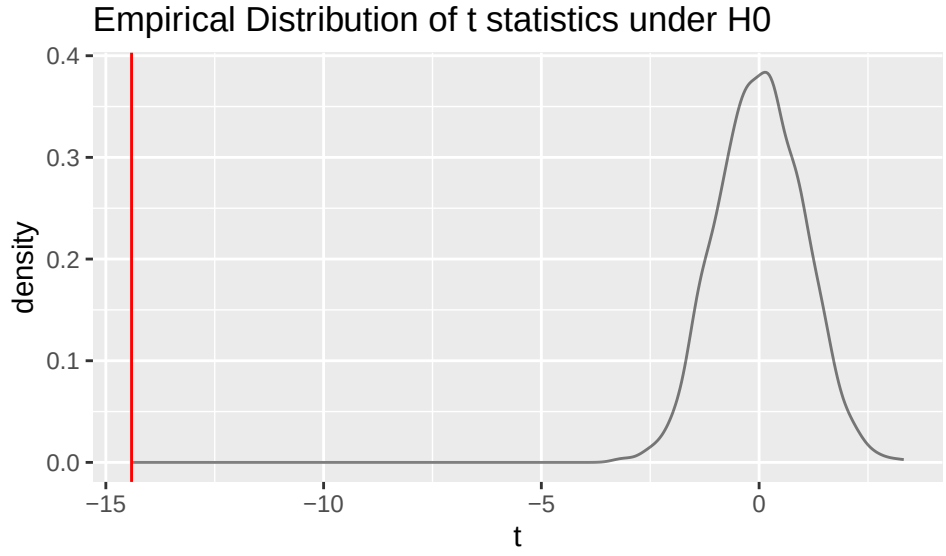
In this particular example, code to do this could look like this:

```
credit2 <- filter(credit, group != "NA")

t.test(age ~ group, data = credit2)
```

```
##
##  Welch Two Sample t-test
##
## data:  age by group
## t = -14.41, df = 17863, p-value <2e-16
## alternative hypothesis: true difference in means between group 0 and group 1 is not equal to 0
## 95 percent confidence interval:
##  -2.93313 -2.23077
## sample estimates:
## mean in group 0 mean in group 1
##         50.5875         53.1694
```

```
set.seed(500) #make the results reproducible
ttest <- do(1000) * (t.test(age ~ shuffle(group), data = credit2)$statistic)
ttest <- as.data.frame(ttest)

gf_dens(~ t, data = ttest) %>%
  gf_vline(xintercept = -14.41, color = "red") %>%
  gf_labs(title = "Empirical Distribution of t statistics under H0")
```

```
## Warning: geom_vline(): Ignoring `mapping` because `xintercept` was provided.
```



```
pdata(~ t, -14.41, data = ttest, lower.tail = TRUE)
```

```
## [1] 0
```

In this particular case, it's clear that none of the statistics under the empirical null distribution are anywhere near our observed t from the original data. This suggests that the average age for the higher income group is above that of the average age of the lower income group.

Add 6

The diabetes data set (loaded below) is used to demonstrate ridge regression in Chapter 7.

```
diabetes <- read.csv("http://web.stanford.edu/~hastie/CASI_files/DATA/diabetes.csv", header = TRUE)
diabetes <- select(diabetes, -X)
```

The data was pre-processed. The text states that the predictors were standardized to mean 0 and sum of squares 1, and the response had it's mean subtracted off (i.e. it was centered), but not scaled.

```
n <- nrow(diabetes)
diabetes2 <- mutate(diabetes, prog = scale(prog, scale = FALSE),
                    age = scale(age)/sqrt(n-1),
                    sex = scale(sex)/sqrt(n-1),
                    bmi = scale(bmi)/sqrt(n-1),
                    map = scale(map)/sqrt(n-1),
                    tc = scale(tc)/sqrt(n-1),
                    ldl = scale(ldl)/sqrt(n-1),
                    hdl = scale(hdl)/sqrt(n-1),
                    tch = scale(tch)/sqrt(n-1),
                    ltg = scale(ltg)/sqrt(n-1),
                    glu = scale(glu)/sqrt(n-1))
```

```
#so you can see the matrix algebra to verify sum of squares 1
# we keep the intercept column here to do the linear algebra ourselves
x <- model.matrix(prog ~ ., diabetes2)[, ]
y <- diabetes2 %>% select(prog) %>%  unlist() %>%  as.numeric()
S <- t(x) %*% x
diag(S)
```

```
## (Intercept)         age         sex         bmi         map          tc
##         442           1           1           1           1           1
##         ldl         hdl         tch         ltg         glu
##           1           1           1           1           1
```

Note that while you'd usually want a training/test set here, the book used the entire data set, so you should do that to try to verify their work.

(a) Use OLS to predict *prog* using *diabetes2* and verify you obtain the results in Table 7.3.

SOLUTION

First we get the coefficients.

```
# OLS via matrix algebra
beta <- solve(S) %*% t(x) %*% y
round(beta, 2)
```

```
##                [,1]
## (Intercept)    0.00
## age          -10.01
```

```
## sex         -239.82
## bmi          519.84
## map          324.39
## tc          -792.18
## ldl          476.75
## hdl          101.04
## tch          177.06
## ltg          751.28
## glu           67.63
```

```
# OLS via lm
proglm <- lm(prog ~ ., data = diabetes2)
summary(proglm)
```

```
##
## Call:
## lm(formula = prog ~ ., data = diabetes2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -155.83  -38.53   -0.23   37.81  151.35
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.40e-14   2.58e+00    0.00   1.0000
## age         -1.00e+01   5.97e+01   -0.17   0.8670
## sex         -2.40e+02   6.12e+01   -3.92   0.0001 ***
## bmi          5.20e+02   6.65e+01    7.81  4.3e-14 ***
## map          3.24e+02   6.54e+01    4.96  1.0e-06 ***
## tc          -7.92e+02   4.17e+02   -1.90   0.0579 .
## ldl          4.77e+02   3.39e+02    1.41   0.1604
## hdl          1.01e+02   2.13e+02    0.48   0.6347
## tch          1.77e+02   1.61e+02    1.10   0.2735
## ltg          7.51e+02   1.72e+02    4.37  1.6e-05 ***
## glu          6.76e+01   6.60e+01    1.02   0.3060
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 54.2 on 431 degrees of freedom
## Multiple R-squared:  0.518,  Adjusted R-squared:  0.507
## F-statistic: 46.3 on 10 and 431 DF,  p-value: <2e-16
```

```
round(coef(proglm),2)
```

```
## (Intercept)         age         sex         bmi         map          tc
##        0.00      -10.01     -239.82      519.84      324.39     -792.18
##         ldl         hdl         tch         ltg         glu
##      476.75      101.04      177.06      751.28       67.63
```

