ORF 350: Assignment 3

David Fan

3/27/2017

Collaborator: Brandon Tan

Question 1: Conceptual Challenges (15 points)

(1) B, C, D, E
(2) A, B, C
(3)
(4) A, C
(5) A, F

Question 2: Short Question (10 points)

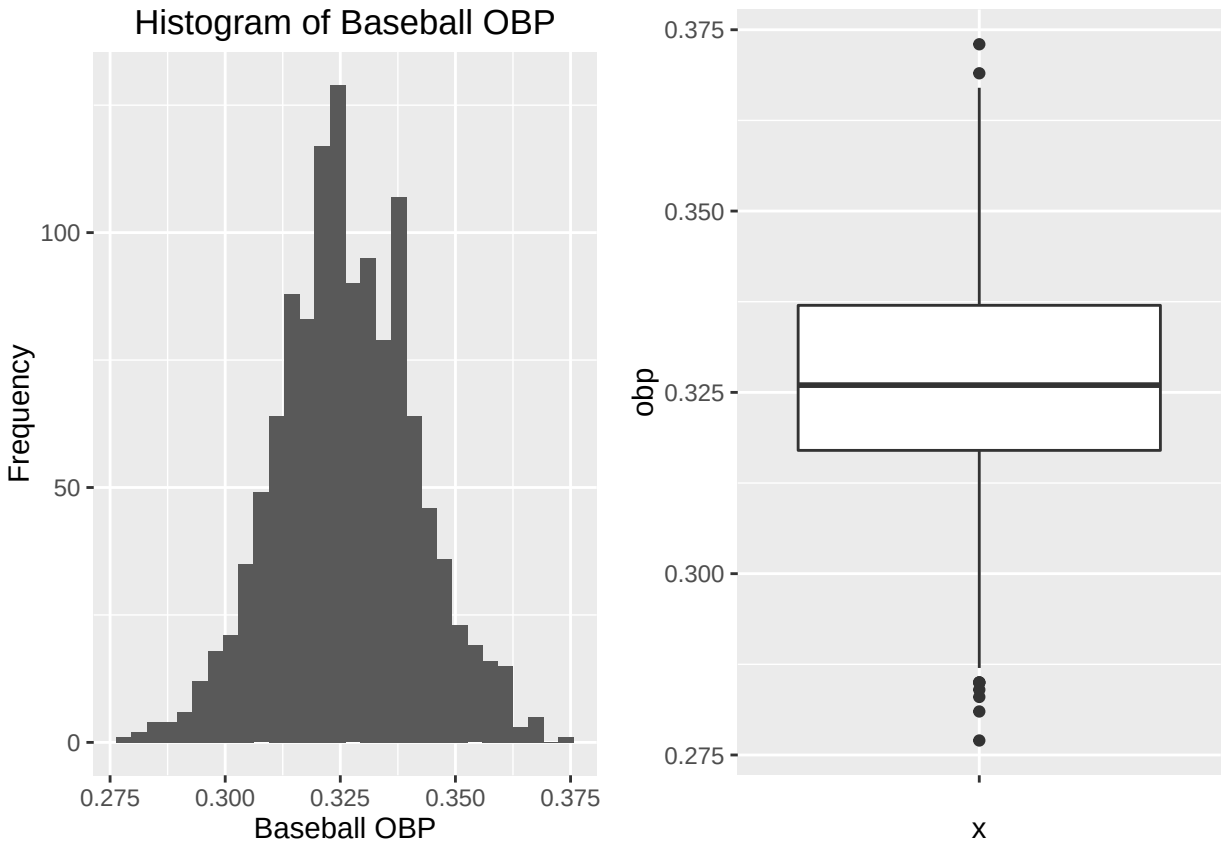
Proof 1: If $X_n \xrightarrow{P} X$ and $Y_n \xrightarrow{P} Y$, then $X_n + Y_n \xrightarrow{P} X + Y$.
If this is true, then $\lim_{n\to\infty} P(|X_n + Y_n - X - Y| \geq \epsilon) = 0$. By the triangle inequality, $|X_n + Y_n - X - Y| \leq |X_n - X| + |Y_n - Y|$. Thus, $\lim_{n\to\infty} P(|X_n + Y_n - X - Y| \geq \epsilon) \leq \lim_{n\to\infty} P(|X_n - X| + |Y_n - Y| \geq \epsilon)$. If $|X_n - X| + |Y_n - Y| \geq \epsilon$, it must be the case that either $|X_n - X| \geq \frac{\epsilon}{2}$ or $|Y_n - Y| \geq \frac{\epsilon}{2}$. So, $\lim_{n\to\infty} P(|X_n - X| + |Y_n - Y| \geq \epsilon) = \lim_{n\to\infty} P(|X_n - X| \geq \frac{\epsilon}{2}) + \lim_{n\to\infty} P(|Y_n - Y| \geq \frac{\epsilon}{2})$. From the definition of convergence in probability, we know $\lim_{n\to\infty} P(|X_n - X| \geq \frac{\epsilon}{2}) = 0$ and $\lim_{n\to\infty} P(|Y_n - Y| \geq \frac{\epsilon}{2}) = 0$ (true for any arbitrarily small epsilon). Thus, $\lim_{n\to\infty} P(|X_n + Y_n - X - Y| \geq \epsilon) \leq 0$. But since probabilities can't be less than 0, $\lim_{n\to\infty} P(|X_n + Y_n - X - Y| \geq \epsilon) = 0$.

Proof 2: If $X_n \xrightarrow{D} X$ and $Y_n \xrightarrow{D} Y$, then $X_n + Y_n \xrightarrow{D} X + Y$.
Counter-example: Suppose $X_n \sim N(0, 1)$ and $Y_n \sim -X_n$. $X_n \to X \sim N(0, 1)$ and $Y_n \to Y \sim N(0, 1)$. $X_n + Y_n \to 0$ but $X + Y \to N(0, \sqrt{2})$.