To check the standard deviations reported, remember that $\sigma^2 = 54.16^2$.

```
sqrt(diag(54.16^2 * solve(S)))
```

```
## (Intercept)         age         sex         bmi         map          tc
##     2.57613    59.75560    61.22889    66.54071    65.42897   416.72868
##         ldl         hdl         tch         ltg         glu
##   339.07093   212.55542   161.49297   171.92044    65.99130
```

The results here are consistent with the results in Table 7.3.

(b) Use ridge regression with lambda = 0.1 and verify you obtain the results in Table 7.3.

SOLUTION

```
#so you can see the matrix algebra
x <- model.matrix(prog ~ ., diabetes2)[, ]
y <- diabetes2 %>% select(prog) %>%  unlist() %>%  as.numeric()
S <- t(x) %*% x
beta <- solve(S + diag(rep(0.1, 11))) %*% t(x) %*% y
round(beta, 2)
```

```
##                [,1]
## (Intercept)    0.00
## age            1.31
## sex         -207.19
## bmi          489.69
## map          301.77
## tc           -83.47
## ldl          -70.83
## hdl         -188.68
## tch          115.71
## ltg          443.81
## glu           86.75
```

To check the SDs, with the linear algebra it turns out that you can do that with:

```
sqrt(diag(54.16^2 * solve(S + diag(rep(0.1, 11))) %*% S %*% t(solve(S + diag(rep(0.1, 11))))))
```

```
## (Intercept)         age         sex         bmi         map          tc
##     2.57555    52.75252    53.24875    56.41770    55.80140    43.68201
##         ldl         hdl         tch         ltg         glu
##    52.48001    58.49404    70.84173    58.43990    56.63693
```

Minor differences here that I see may be due to rounding, but overall, the results for estimators and SEs are consistent with the text.

You should expect to see glmnet output for ridge and lasso, which is why you are implementing that for practice below instead of doing the linear algebra solution.

(c) Explain what cross-validation is (generally), and how it can be used to help choose a lambda in ridge regression.

SOLUTION

Cross-validation is a practice that can be useful when trying to choose tuning parameters for models. In the context of ridge regression, we have to choose our lambda value. For k-fold CV, we divide the data set into k non-overlapping pieces of equal sizes randomly. For example if $n = 100$ and $k = 5$, we would split the data set into 5 pieces of 20 observations each (it's technically a partition - each observation is in exactly one of the pieces). We would then fit our model with various values of the tuning parameter(s) on 4/5 of the data (leaving out one piece), and save some measure of performance as measured on the 1/5 left out. We cycle through this process, letting each piece take a turn as the piece left out. At the end, we average across the measures of performance we saved for each value of the tuning parameter(s). We choose the tuning parameter that has the best measure of performance on average. For ridge regression, we minimize the MSE. So, the lambda selected has the lowest average MSE of the lambda values studied via CV.

(d) Use CV with ridge regression to select your lambda running the default grid through glmnet (i.e. don't specify a grid). What lambda is selected?

SOLUTION

Remember that we were told to use the entire data set (not a training/test split) to try to compare to the text results. Normally, you'd want the training/test split!

```
# x2 and y2 created from unscaled data
# the -1 here takes off the intercept column - important to remove!
x2 <- model.matrix(prog ~ ., diabetes)[, -1]
y2 <- diabetes %>% select(prog) %>%  unlist() %>%  as.numeric()

set.seed(1)
cv.out <- cv.glmnet(x2, y2, alpha = 0) # Fit ridge regression model
bestlam <- cv.out$lambda.min  # Select lambda that minimizes MSE
bestlam
```

```
## [1] 4.516
```

```
# just to show you that it doesn't matter if you run this on the scaled data or not
# because function has internal scaling anyway
# x and y were created above on the scaled data
set.seed(1)
cv.out <- cv.glmnet(x, y, alpha = 0) # Fit ridge regression model
bestlam <- cv.out$lambda.min  # Select lambda that minimizes MSE
bestlam
```

```
## [1] 4.516
```

From glmnet, the best lambda is 4.516 for ridge.

Let's look at our coefficients, etc.

```
ridge_mod <- glmnet(x2, y2, alpha = 0)
round(coefficients(ridge_mod, s = bestlam), 3)
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                    s1
## (Intercept) -256.424
## age           -0.007
## sex          -20.847
## bmi            5.437
## map            1.066
## tc            -0.165
## ldl           -0.078
## hdl           -0.664
## tch            4.190
## ltg           99.256
## glu            0.334
```

This drove many of the coefficients towards 0.

(e) Implement LASSO with CV to select your lambda running the default grid through glmnet. What lambda is selected?

SOLUTION

```
set.seed(1)
cv.outl <- cv.glmnet(x2, y2, alpha = 1) # Fit lasso model
bestlaml <- cv.outl$lambda.min  # Select lambda that minimizes MSE
bestlaml
```

```
## [1] 0.0972943
```

From glmnet, the best lambda is 0.0972943 for lasso.

Let's look at our coefficients, etc.

```
lasso_mod <- glmnet(x2, y2, alpha = 1)
round(coefficients(lasso_mod, s = bestlaml), 3)
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                    s1
## (Intercept) -322.932
## age           -0.021
## sex          -22.361
## bmi            5.635
## map            1.103
## tc            -0.744
## ldl            0.433
## hdl           -0.026
## tch            5.403
## ltg          138.094
## glu            0.275
```

Here, lasso doesn't appear to have removed any variables and instead just did shrinkage like ridge. It actually looks to be less shrinkage than ridge (the coefficients, returned on their original scales, are larger in magnitude than the ridge solution).

(f) Implement a regression tree to predict *prog*. What cp corresponds to the default tree?

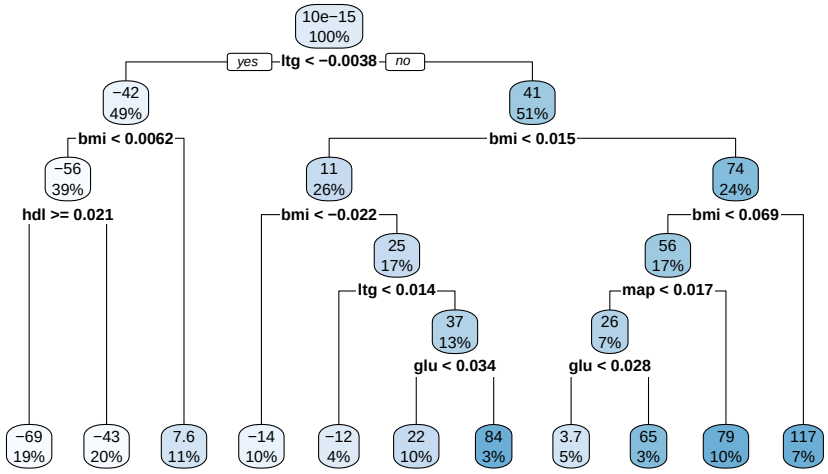
SOLUTION

I continued to use the standardization of the variables that was used above.

```
# method = "anova" for regression
set.seed(495)
prog.rpart <- rpart(prog ~ . , data = diabetes2, method = "anova")
printcp(prog.rpart)
```

```
##
## Regression tree:
## rpart(formula = prog ~ ., data = diabetes2, method = "anova")
##
## Variables actually used in tree construction:
## [1] bmi glu hdl ltg map
##
## Root node error: 2621009/442 = 5930
##
## n= 442
##
##         CP nsplit rel error xerror    xstd
## 1  0.29154      0    1.0000 1.0069 0.05083
## 2  0.08523      1    0.7085 0.8287 0.05314
## 3  0.05660      2    0.6232 0.7657 0.05190
## 4  0.03066      3    0.5666 0.6952 0.04659
## 5  0.02031      4    0.5360 0.6825 0.04716
## 6  0.01569      5    0.5157 0.6841 0.04713
## 7  0.01345      6    0.5000 0.6680 0.04713
## 8  0.01101      8    0.4731 0.6537 0.04647
## 9  0.01055      9    0.4621 0.6454 0.04660
## 10 0.01000     10    0.4515 0.6452 0.04619
```

```
rpart.plot(prog.rpart)
```




```
pdf("prog.pdf", width = 14, height = 14)
plot(rpart.plot(prog.rpart))
dev.off()
```

```
## pdf
##   2
```

This doesn't grow a huge tree. In OLS, the significant variables were sex, bmi, map, and ltg. The variables involved here leave out sex and include glu and hdl. So, in terms of the variables used, they are similar. The cp for the default tree at the end is 0.01000. (Reported in the cp output.)

(g) Explain why we don't need a *glm* here to predict *prog*.

SOLUTION

prog is a continuous, numeric response. We use *glm*'s when the response is different from this - logistic for 0/1, poisson for counts, etc. So, we don't need a glm here - OLS is fine conceptually (though we didn't check conditions for it).

(h) Which model of these do you prefer? How do the fits differ? (Hint: Besides comparing coefficients/involved variables, you can compare MSEs.)

SOLUTION

Let's do the computations to get our MSEs before we do a comparison.

Since we have the OLS fit, we can get that MSE fairly easily.

```
54.16^2
```

```
## [1] 2933.31
```

For the ridge fit with lambda of 0.1, we can use our vector of betas to get predictions ourselves. Here, I didn't use the rounded betas, but if you do that, I'm not sure you'll see a huge change.

```
ridgesmallpred <- x %*% beta
ridgesmallMSE <- mean((ridgesmallpred-y)^2)
ridgesmallMSE
```

```
## [1] 2890.45
```

Now we get the MSEs from the glmnet fits of ridge and lasso with lambdas chosen by CV:

```
ridge_pred <- predict(ridge_mod, s = bestlam, newx = x2) # Use best lambda
mean((ridge_pred - y2)^2) # Calculate test MSE
```

```
## [1] 2881.33
```

```
lasso_pred <- predict(lasso_mod, s = bestlaml, newx = x2) # Use best lambda
mean((lasso_pred - y2)^2) # Calculate test MSE
```

```
## [1] 2862.19
```

Then, for our tree, we get:

```
fittedtree <- predict(prog.rpart)
mean((fittedtree - diabetes2$prog)^2)
```

```
## [1] 2677.44
```

Now we can do our comparison.

OLS has an MSE of 2933.31 and keeps all predictors. The ridge with lambda of 0.1 has an MSE of 2890.45 and keeps all predictors. Ridge with lambda chosen by cv has an MSE of 2881.33 and keeps all predictors. The lasso here also keeps all predictors (no selection) and the MSE is 2862.19. Finally, the tree uses ltg, bmi, hdl, map, and glu, and has the lowest MSE of 2677.44. From these results, it looks like the tree is doing the best in terms of both reducing the number of variables we need to consider and giving us better (on average) predictions. The shrinkage methods (ridge and lasso) all improve over OLS, but not as much as the tree.