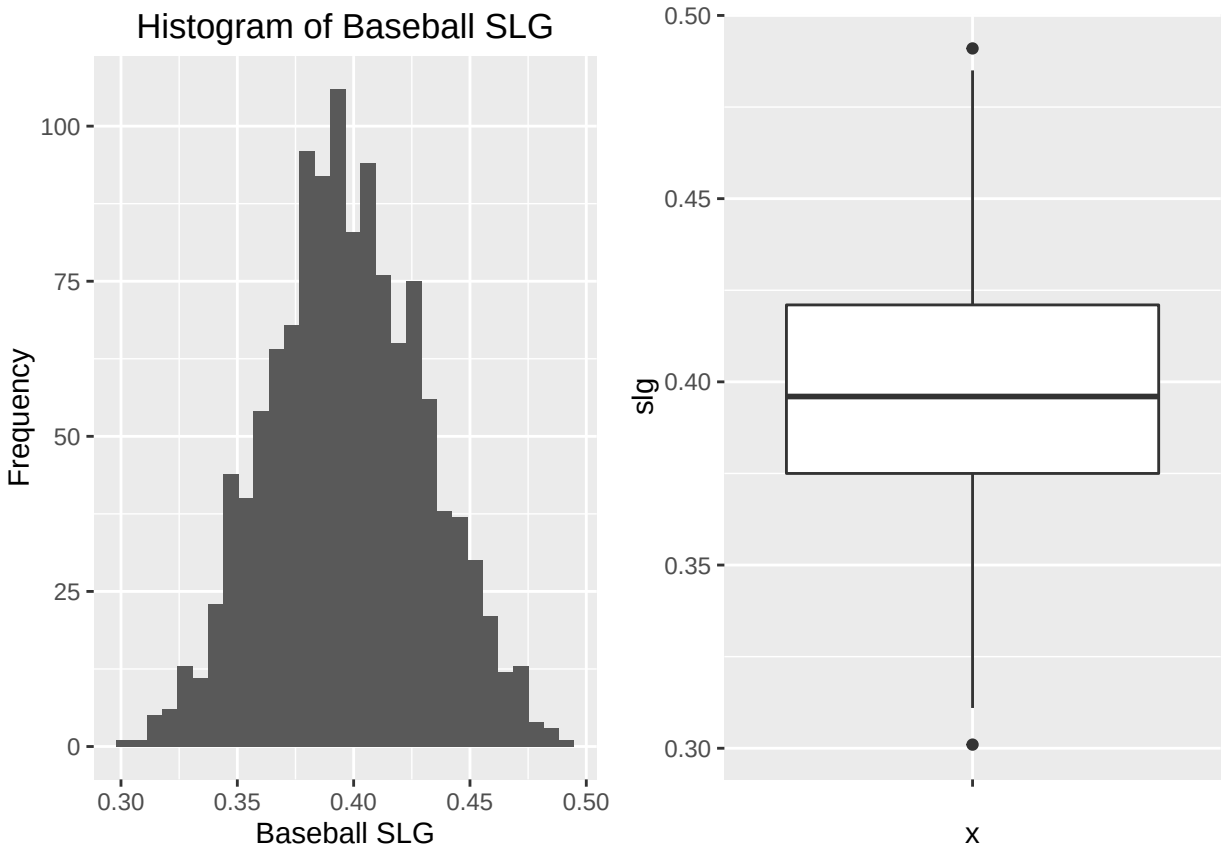
Proof 3: If $X_n \xrightarrow{D} C$ for some constant C, then $X_n \xrightarrow{P} C$.
$\lim_{n\to\infty} P(|X_n - C| \geq \epsilon) = \lim_{n\to\infty} P(X_n \geq C - \epsilon) + \lim_{n\to\infty} P(X_n \leq C + \epsilon) = 1 - \lim_{n\to\infty} P(X_n \leq C - \epsilon) - \lim_{n\to\infty} P(X_n \leq C + \epsilon)$. We know $\lim_{n\to\infty} P(X_n \leq C - \epsilon) = \lim_{n\to\infty} F_{X_n}(C - \epsilon) = 1$ and $\lim_{n\to\infty} P(X_n \leq C + \epsilon) = \lim_{n\to\infty} F_{X_n}(C + \epsilon) = 0$. Thus, $\lim_{n\to\infty} P(|X_n - C| \geq \epsilon) = 0$ and thus $X_n \xrightarrow{P} C$.

Question 3: Moneyball: The Analytic Edge in Sports (15 points)

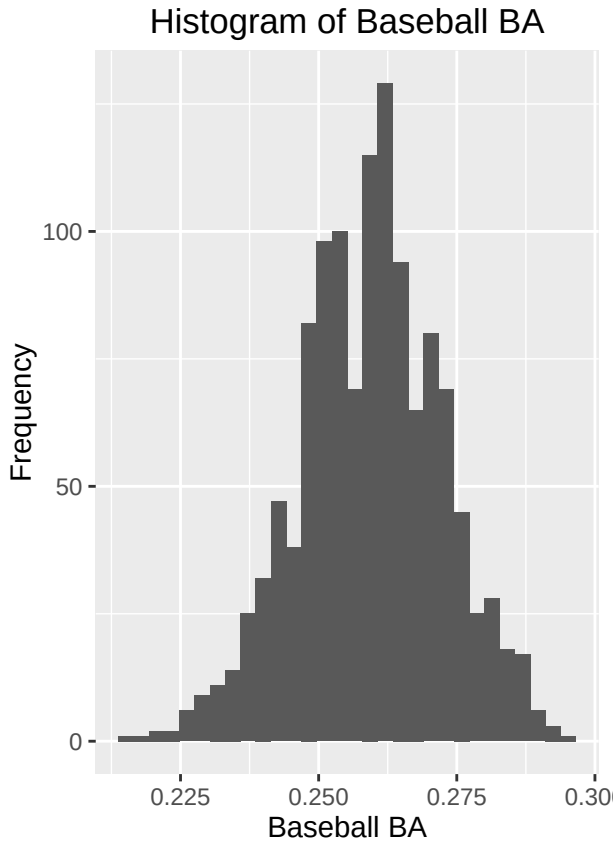
```
setwd("/Users/dfan/Dropbox/School/Sophomore Year/Spring 2017/ORF 350/Assignments/HW3")
load("moneyball.rda")

# 3.1)
obp <- data.frame(moneyball$OBP)
slg <- data.frame(moneyball$SLG)
ba <- data.frame(moneyball$BA)

plot1 <- ggplot(data = obp, aes(obp)) + geom_histogram() + xlab("Baseball OBP") +
    ylab("Frequency") + ggtitle("Histogram of Baseball OBP")
plot2 <- ggplot(data = obp, aes(x = "", y = obp)) + geom_boxplot()
grid.arrange(plot1, plot2, ncol = 2)
```