Add 7

The Spam email data set (loaded below) is used to demonstrate logistic regression and classification trees in Chapter 8. Note that while you'd usually want a training/test set here, the book used the entire data set, so you should do that to try to verify their work.

```
spam <- read.csv("http://web.stanford.edu/~hastie/CASI_files/DATA/SPAM.csv", header = TRUE)
```

(a) Why does it not make sense to use OLS to predict *spam*?

SOLUTION

The response is a 0/1 variable, so OLS doesn't make sense. It doesn't have a way to constrain the response values to be 0/1.

(b) Use logistic regression to predict *spam*. Verify you obtain the results in Table 8.3.

We'd usually want a training/test set idea here but to match the book, I'm using the entire data set.

The footnote reminds us that the X matrix was standardized. I'm assuming that means center = 0 and sd = 1. I'm hoping that thinking through these issues with scaling give you a sense of why reproducibility is important. You should specify transformations you do CLEARLY so that others can replicate your results.

```
# remove id variable and scale all predictors
# this is done here by adding the response back in
# can also use a mutate targeted at only the numeric predictors
spamdata <- select(spam, -spam, -testid)
spamdata <- data.frame(scale(spamdata))
spamdata <- mutate(spamdata, spam = spam$spam)
```

With the variables appropriately transformed, we can fit the logistic regression.

```
loglm <- glm(spam ~ . , data = spamdata, family = "binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(loglm)
```

```
##
## Call:
## glm(formula = spam ~ ., family = "binomial", data = spamdata)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -4.127  -0.203   0.000   0.114   5.364
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -12.2653     1.9908   -6.16  7.2e-10 ***
## make         -0.1189     0.0707   -1.68  0.09239 .
## address      -0.1881     0.0894   -2.10  0.03536 *
## all           0.0575     0.0556    1.03  0.30076
## X3d           3.1412     2.1025    1.49  0.13517
## our           0.3782     0.0685    5.52  3.3e-08 ***
## over          0.2418     0.0684    3.53  0.00041 ***
## remove        0.8919     0.1303    6.85  7.6e-12 ***
## internet      0.2285     0.0675    3.39  0.00071 ***
## order         0.2046     0.0794    2.58  0.00996 **
## mail          0.0822     0.0468    1.76  0.07923 .
## receive      -0.0515     0.0600   -0.86  0.39066
## will         -0.1192     0.0638   -1.87  0.06177 .
## people       -0.0240     0.0693   -0.35  0.72956
## report        0.0485     0.0457    1.06  0.28885
## addresses     0.3200     0.1878    1.70  0.08837 .
## free          0.8577     0.1203    7.13  1.0e-12 ***
## business      0.4262     0.1000    4.26  2.0e-05 ***
## email         0.0639     0.0622    1.03  0.30453
## you           0.1444     0.0622    2.32  0.02033 *
## credit        0.5339     0.2744    1.95  0.05167 .
## your          0.2905     0.0630    4.61  3.9e-06 ***
## font          0.2065     0.1669    1.24  0.21584
## X000          0.7865     0.1651    4.76  1.9e-06 ***
## money         0.1887     0.0718    2.63  0.00853 **
## hp           -3.2097     0.5228   -6.14  8.3e-10 ***
## hpl          -0.9226     0.3899   -2.37  0.01797 *
## george      -39.6236     7.1155   -5.57  2.6e-08 ***
## X650          0.2399     0.1072    2.24  0.02526 *
## lab          -1.4752     0.8909   -1.66  0.09774 .
## labs         -0.1506     0.1432   -1.05  0.29297
## telnet       -0.0687     0.1943   -0.35  0.72374
## X857          0.8374     1.0787    0.78  0.43757
## data         -0.4105     0.1733   -2.37  0.01784 *
## X415          0.2200     0.5273    0.42  0.67649
## X85          -1.0940     0.4196   -2.61  0.00912 **
## technology    0.3719     0.1244    2.99  0.00280 **
## X1999         0.0197     0.0743    0.27  0.79082
## parts        -0.1317     0.0934   -1.41  0.15847
## pm           -0.3760     0.1664   -2.26  0.02384 *
## direct       -0.1066     0.1272   -0.84  0.40221
## cs          -16.2716     9.6074   -1.69  0.09033 .
```

```
## meeting      -2.0617     0.6429   -3.21  0.00134 **
## original     -0.2791     0.1805   -1.55  0.12198
## project      -0.9785     0.3292   -2.97  0.00295 **
## re           -0.8016     0.1575   -5.09  3.6e-07 ***
## edu          -1.3295     0.2447   -5.43  5.5e-08 ***
## table        -0.1774     0.1266   -1.40  0.16096
## conference   -1.1474     0.4603   -2.49  0.01267 *
## ch.          -0.3143     0.1077   -2.92  0.00350 **
## ch..1        -0.0509     0.0674   -0.75  0.45066
## ch..2        -0.0719     0.0917   -0.78  0.43291
## ch..3         0.2832     0.0728    3.89  0.00010 ***
## ch..4         1.3120     0.1737    7.55  4.2e-14 ***
## ch..5         1.0318     0.4780    2.16  0.03088 *
## crl.ave       0.3803     0.5976    0.64  0.52451
## crl.long      1.7771     0.4912    3.62  0.00030 ***
## crl.tot       0.5116     0.1365    3.75  0.00018 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6170.2  on 4600  degrees of freedom
## Residual deviance: 1815.8  on 4543  degrees of freedom
## AIC: 1932
##
## Number of Fisher Scoring iterations: 13
```

Huzzah! It matches.

(c) Obtain a confusion matrix for the logistic regression. How well is the model doing?

SOLUTION

Let's get a sense of our baseline.

```
tally(~ spam, data =spamdata)
```

```
## spam
##  TRUE FALSE
##  1813  2788
```

```
1813/4601
```

```
## [1] 0.394045
```

About 39.4% of the data is spam. So we could get about 60.6% right just by saying it's all not spam. We want to do better than that.