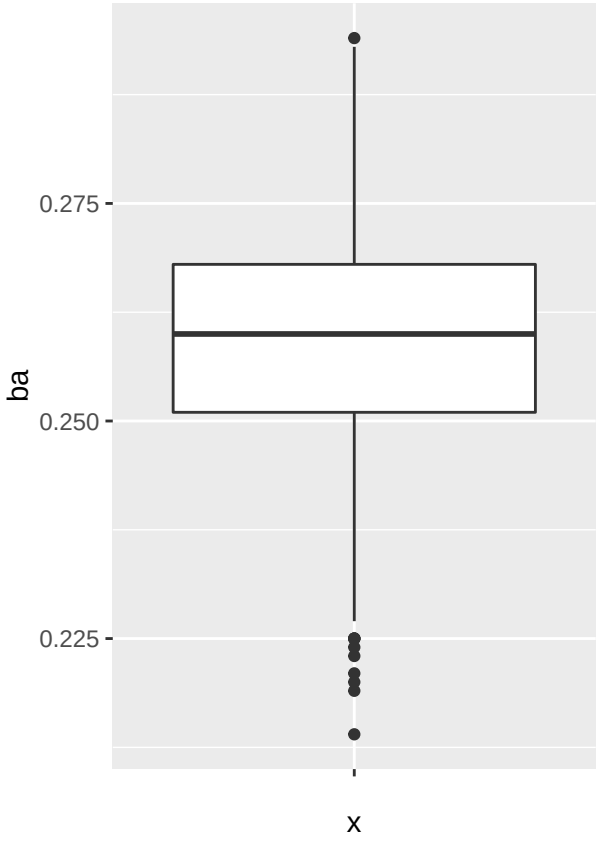



```
plot1 <- ggplot(data = slg, aes(slg)) + geom_histogram() + xlab("Baseball SLG") +
    ylab("Frequency") + ggtitle("Histogram of Baseball SLG")
plot2 <- ggplot(data = slg, aes(x = "", y = slg)) + geom_boxplot()
grid.arrange(plot1, plot2, ncol = 2)
```




```
plot1 <- ggplot(data = ba, aes(ba)) + geom_histogram() + xlab("Baseball BA") +
    ylab("Frequency") + ggtitle("Histogram of Baseball BA")
plot2 <- ggplot(data = ba, aes(x = "", y = ba)) + geom_boxplot()
grid.arrange(plot1, plot2, ncol = 2)
```






```
paste("OBP mean:", mean(moneyball$OBP), ", OBP median:", median(moneyball$OBP))
```

```
## [1] "OBP mean: 0.326331168831169 , OBP median: 0.326"
```

```
paste("SLG mean:", mean(moneyball$SLG), ", SLG median:", median(moneyball$SLG))
```

```
## [1] "SLG mean: 0.397341720779221 , SLG median: 0.396"
```

```
paste("BA mean:", mean(moneyball$BA), ", BA median:", median(moneyball$BA))
```

```
## [1] "BA mean: 0.259272727272727 , BA median: 0.26"
```

As demonstrated by the histograms and boxplots, the distribution of OBP, SLG and BA are not skewed since the mean of each quantity is essentially equal to the median of each quantity.

```
# 3.2)
moneyball_train <- moneyball[which(moneyball$Year <= 1997), ]
model_one <- lm(R ~ BA, data = moneyball_train)
coef(model_one)
```

```
## (Intercept)          BA
##   -697.6089   5401.8102
```

```
summary(model_one)$r.squared
```

```
## [1] 0.6684324
```

```
model_two <- lm(R ~ OBP + SLG, data = moneyball_train)
coef(model_two)
```

```
## (Intercept)         OBP         SLG
```

```
##   -782.1721   2594.0436   1648.2667
```

```
summary(model_two)$r.squared
```

```
## [1] 0.9161502
```

I would pick the second model that predicts R using both OBP and SLG, based on the R-squared values.

```
# 3.3)
moneyball_train <- moneyball[which(moneyball$Year >= 1999 & moneyball$Year <=
    2011), ]
model_r <- lm(R ~ OBP + SLG, data = moneyball_train)
model_ra <- lm(RA ~ OOBP + OSLG, data = moneyball_train)
model_w <- lm(W ~ R + RA, data = moneyball_train)
moneyball_test <- moneyball[which(moneyball$Year == 2012), ]
moneyball_test$R <- predict(model_r, moneyball_test)
moneyball_test$RA <- predict(model_ra, moneyball_test)
predicted_wins <- predict(model_w, moneyball_test)
df <- data.frame(predicted_wins, moneyball_test$W)
names(df) <- c("Predicted", "Actual")
cor(predicted_wins, moneyball_test$W, method = "pearson")
```

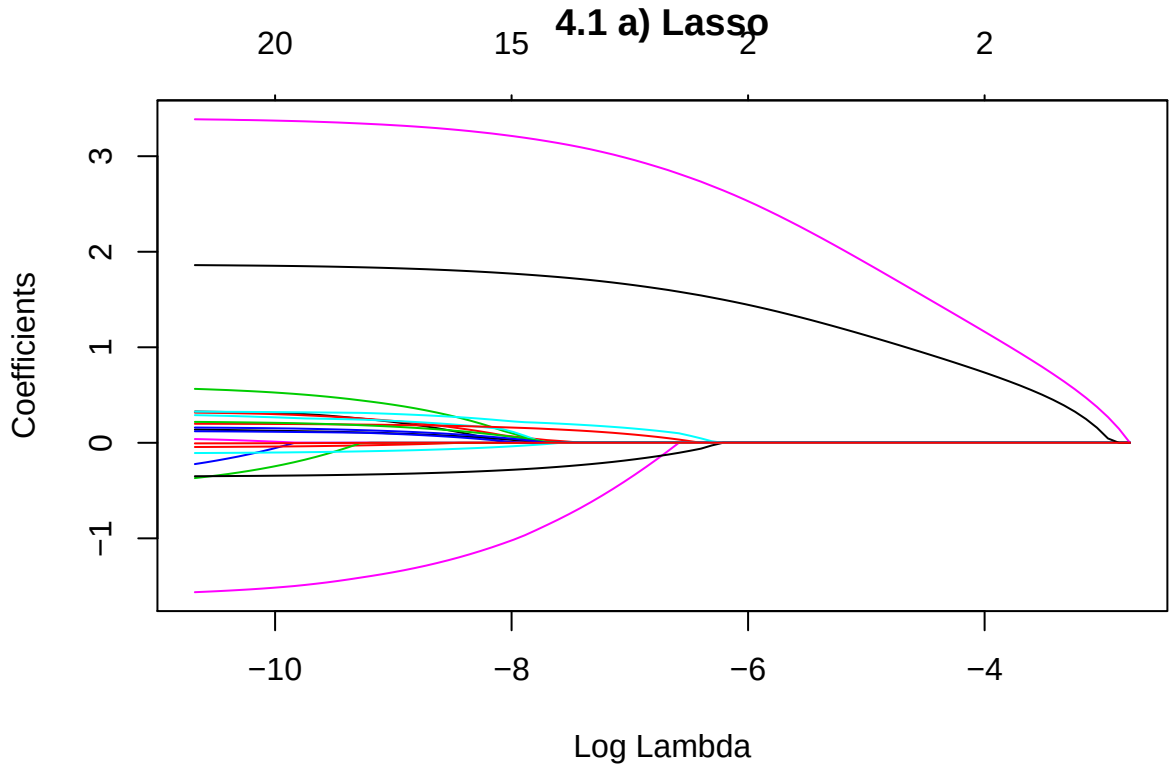
```
## [1] 0.9199287
```

Question 4: Big Data. Big Profit. (35 points)

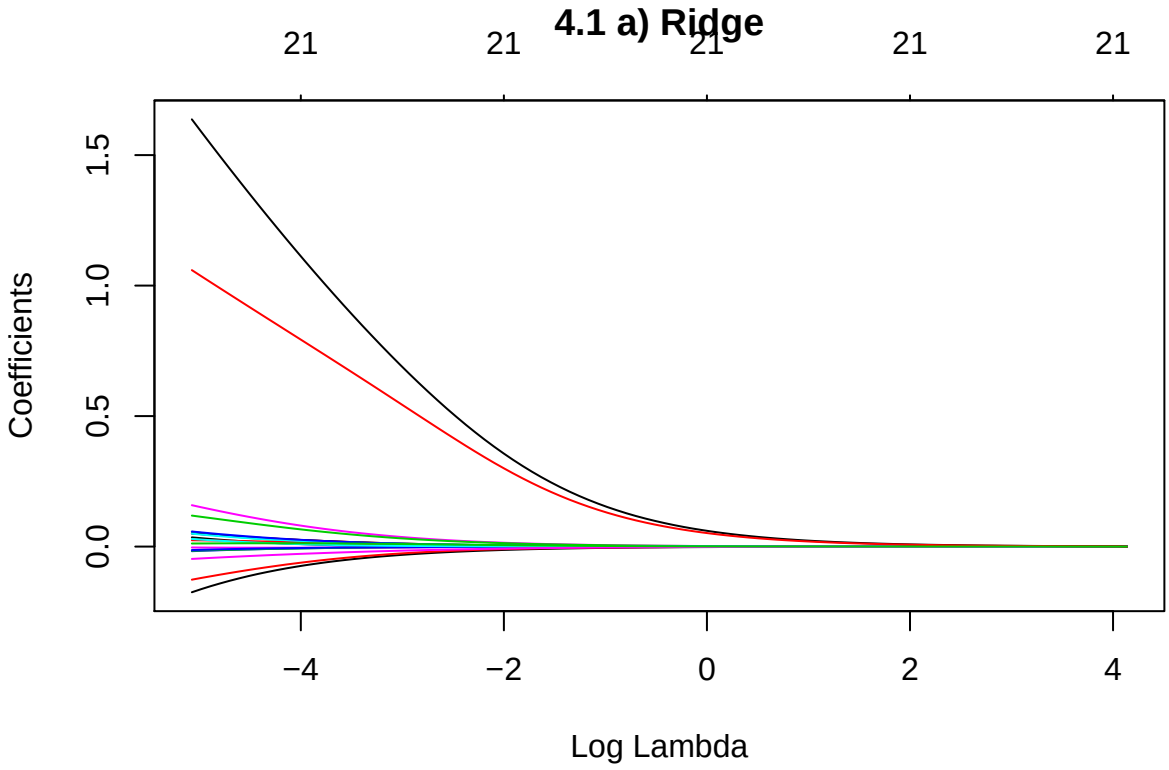
4.1 Click Me! Click Me! (20 points)

```
# 4.1 Preprocessing
load("iPinYou.RData")  # loads dataTest, dataTestRes and dataTrainAll objects
features <- c("Region", "City", "AdX", "Domain", "Key_Page",
    "Ad_Vis", "Ad_Form")
for (i in 1:length(features)) {
    dataTrainAll[, features[i]] <- factor(dataTrainAll[, features[i]])
}

# Converting to indicator variables Implicit category for
# Region is 1
training_data <- data.frame(model.matrix(~Region, data = dataTrainAll)[,
    -1])
# Implicit category for City is 1
training_data <- cbind(training_data, model.matrix(~City, data = dataTrainAll)[,
    -1])
# Implicit category for AdX is 1
training_data <- cbind(training_data, model.matrix(~AdX, data = dataTrainAll)[,
    -1])
# Implicit category for Domain is 5Fa-expoBTTR1m58uG
training_data <- cbind(training_data, model.matrix(~Domain, data = dataTrainAll)[,
    -1])
# Implicit category for Key_page is
# 3a7eb50444df6f61b2409f4e2f16b687
training_data <- cbind(training_data, model.matrix(~Key_Page,
    data = dataTrainAll)[, -1])
# Implicit category for Ad_Vis is 0
training_data <- cbind(training_data, model.matrix(~Ad_Vis, data = dataTrainAll)[,
    -1])
# Implicit category for Ad_Form is 0
training_data <- cbind(training_data, model.matrix(~Ad_Form,
    data = dataTrainAll)[, -1, drop = FALSE])

# Standardize remaining three columns
training_data <- cbind(training_data, Ad_Width = dataTrainAll$Ad_Width)
training_data <- cbind(training_data, Ad_Height = dataTrainAll$Ad_Height)
training_data <- cbind(training_data, Floor_Price = dataTrainAll$Floor_Price)
# only because RMarkdown isn't wrapping lines correctly...
diff <- (training_data$Ad_Width - mean(training_data$Ad_Width))
training_data$Ad_Width <- diff/sd(training_data$Ad_Width)
diff <- (training_data$Ad_Height - mean(training_data$Ad_Height))
training_data$Ad_Height <- diff/sd(training_data$Ad_Height)
diff <- (training_data$Floor_Price - mean(training_data$Floor_Price))
training_data$Floor_Price <- diff/sd(training_data$Floor_Price)

# Preprocess Click column
training_data <- cbind(training_data, Click = sapply(dataTrainAll$Click,
    function(x) {
        if (x >= 1) {
            return(1)
```

```
        } else {
            return(0)
        }
    }))

# 4.1a)
lasso_model <- glmnet(as.matrix(training_data[, 1:dim(training_data)[2] -
    1]), training_data$Click, family = "binomial", alpha = 1,
    standardize = FALSE)
ridge_model <- glmnet(as.matrix(training_data[, 1:dim(training_data)[2] -
    1]), training_data$Click, family = "binomial", alpha = 0,
    standardize = FALSE)
plot(lasso_model, xvar = "lambda", main = "4.1 a) Lasso")
```




```
plot(ridge_model, xvar = "lambda", main = "4.1 a) Ridge")
```