If we use a naive cutoff of 0.5 to make predictions, what do we get?

```
logaug <- loglm %>% augment(type.predict = "response")
logaug <- mutate(logaug, binprediction = round(.fitted, 0))
tally(~ binprediction, data = logaug)
```

```
## binprediction
##    0    1
## 2860 1741
```

This is actually pretty close to the fraction we'd expect:

```
1741/4601
```

```
## [1] 0.378396
```

It's predicting 37.84 percent spam. So, we can just make a confusion matrix from this.

```
with(logaug, table(spam, binprediction))
```

```
##        binprediction
## spam       0    1
##   FALSE 2666  122
##   TRUE   194 1619
```

Overall, we are getting 93.1% correct. So, the model is doing very well.

```
(2666+1619)/4601
```

```
## [1] 0.931319
```

In terms of each specific class, we can get error rates too.

```
122/(2666+122)
```

```
## [1] 0.043759
```

```
194/(194+1619)
```

```
## [1] 0.107005
```

We are misclassifying 4.37 percent of not spam as spam, and 10.7 percent of spam as not spam.

(d) Use a classification tree to predict *spam*. Attempt to obtain the tree in Figure 8.7. Note that you may need to prune or adjust control parameters to obtain this tree. If you cannot obtain the tree, obtain one you want to work with for part e below.

SOLUTION

The text says it used rpart, but doesn't specify how the variables were standardized. We leave them as above, continuing with the spamdata data set.

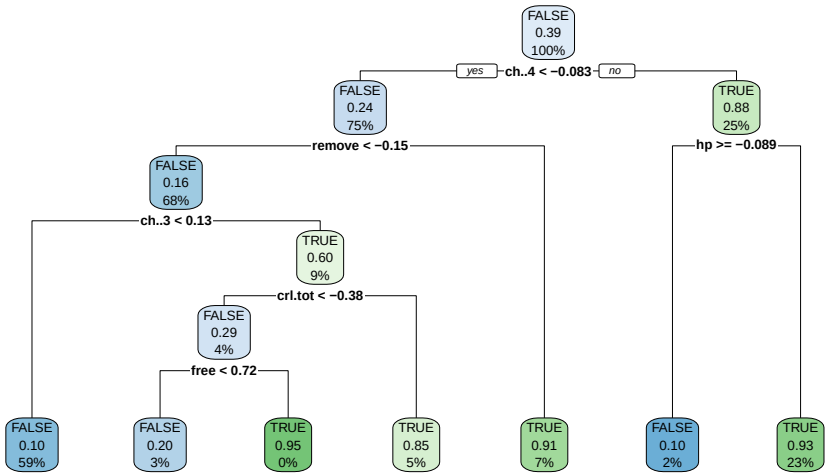
From here, it looks like they may have indeed just used the default tree grown by rpart. We can check if we view the tree.

```
set.seed(495)
spam.rpart <- rpart(factor(spam) ~ ., method ="class", data = spamdata)
printcp(spam.rpart)
```

```
##
## Classification tree:
## rpart(formula = factor(spam) ~ ., data = spamdata, method = "class")
##
## Variables actually used in tree construction:
## [1] ch..3   ch..4   crl.tot free    hp      remove
##
## Root node error: 1813/4601 = 0.394
##
## n= 4601
##
##         CP nsplit rel error xerror    xstd
## 1 0.47656      0    1.0000 1.0000 0.01828
## 2 0.14892      1    0.5234 0.5433 0.01535
## 3 0.04302      2    0.3745 0.4468 0.01425
```

```
## 4 0.03089      4    0.2885 0.3271 0.01254
## 5 0.01048      5    0.2576 0.2796 0.01172
## 6 0.01000      6    0.2471 0.2653 0.01145
```

```
rpart.plot(spam.rpart)
```




```
pdf("spamtree.pdf", width = 14, height = 14)
plot(rpart.plot(spam.rpart))
dev.off()
```

```
## pdf
##   2
```

Indeed! Default tree. I had an odd error when I didn't have spam as a factor here, even though it's a logical variable. Either way, default tree! Note that the CV error and such is off, but extremely consistent, as we don't know what seed they used when that was being evaluated.

(e) Obtain a confusion matrix for your classification tree. How do your error rates compare to those reported in Figure 8.7?

SOLUTION

```
spampredict <- predict(spam.rpart, type = "class")
table(spampredict, spamdata$spam)
```

```
##
## spampredict FALSE TRUE
##       FALSE  2654  314
##       TRUE    134 1499
```

Let's take a look at the error rates.

```
(2654+1499)/4601
```

```
## [1] 0.90263
```

```
314/(314+2654)
```

```
## [1] 0.105795
```

```
134/(134+1499)
```

```
## [1] 0.0820576
```

Overall, the tree is about 90.3 percent accurate. We have training error of 10.6 percent on not spam and 8.2 percent on spam.

This is overly optimistic. We can look at the CV error in the output to see a better estimate of error.

```
0.394045 * 0.274683
```

```
## [1] 0.108237
```

The estimated error is 10.83 percent, which is still not bad. We don't have a similar value to compare to for the logistic regression, since we didn't split into training/test. It's not clear if they did training/test here using the testid variable, but they had different error rates in Figure 8.7.

(f) Which method do you prefer for predicting *spam*?

SOLUTION

The tree has slightly worse performance but is MUCH easier to understand. It requires fewer variables. So I'd probably go with the tree, though I might grow it out a bit more if I find that's helpful.

CASI 8.6

```
galaxy <- read.table("http://web.stanford.edu/~hastie/CASI_files/DATA/galaxy.txt", header = TRUE)
```

Data set description: Table of counts of galaxies binned into categories defined by redshift and magnitude. The column labels are log-redshift values, and the row labels magnitude.

Fit the Poisson regression model (8.39) to the galaxy data.