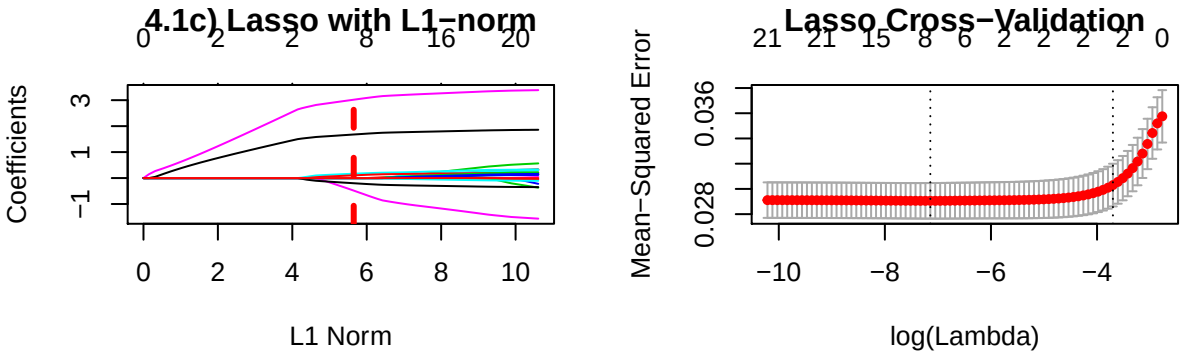


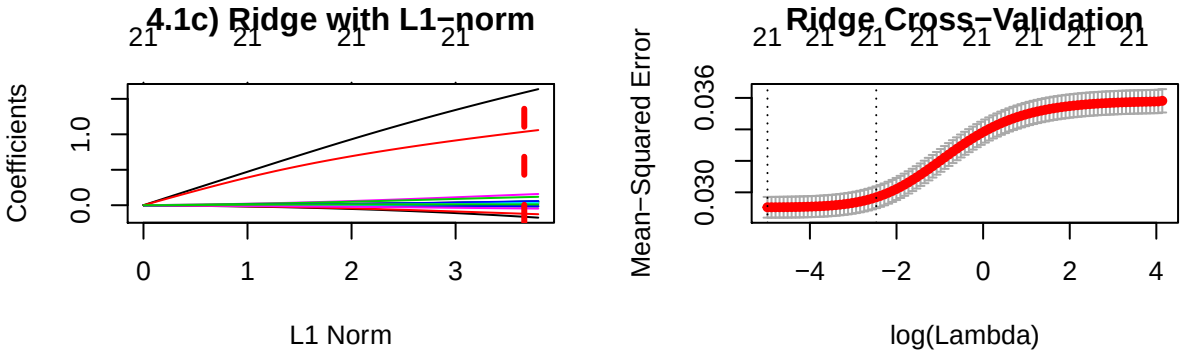
It seems the two most important features are Ad_Width and Ad_Height, since they have the highst coefficients (confirmed by *coef(lasso_model, s=-1) and coef(ridge_model, s=-1))* and thus decay to zero the slowest as lambda gets largest. Bigger ads are easier to see! I used model.matrix() to convert each column to multiple columns of indicator variables. I indicated the implicit category for each categorical variable in the comments of my code. I chose to make the x-axis log(lambda) for part a just so I could visualize the coefficients as a function of lambda more clearly. The same two plots with L1 norm as the x-axis instead can be found in 4.1c).

```
# 4.1c)
cv_lasso <- cv.glmnet(as.matrix(training_data[, 1:dim(training_data)[2] -
    1]), training_data$Click, nfolds = 5, alpha = 1, standardize = FALSE)
cv_ridge <- cv.glmnet(as.matrix(training_data[, 1:dim(training_data)[2] -
    1]), training_data$Click, nfolds = 5, alpha = 0, standardize = FALSE)
lasso_lambda <- cv_lasso$lambda.min
lasso_l1norm <- sum(abs(coef(lasso_model, s = lasso_lambda)[2:22]))
ridge_lambda <- cv_ridge$lambda.min
ridge_l1norm <- sum(abs(coef(ridge_model, s = ridge_lambda)[2:22]))
par(mfrow = c(2, 2))
plot(lasso_model, main = "4.1c) Lasso with L1-norm")
abline(v = lasso_l1norm, col = "red", lty = 2, lwd = 3)
plot(cv_lasso, main = "Lasso Cross-Validation")
paste("L1 norm for lasso:", lasso_l1norm)
```

```
## [1] "L1 norm for lasso: 5.65013795956413"
```

```
plot(ridge_model, main = "4.1c) Ridge with L1-norm")
abline(v = ridge_l1norm, col = "red", lty = 2, lwd = 3)
plot(cv_ridge, main = "Ridge Cross-Validation")
```






```
paste("L1 norm for ridge:", ridge_l1norm)
```

```
## [1] "L1 norm for ridge: 3.65987224025813"
```

The two graphs from 4.1 show how the coefficients for each observation variable change as L1 Norm increases. The optimum L1 Norm is smaller for lasso than it is for ridge, because lasso enforces sparcity. While few coefficients are non-zero in the lasso model, more are non-zero in the ridge model and thus the optimal L1 Norm is higher. In the two plots on the right half, the top axis shows the degrees of freedom or number of non-zero coefficients for a given lambda value. We see that as degrees of freedom grow smaller (as the coefficients shrink), the MSE grows larger. Thus, as the regression becomes more regularized with higher values of lambda, the less overfitting we see and the greater the MSE.

```
# Q4.1d) format the testing data and standardize it
for (i in 1:length(features)) {
    dataTest[, features[i]] <- factor(dataTest[, features[i]])
}
# Converting to indicator variables Implicit category for
# Region is 1
testing_data <- data.frame(model.matrix(~Region, data = dataTest)[,
    -1])
# Implicit category for City is 1
testing_data <- cbind(testing_data, model.matrix(~City, data = dataTest)[,
    -1])
# Implicit category for AdX is 1
testing_data <- cbind(testing_data, model.matrix(~AdX, data = dataTest)[,
    -1])
# Implicit category for Domain is 5Fa-expoBTTR1m58uG
testing_data <- cbind(testing_data, model.matrix(~Domain, data = dataTest)[,
    -1])
# Implicit category for Key_page is
```

```
# 3a7eb50444df6f61b2409f4e2f16b687
testing_data <- cbind(testing_data, model.matrix(~Key_Page, data = dataTest)[,
    -1])
# Implicit category for Ad_Vis is 0
testing_data <- cbind(testing_data, model.matrix(~Ad_Vis, data = dataTest)[,
    -1])
# Implicit category for Ad_Form is 0
testing_data <- cbind(testing_data, model.matrix(~Ad_Form, data = dataTest)[,
    -1, drop = FALSE])

# Standardize remaining three columns
testing_data <- cbind(testing_data, Ad_Width = dataTest$Ad_Width)
testing_data <- cbind(testing_data, Ad_Height = dataTest$Ad_Height)
testing_data <- cbind(testing_data, Floor_Price = dataTest$Floor_Price)

# only because RMarkdown isn't wrapping lines correctly...
diff <- (testing_data$Ad_Width - mean(dataTrainAll$Ad_Width))
testing_data$Ad_Width <- diff/sd(dataTrainAll$Ad_Width)
diff <- (testing_data$Ad_Height - mean(dataTrainAll$Ad_Height))
testing_data$Ad_Height <- diff/sd(dataTrainAll$Ad_Height)
diff <- (testing_data$Floor_Price - mean(dataTrainAll$Floor_Price))
testing_data$Floor_Price <- diff/sd(dataTrainAll$Floor_Price)

# Preprocess Click column
testing_data <- cbind(testing_data, Click = sapply(dataTestRes$Click,
    function(x) {
        if (x >= 1) {
            return(1)
        } else {
            return(0)
        }
    }))

# Predictions:
lasso_predictions <- as.numeric(predict(lasso_model, as.matrix(testing_data[,
    -22]), type = "class", s = lasso_lambda))
lasso_truepositive <- length(intersect(which(lasso_predictions ==
    1), which(testing_data$Click == 1)))/length(which(testing_data$Click ==
    1))
lasso_truenegative <- length(intersect(which(lasso_predictions ==
    0), which(testing_data$Click == 0)))/length(which(testing_data$Click ==
    0))
paste("Prediction accuracy for yi = 1 in lasso:", lasso_truepositive)
```

```
## [1] "Prediction accuracy for yi = 1 in lasso: 0.270186335403727"
```

```
paste("Prediction accuracy for yi = 0 in lasso:", lasso_truenegative)
```

```
## [1] "Prediction accuracy for yi = 0 in lasso: 0.997623475924778"
```

```
ridge_predictions <- as.numeric(predict(ridge_model, as.matrix(testing_data[,
    -22]), type = "class", s = ridge_lambda))
ridge_truepositive <- length(intersect(which(ridge_predictions ==
    1), which(testing_data$Click == 1)))/length(which(testing_data$Click ==
    1))
```

```
ridge_truenegative <- length(intersect(which(ridge_predictions ==
    0), which(testing_data$Click == 0)))/length(which(testing_data$Click ==
    0))
paste("Prediction accuracy for yi = 1 in ridge:", ridge_truepositive)
```

```
## [1] "Prediction accuracy for yi = 1 in ridge: 0.214285714285714"
```

```
paste("Prediction accuracy for yi = 0 in ridge:", ridge_truenegative)
```

```
## [1] "Prediction accuracy for yi = 0 in ridge: 0.998553420128126"
```

4.2 Know The Enemy (15 points)

```
training_data <- data.frame(AdX = dataTrainAll$AdX, iPinYou_Bid = dataTrainAll$iPinYou_Bid,
    Comp_Bid = dataTrainAll$Comp_Bid)
for (i in 1:ncol(training_data)) {
    training_data[, i] <- as.numeric(training_data[, i])
    training_data[, i] <- (training_data[, i] - mean(training_data[,
        i]))/sd(training_data[, i])
}
lm_model <- lm(Comp_Bid ~ AdX + iPinYou_Bid + 0, training_data)  # no intercept
mle_l1norm <- sum(abs(coef(lm_model)))
paste("MLE Coefficients:")
```

```
## [1] "MLE Coefficients:"
```

```
coef(lm_model)
```

```
##         AdX iPinYou_Bid
##  -0.1664490   0.7721763
```

```
lasso_model <- glmnet(as.matrix(training_data[, -ncol(training_data)]),
    training_data$Comp_Bid, family = "gaussian", alpha = 1, standardize = FALSE,
    intercept = FALSE)
lasso_coef <- coef(lasso_model)[, which.min(abs(colSums(coef(lasso_model),
    na.rm = TRUE, sparseResult = TRUE) - mle_l1norm/2))]
paste("Lasso Coefficients:")
```

```
## [1] "Lasso Coefficients:"
```

```
lasso_coef
```

```
## (Intercept)         AdX iPinYou_Bid
##   0.0000000   0.0000000   0.4734639
```

```
# discretized grid
beta1 <- seq(-0.5, 1, length.out = 100)
beta2 <- seq(-0.5, 1, length.out = 100)
mse <- matrix(rep(0, 10000), ncol = 100)
for (i in 1:length(beta1)) {
    for (j in 1:length(beta2)) {
        mse[i, j] <- sum((training_data$Comp_Bid - (beta1[i] *
            training_data$AdX + beta2[j] * training_data$iPinYou_Bid))^2)
    }
}
```

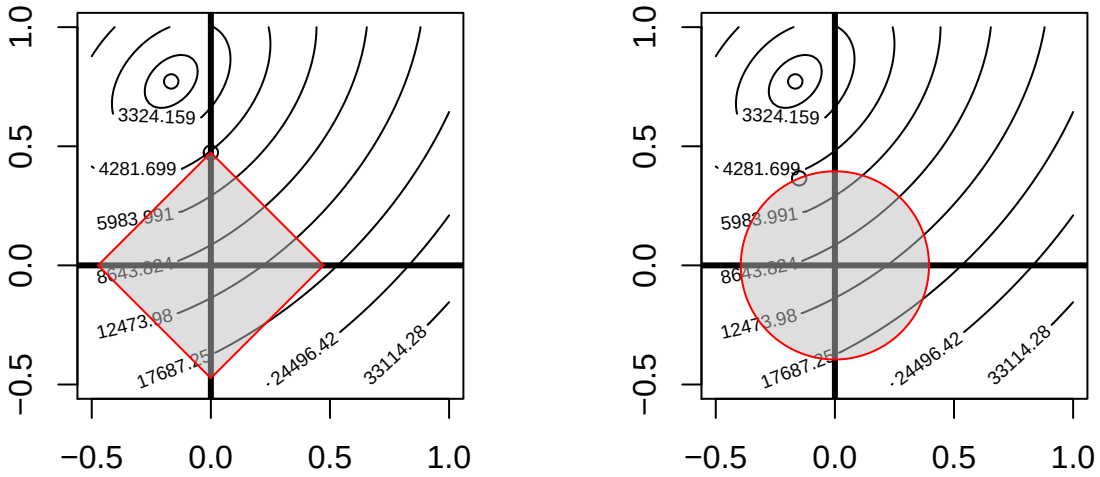
```
par(mfrow = c(1, 2), pty = "s")
seqz = seq(min(mse), max(mse), length.out = 386)[c(2, 6, 15,
    31, 56, 92, 141, 205, 286)]  # handpicked to match lines in Figure 2 of handout
contour(beta1, beta2, mse, levels = seqz)
abline(h = 0, lwd = 3)
abline(v = 0, lwd = 3)
points(c(coef(lm_model)[1], lasso_coef[2]), c(coef(lm_model)[2],
    lasso_coef[3]))
diamondX <- c(-lasso_coef["iPinYou_Bid"], 0, lasso_coef["iPinYou_Bid"],
    0)
diamondY <- c(0, lasso_coef["iPinYou_Bid"], 0, -lasso_coef["iPinYou_Bid"])
polygon(diamondX, diamondY, col = rgb(0.75, 0.75, 0.75, 0.5),
    border = "red")

# ridge
ridge_model <- glmnet(as.matrix(training_data[, -ncol(training_data)]),
    training_data$Comp_Bid, family = "gaussian", alpha = 0, standardize = FALSE,
    intercept = FALSE)
mle_l2norm <- sqrt(sum(coef(lm_model)^2))
ridge_coef <- coef(ridge_model)[, which.min(abs(sqrt(colSums(coef(ridge_model)^2,
    na.rm = TRUE, sparseResult = TRUE)) - mle_l2norm/2))]
paste("Ridge Coefficients:")
```

```
## [1] "Ridge Coefficients:"
```

```
ridge_coef
```

```
## (Intercept)         AdX iPinYou_Bid
##   0.0000000  -0.1500652   0.3654805
```

```
seqz = seq(min(mse), max(mse), length.out = 386)[c(2, 6, 15,
    31, 56, 92, 141, 205, 286)]
contour(beta1, beta2, mse, levels = seqz, asp = 1)
abline(h = 0, lwd = 3)
abline(v = 0, lwd = 3)
points(c(coef(lm_model)[1], ridge_coef[2]), c(coef(lm_model)[2],
    ridge_coef[3]))
radius <- sqrt(ridge_coef[2]^2 + ridge_coef[3]^2)
draw.circle(0, 0, radius, col = rgb(0.75, 0.75, 0.75, 0.5), border = "red")
```