(Note that some data wrangling is required based on how the data is provided.)

SOLUTION

```
# add m values to the data set
galaxy2 <- mutate(galaxy, m = 18:1)

# pivot to get the log_r values in a variable
galaxydata <- galaxy2 %>%
  pivot_longer(-m, names_to = "log_r", values_to = "count")

# add r values 1:15 repeated
galaxydata <- mutate(galaxydata, r = c(rep(1:15, 18)))
```

This is slightly odd, as they are using the bin numbers, not actual variable values in the models, at least, based on their description. You CAN set up (I started to) the wrangling to get the actual m and r values back. Either way, the idea here is to fit a Poisson regression to verify you can read the output, etc.

Now we can go fit the model.

```
myglm <- glm(count ~ r*m + I(r^2) + I(m^2), data = galaxydata, family = poisson)
```

This is a complete second-order model.

```
lmtest::lrtest(myglm)
```

```
## Likelihood ratio test
##
## Model 1: count ~ r * m + I(r^2) + I(m^2)
```

```
## Model 2: count ~ 1
##   #Df LogLik Df Chisq Pr(>Chisq)
## 1   6 -317.7
## 2   1 -654.4 -5 673.4     <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

It looks like the overall model is significant.

Add 8

```
data(Credit)
Credit <- select(Credit, -ID)
```

You may want to look over the help file for this data set. Note that the number of observations stated there is inaccurate. Our goal is to predict Balance. Do not worry about transforming Balance or any of the other predictors here.

(a) Create an appropriate training/test split using a 75/25 percent split.

SOLUTION

```
set.seed(495) # set any value you want!

n <- nrow(Credit)
train_index <- sample(1:n, 0.75 * n)
test_index <- setdiff(1:n, train_index)
train <- Credit[train_index, ]
test <- Credit[test_index, ]

# the -1 removes the intercept column
x_train <- model.matrix(Balance ~ ., train)[, -1]
x_test <- model.matrix(Balance ~ ., test)[, -1]

y_train <- train %>%
  select(Balance) %>%
  unlist() %>%
  as.numeric()

y_test <- test %>%
  select(Balance) %>%
  unlist() %>%
  as.numeric()
```

(b) Perform best subsets and choose a model based on an appropriate descriptive statistic. Compute the test MSE.

SOLUTION

```
best <- regsubsets(Balance ~ . , data = train, nbest = 1)
with(summary(best), data.frame(rsq, adjr2, cp, rss, outmat))
```

```
##               rsq    adjr2         cp      rss Income Limit Rating Cards Age
## 1  ( 1 ) 0.737498 0.736617 1372.22797 17219852                  *
## 2  ( 1 ) 0.876864 0.876035  488.53965  8077563      *           *
## 3  ( 1 ) 0.948394 0.947871   35.96175  3385300      *           *
## 4  ( 1 ) 0.952658 0.952016   10.86482  3105599      *     *           *
```

```
## 5  ( 1 ) 0.953458 0.952666    7.78012  3053114      *     *           *   *
## 6  ( 1 ) 0.953880 0.952936    7.09635  3025411      *     *           *   *
## 7  ( 1 ) 0.954233 0.953136    6.85578  3002284      *     *      *    *   *
## 8  ( 1 ) 0.954453 0.953201    7.45686  2987844      *     *      *    *   *
##          Education GenderFemale StudentYes MarriedYes EthnicityAsian
## 1  ( 1 )
## 2  ( 1 )
## 3  ( 1 )                              *
## 4  ( 1 )                              *
## 5  ( 1 )                              *
## 6  ( 1 )         *                    *
## 7  ( 1 )         *                    *
## 8  ( 1 )         *           *        *
##          EthnicityCaucasian
## 1  ( 1 )
## 2  ( 1 )
## 3  ( 1 )
## 4  ( 1 )
## 5  ( 1 )
## 6  ( 1 )
## 7  ( 1 )
## 8  ( 1 )
```

We will choose our model based on Cp. In which case, it looks like we want model 7, which has predictors Income, Limit, Rating, Cards, Age, Education, and Student.

```
bestmod <- lm(Balance ~ Income + Limit + Rating + Cards + Age + Education + Student,
              data = train)
bestpred <- predict(bestmod, newdata = test)
mean((bestpred-test$Balance)^2)
```

```
## [1] 8344.53
```

The test MSE is 8344.53.

(c) Perform forward selection and choose a model based on an appropriate descriptive statistic. Compute the test MSE.

SOLUTION

```
forward <- regsubsets(Balance ~ . , data = train, method = "forward", nbest = 1)
with(summary(forward), data.frame(rsq, adjr2, cp, rss, outmat))
```

```
##              rsq    adjr2         cp      rss Income Limit Rating Cards Age
## 1  ( 1 ) 0.737498 0.736617 1372.22797 17219852                  *
## 2  ( 1 ) 0.876864 0.876035  488.53965  8077563      *           *
## 3  ( 1 ) 0.948394 0.947871   35.96175  3385300      *           *
## 4  ( 1 ) 0.951009 0.950344   21.34600  3213789      *     *     *
## 5  ( 1 ) 0.953005 0.952206   10.65582  3082797      *     *     *     *
## 6  ( 1 ) 0.953843 0.952897    7.33548  3027880      *     *     *     *   *
## 7  ( 1 ) 0.954233 0.953136    6.85578  3002284      *     *     *     *   *
## 8  ( 1 ) 0.954453 0.953201    7.45686  2987844      *     *     *     *   *
##          Education GenderFemale StudentYes MarriedYes EthnicityAsian
## 1  ( 1 )
## 2  ( 1 )
## 3  ( 1 )                              *
## 4  ( 1 )                              *
```

```
## 5  ( 1 )                                *
## 6  ( 1 )                                *
## 7  ( 1 )         *                      *
## 8  ( 1 )         *           *          *
##          EthnicityCaucasian
## 1  ( 1 )
## 2  ( 1 )
## 3  ( 1 )
## 4  ( 1 )
## 5  ( 1 )
## 6  ( 1 )
## 7  ( 1 )
## 8  ( 1 )
```

Based on forward selection and the cp stat, we want model 7, which is the same model chosen with best subsets. Note that this model turns out to have severe multicollinearity issues between Rating and Limit. Since we know this (you can check out the VIFs), we can consider an alternative model.

```
car::vif(bestmod)
```

```
##    Income     Limit    Rating     Cards       Age Education   Student
##   2.83614 246.92106 251.45380   1.50092   1.04566   1.01319   1.01403
```

Limit and Rating are where the multicollinearity issues are, so suppose we drop Limit from the model.

```
adjmod <- lm(Balance ~ Income + Rating + Cards + Age + Education + Student, data = train)
adjpred <- predict(adjmod, newdata = test)
mean((adjpred-test$Balance)^2)
```

```
## [1] 8998.83
```

```
car::vif(adjmod)
```

```
##    Income    Rating     Cards       Age Education   Student
##   2.83573   2.81579   1.04947   1.04368   1.01126   1.00850
```

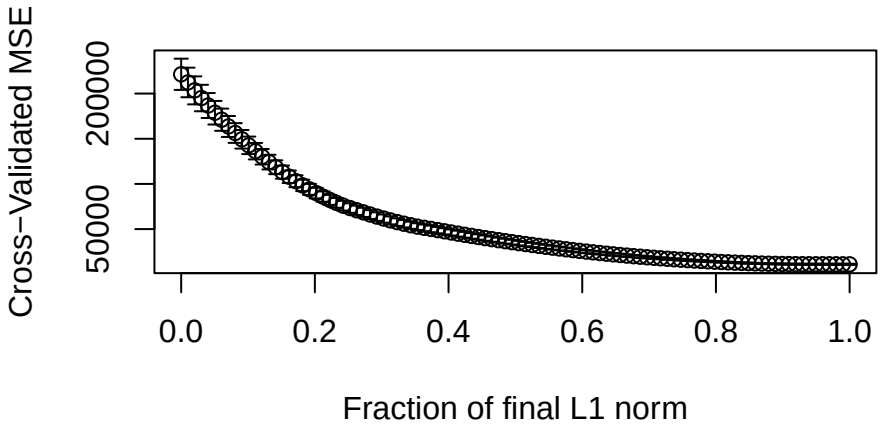
The test MSE is 8998.83, and we don't see evidence of multicollinearity issues anymore.

(d) Perform the lasso and choose a model based on an appropriate descriptive statistic. Compute the test MSE.

SOLUTION

For this solution, I wanted to show you example output from lars, so it's coded using that function.

```
set.seed(495)
object1 <- lars(x_train, y_train, type = "lasso")
object2 <- cv.lars(x_train, y_train, type = "lasso", trace = FALSE) #TRUE would show all the steps
```




```
min(object2$cv)
```

```
## [1] 11016.6
```

```
object2$cv
```

```
##   [1] 221326.9 212309.6 203548.5 195043.7 186795.1 178801.9 171065.0 163584.7
##   [9] 156361.0 149394.4 142697.0 136256.2 130071.5 124143.0 118470.7 113054.6
##  [17] 107898.1 103001.7  98361.1  93976.3  89847.3  85974.1  82356.8  78995.2
##  [25]  75897.1  73273.4  70838.4  68504.4  66110.1  63826.1  61664.2  59626.0
##  [33]  57711.3  55920.4  54253.0  52705.7  51409.1  50184.5  48861.0  47536.3
##  [41]  46235.4  44955.3  43694.7  42458.0  41245.2  40056.3  38891.4  37750.3
##  [49]  36633.1  35539.8  34470.5  33425.0  32403.5  31405.8  30432.1  29482.2
##  [57]  28556.3  27654.3  26776.2  25922.0  25092.4  24294.7  23520.6  22769.9
##  [65]  22043.5  21342.8  20672.1  20029.8  19410.5  18814.2  18238.5  17680.4
##  [73]  17143.5  16627.2  16133.4  15660.8  15209.1  14775.7  14361.0  13964.3
##  [81]  13590.1  13238.3  12908.9  12603.9  12335.2  12094.3  11878.3  11687.4
##  [89]  11517.6  11374.5  11263.7  11183.5  11125.6  11093.5  11078.2  11065.0
##  [97]  11051.1  11034.4  11017.5  11016.6
```

```
object2$cv[100]
```

```
## [1] 11016.6
```

```
object2$index[100]
```

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

```
mycoefs <- predict(object1, type = "coefficients", s= object2$index[100], mode = "fraction")$coefficien
mycoefs
```

```
##            Income             Limit            Rating             Cards
##         -7.825697          0.203155          0.965542         17.184767
##               Age         Education       GenderFemale        StudentYes
##         -0.755222         -3.021395        -13.804564        418.872974
##        MarriedYes     EthnicityAsian EthnicityCaucasian
##         -7.726900         15.224450         14.155220
```

```
lasso_pred <- predict(object1, newx = x_test, s= object2$index[100], mode = "fraction")
mean((lasso_pred$fit - y_test)^2)
```

```
## [1] 8405.16
```

This model chosen by CV error via the lasso includes all predictors (though it is still doing some shrinkage).

The test MSE is 8405.16.

Based on the plots, it starts plateauing a bit before this, so you could potentially choose a different model (lower fraction).

(e) Fit an elastic-net penalty with alpha = 0.5, and choose a model based on an appropriate descriptive statistic. Compute the test MSE.

SOLUTION

To fit an elastic-net penalty, we need to go use glmnet.