MLE point: (-0.1664490, 0.7721763)

Q: In relationship to the level curves, where do the MLE coefficients sit?
The MLE coefficients lie in the center of the level curves.
Q: What does this say about the MSE of the MLE coefficients?
This shows that the MLE coefficients minimize the MSE.
Q: In relationship to the level curves and the L1-ball (or L2-ball), where do the Lasso (or Ridge) coefficients sit?
The lasso and ridge coefficients sit on the edge of the L1 and L2 constraints respectively and intersect with the closest level curve of the MSE.
Q: What does this say about the MSE of the Lasso and Ridge coefficients?
This shows that the lasso and ridge coefficients minimize the MSE with respect to the L1 and L2 constraints. The coefficients lie at the earliest point at which the constraint boundary intersects with a MSE level curve.
Q: Lastly, based on our geometric representation of the L1- and L2-balls, why can we believe that Lasso coefficients favor sparsity over Ridge coefficients?
The lasso solution is more sparse, since the earliest point at which the constraint boundary intersects with a MSE level curve is at one of the four corners of the diamond. In this case, the solution is when $\beta_1 = 0$ so only one coefficient is non-zero. In contrast, the ridge constraint boundary is a circle and less likely to yield a sparce solution. Here, the ridge solution lies a little to the left so both coefficients are non-zero.

Question 5: A Regression by Any Other Form Will Predict Just As Sweetly. (15 points)

5.1 Constrained and Unconstrained Optimization

$(5.1) : \arg\max_{\beta} \frac{1}{2n}||Y - X\beta||_2^2 + \lambda||\beta||_1$

$(5.2) : \arg\max_{\beta} \frac{1}{2n}||Y - X\beta||_2^2$ such that $||\beta||_1 \leq C$

Suppose $\hat{\beta}$ is a minimizer of (5.1) where λ is positive. Assume $\hat{\beta}$ is not a minimizer of (5.2), but there exists a $\tilde{\beta}$ that is a minimizer of (5.2).

Then, $\frac{1}{2n}||Y - X\tilde{\beta}||_2^2 < \frac{1}{2n}||Y - X\hat{\beta}||_2^2$, such that $||\tilde{\beta}||_1 \leq \{C = ||\hat{\beta}||_1\}$, where we chose the constant C to equal $||\hat{\beta}||_1$.

Since $\lambda > 0$, $\lambda||\tilde{\beta}||_1 \leq \lambda||\hat{\beta}||_1$.

Thus, for (5.1), we see that $\frac{1}{2n}||Y - X\tilde{\beta}||_2^2 + \lambda||\tilde{\beta}||_1 < \frac{1}{2n}||Y - X\hat{\beta}||_2^2 + \lambda||\hat{\beta}||_1$. But this contradicts the original statement that $\hat{\beta}$ is a minimizer of (5.1), so for any positive λ, it is possible to pick a constant C such that (5.2) shares the same minimizer as (5.1).

In making our plots, we chose the same constraint of $C = ||\hat{\beta}||_1$. Thus, we were solving the same minimization problem as 5.2: finding the value of β that gets the smallest MSE subject to the constraint, or in otherwords the intersection between the diamond corner and the nearest level curve. Our optimal β is also the optimal solution for (5.1) as we just proved, which is also the function glmnet was minimizing. This is consistent.

5.2 Regularized Regression

$(5.3) : \min_{\beta}\{||y - X\beta||_2^2 + \lambda\alpha||\beta||_2^2 + \lambda(1-\alpha)||\beta||_1\}$

$(5.4) : \min_{\tilde{\beta}}\{||\tilde{y} - \tilde{X}\beta||_2^2 + \tilde{\lambda}||\beta||_1\}$

X is a $n \times p$ matrix, Y is a $n \times 1$ matrix and β is a $p \times 1$ matrix.

Let $\tilde{\lambda} = \lambda(1-\alpha)$, so we don't have to worry about the L1 norm terms in each equation, since they are equivalent.

Expanding (5.3) without the L1 norm term, we get

$$\begin{aligned} ||y - X\beta||_2^2 + \lambda\alpha||\beta||_2^2 &= (y - X\beta)^T(y - X\beta) + \lambda\alpha(\beta^T\beta) \\ &= y^Ty - 2y^TX\beta + \beta^TX^TX\beta + \lambda\alpha(\beta^T\beta) \end{aligned}$$

Expanding (5.4) without the L1 norm term, we get

$$\begin{aligned} ||\tilde{y} - \tilde{X}\beta||_2^2 &= (\tilde{y} - \tilde{X}\beta)^T(\tilde{y} - \tilde{X}\beta)) \\ &= \tilde{y}^T\tilde{y} - 2\tilde{y}^T\tilde{X}\beta + \beta^T\tilde{X}^T\tilde{X}\beta \end{aligned}$$

Let $\tilde{y}$ be a $(n+p) \times 1$ matrix, where the first n entries are the same as y, and the last p entries are 0s.

Let $\tilde{X}$ be a $(n+p) \times p$ matrix, where the first n rows are the same as X, and the last p rows are the pxp matrix $\sqrt{\alpha\lambda} * I_k$. In otherwords, a $p \times p$ matrix where the only non-zero values are the diagonal of entirely $\sqrt{\alpha\lambda}$.

$\tilde{y}^T\tilde{y} = y^T y$, since the last p entries in $\tilde{y}$ don't contribute to the value of this product.