```
set.seed(1)
cv.outen <- cv.glmnet(x_train, y_train, alpha = 0.5) # example net penalty
bestlamen <- cv.outen$lambda.min  # Select lambda that minimizes training MSE
bestlamen
```

```
## [1] 0.621852
```

```
net_mod <- glmnet(x_train, y_train, alpha = 0.5)

net_pred <- predict(net_mod, s = bestlamen, newx = x_test) # Use best lambda
mean((net_pred - y_test)^2) # Calculate test MSE
```

```
## [1] 8340.31
```

```
predict(net_mod, type = "coefficients", s = bestlamen)[1:12,]
```

```
##         (Intercept)              Income               Limit              Rating
##        -447.384069          -7.764973           0.173172           1.398620
##               Cards                 Age           Education       GenderFemale
##          15.035737          -0.762493          -2.848582         -13.495530
##          StudentYes         MarriedYes     EthnicityAsian EthnicityCaucasian
##         416.237701          -7.905319          14.156101          12.324162
```

The test MSE is 8340.31.

(f) How do your models compare? Which model do you prefer? Why? (Should be able to compare coefficients.)

SOLUTION

The MSEs are very similar, but the adjmod we had above that didn't use BOTH Limit and Rating is my personal preference because we removed the issue with multicollinearity encountered.

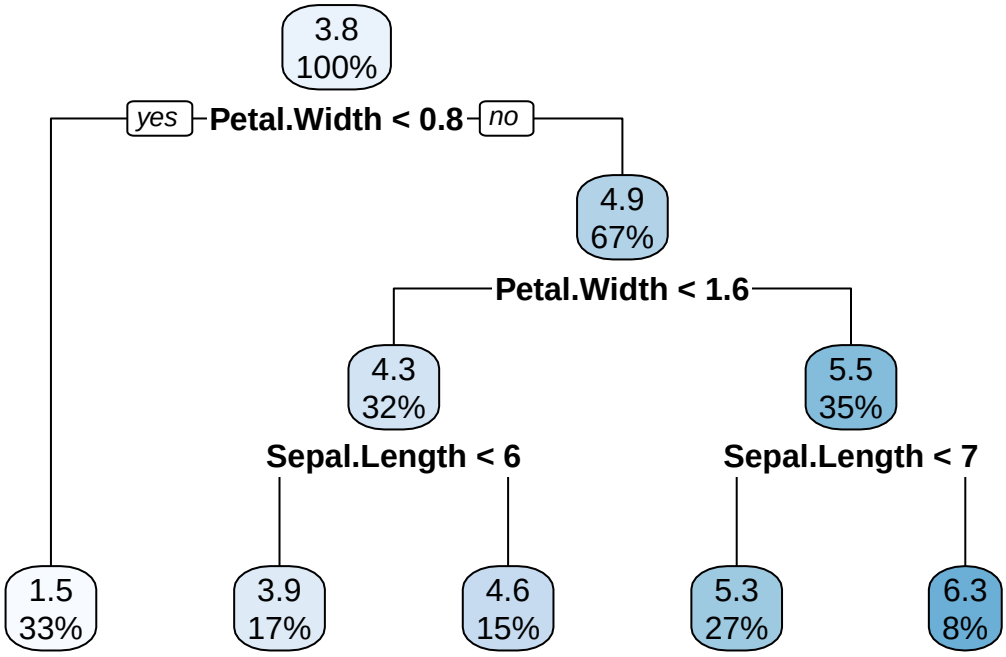
(Note, there is more going on here, with a huge problem with Balance values = 0, so we probably should strategize a bit differently. It may be better to do two procedures - first a logistic regression to decide if Balance is 0 or greater than 0, and then a model to predict Balances greater than 0, but this works for a practice problem to run the methods and examine the code.)

Add 9

```
data(iris)
glimpse(iris)
```

```
## Rows: 150
## Columns: 5
## $ Sepal.Length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4, 4.~
## $ Sepal.Width  <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.~
## $ Petal.Length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1.~
## $ Petal.Width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0.~
## $ Species      <fct> setosa, setosa, setosa, setosa, setosa, setosa, setosa, s~
```

```
# for parts a and b
iris.control <- rpart.control(minbucket = 10, minsplit = 30)
iris.rpart <- rpart(Petal.Length ~ . - Species, data = iris, method = "anova",
                    control = iris.control)
rpart.plot(iris.rpart)
```




```
# for part c
iris[121,]
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 121          6.9         3.2          5.7         2.3 virginica
```

```
# for part d sketch
favstats(~ Petal.Width, data = iris)
```

```
##  min  Q1 median  Q3 max    mean       sd   n missing
##  0.1 0.3    1.3 1.8 2.5 1.19933 0.762238 150       0
```

```
favstats(~ Sepal.Length, data = iris)
```

```
##  min  Q1 median  Q3 max    mean       sd   n missing
##  4.3 5.1    5.8 6.4 7.9 5.84333 0.828066 150       0
```

(a) Is this a classification or a regression tree? How do you know?

SOLUTION

This is a regression tree. We know the response is numeric and we see method = "anova" in the rpart command.

(b) Explain what the minbucket and minsplit control options do.

SOLUTION

Minbucket and minsplit are both control options to govern how the tree is created. Minbucket says that when a split occurs, the resulting nodes must have at least 10 observations in them. Minsplit says that in order for a split to occur, the node being split must have at least 30 observations in it. Changing these values will cause different trees to be built.

(c) What is the residual for observation 121? (Residual = observed - predicted response value)

SOLUTION

In order to get the residual, we have to track the observation through the tree. Observation 121 has a Petal.Width of 2.3, so it goes right at the first split, and then right at the second split. At the final split, we need it's Sepal.Length, which is 6.9. That is less than 7, so it goes left at that split and ends up in the node with a response prediction of 5.3 (27 percent of observations are in that node).

To get the residual, we note it's actual Petal.Length which is 5.7. So our residual is 0.4.

```
5.7-5.3
```

```
## [1] 0.4
```

(d) Only 2 variables are used in the tree. Sketch the hypercubes formed in 2-D space, labeling them with their predicted response values.

SOLUTION

Created via iPad and attached to compiled pdf of solution.