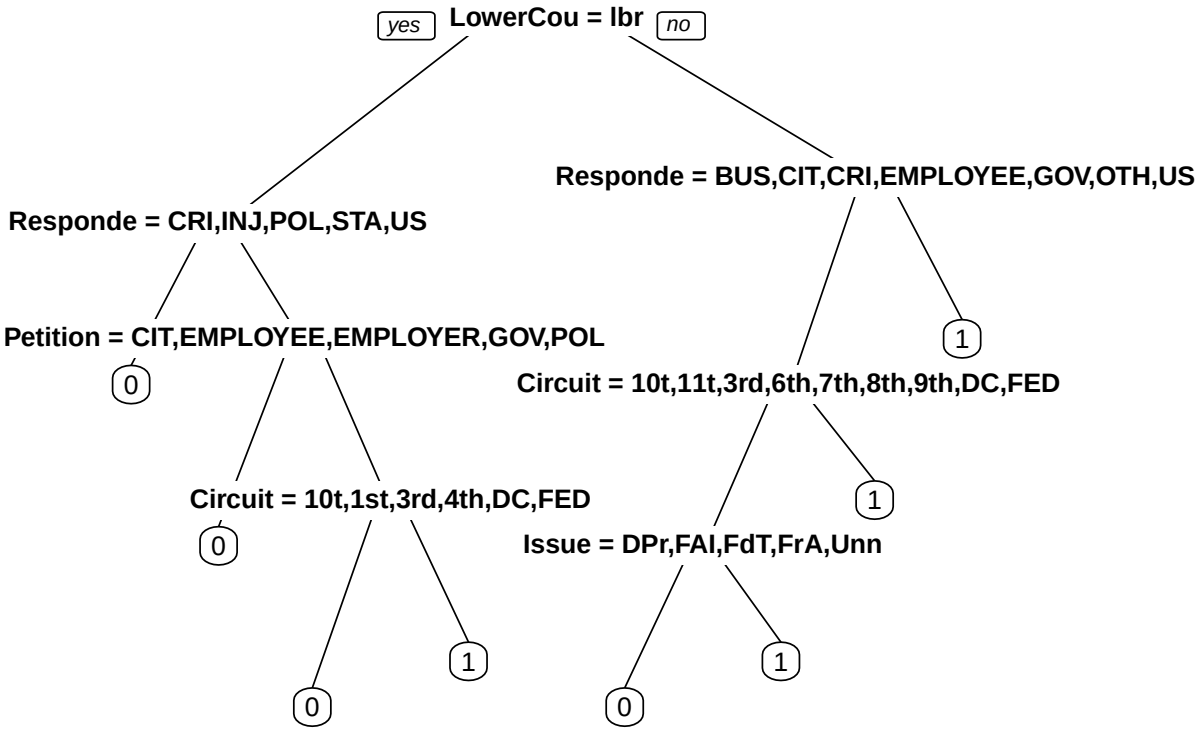
$2\tilde{y}^T\tilde{X}\beta = 2y^T X\beta$, since the last p entries in $\tilde{y}^T$ cancel out the extra p values in each column of $\tilde{X}$.

$\beta^T\tilde{X}^T\tilde{X}\beta = \beta^T X^T X\beta + \alpha\lambda(\beta^T\beta)$, since $\tilde{X}^T\tilde{X}$ is the same as $X^T X$ but with $\alpha\lambda$ added to each entry of the diagonal. $\tilde{X}^T\tilde{X} = X^T X + \alpha\lambda * I_p$.

Thus, we see that the function that we're trying to minimize in (5.4) is exactly the same as the function in (5.3), after making the following modifications to X and y: add the pxp matrix $\sqrt{\alpha\lambda} * I_p$ so that X becomes a $(n+p) \times p$ matrix, and add p zeros to y to become a $(n+p) \times 1$ matrix.

Question 6: Modeling The Decisions of Justice Stevens (10 points)

```
df = read.csv("scotus.csv")

library(caTools)
set.seed(3000)
spl = sample.split(df$Reverse, SplitRatio = 0.7)
df.train = subset(df, spl == TRUE)
df.test = subset(df, spl == FALSE)

df.train$Reverse = as.factor(df.train$Reverse)
df.test$Reverse = as.factor(df.test$Reverse)

# 6.1 Install rpart library install.packages('rpart')
library(rpart)
# install.packages('rpart.plot')
library(rpart.plot)

# CART model
cart.model = rpart(Reverse ~ Circuit + Issue + Petitioner + Respondent +
    LowerCourt + Unconst, data = df.train, method = "class",
    control = rpart.control(minbucket = 25))  # Add something as required by the problem.

prp(cart.model)  # Complete this line
```




```
cart.prediction = predict(cart.model, newdata = df.test, type = "class")  # Complete this line!
error = sum(cart.prediction != df.test$Reverse)/(dim(df.test)[1])
paste("Error:", error)
```

```
## [1] "Error: 0.341176470588235"
```

```r
# 6.2 Random Forest Model

library(randomForest)

set.seed(350)

# Figure out the meaning of k and fill in the value of k
# here!
k = 10

N = dim(df.train)[1]  # Total sample size
n = floor(N/k)  # Split sample size
# error.all records the cross validation error with different
# nodesizes.
error.all = rep(0, 40)
for (j in (1:40)) {
    # Random permutation of all the samples.
    ind = sample(N)
    # error records misclassification error for each split as the
    # testing data within cv.
    error = rep(0, k)
    for (i in 1:k) {
        test = ind[(((i - 1) * n + 1):(min(i * n, N)))]  # Choose the ith split as the testing data
        rf.model = randomForest(Reverse ~ Circuit + Issue + Petitioner +
            Respondent + LowerCourt + Unconst, data = df.train[-test,
            ], ntree = 10, nodesize = j + 10)  # Complete this line!
        rf.prediction = predict(rf.model, newdata = df.train[test,
            1:ncol(df.train) - 1], type = "class")  # Complete this line!
        error[i] = sum(rf.prediction != df.train[test, "Reverse"])/length(test)  # Complete this line!
        # cat('=======', j, ',', i, '=======', '\n')
    }
    error.all[j] = mean(error)
}
# In case there are multiple matches, we take the first one.
nodesize.opt = 10 + which(error.all == min(error.all))[1]

# The following code is for calculating the misclassification
# error on the real testing data using the best tuned
# nodesize.
rf.model <- randomForest(Reverse ~ Circuit + Issue + Petitioner +
    Respondent + LowerCourt + Unconst, data = df.train, ntree = 10,
    nodesize = nodesize.opt)  # Complete this line
rf.prediction = predict(rf.model, newdata = df.test[, 1:ncol(df.test) -
    1], type = "class")  # Complete this line
error = sum(rf.prediction != df.test[, "Reverse"])/length(rf.prediction)  # Complete this line
paste("Error:", error)
```

```
## [1] "Error: 0.317647058823529"
```