Andrew Rothman
anr248@stanford.edu

STATS 202 – Homework #4

November 2nd, 2018
SUID: 06325118

Problem #1:

Modified from exercise 3, section 5.4 of ITSL textbook

We now review *k-fold cross-validation*.

(a) Explain how *k-fold cross validation* is implemented. Write example R-code to illustrate this.

Answer:
When fitting a statistical model (such as a logistic regression model) to use specifically for prediction purposes, we could assess the "fit" or "predictive accuracy" of the model on the same data used to recover estimates of the model parameters. We call this the "training error", and it is not particularly informative as to how the prediction model we will work on "new data" or "new observations" (due to issues such as overfitting).

We could instead use a "validation set" approach where the dataset is split into two parts (a "training set" and a "testing set"). We then fit the model on the "training set" and assess the "testing error" of the model on the "testing set". There is a fair amount of variability in this method however, given the inherent variability of the data chosen into the training and testing sets.

K-fold cross validation is a multi-step process involving repeated non-overlapping "validation set" approaches on the dataset. We choose K to be a positive integer (for instance the number "10"). We then randomly assign each observation in the dataset an integer from 1 to K with equal probability. We call this newly created variable "CV_id". This splits the dataset into 10 roughly equally sized datasets. We then perform the cross-validation:

1. Specify variable "i" to be the number "1"
2. Specify all observations in the dataset with CV_id equal to "i" as the testing set, and all observations with CV_id not equal to "i" as the training set. Fit the statistical model using the training set, and assess the predictive performance of the model on the testing set. Store the performance of the model.
3. Increment "i" by "1" and go back to step one. Continue these iterative steps until i==K

At the end of these iterative steps, each of the "K" datasets has had its turn as the testing set, with the remaining "K-1" datasets constituting the training set. We will also have "K" measures of the model "testing error". We can average these K numbers together to get the average testing error or "Cross validation error". We can also recover an estimate of the standard deviation of the average testing error.

Attached below is R-code running 5-fold cross validation for fitting a multivariate logistic regression model in the "Boston" dataset using variables "indus", "age", and "tax" to predict being above the median crime rate. Using a 0.5 cutoff to categorize predictions into "0/1" classes of "below/above" the median crime rate, the CV predictive performance from the 5-fold cross validation was estimated to be 0.817 with standard deviation 0.044

R Code:

```
################
## Problem #1 ##
################
library(MASS)
library(dplyr)
library(psych)
set.seed(12345)
df <- Boston

## explore the Boston dataset
dim(df)
head(df)
colnames(df)
?Boston
summary(df)

## specify the median of the crime variable
median_cr <- median(df$crim)

## add in CV integer variable for later Cross Validation
k <- 5
df$CV_id <- sample(1:k, dim(df)[1], replace=TRUE)
table(df$CV_id, exclude=NULL)
CV_vector <- rep(NA, k)

## specify the binary outcome
df$outcome <- 0
df$outcome[df$crim>median_cr] <- 1
table(df$outcome, exclude=NULL)

for(i in 1:k){
  df_train <- dplyr::filter(df, CV_id!=i)
  df_test <- dplyr::filter(df, CV_id==i)

  df_test$p_outcome <- predict(glm(outcome ~ indus + age + tax, family=binomial,
data=df_train), df_test, type="response")
  df_test$p_outcome_class <- 0
  df_test$p_outcome_class[df_test$p_outcome>0.5] <- 1

  CV_vector[i] <- (dim(dplyr::filter(df_test, outcome==p_outcome_class))[1]) /
(dim(df_test)[1])

  rm(df_train, df_test)
}

CV_performance <- mean(CV_vector)

CV_vector
CV_performance
sd(CV_performance)
```

(b) What are the advantages and disadvantages of *k-fold cross validation* relative to:
   i. The validation set approach?
   ii. LOOCV?

Answer:

Validation Set approach:

- Advantages:
    - In assessing the predictive performance of a model or method for “new data” or “new observations”, there is a clear advantage of the validation set approach over assessing the fit on the very same data used to recover estimates of the model parameters. As compared to the “training error”, the validation set approach will be less likely to suffer from overfitting and provide a more accurate representation of the “testing error”.
    - The validation set approach is fairly easy to do computationally. We are still just fitting one model, and then assessing the fit of that model on another “testing” set. This approach is not computationally expensive.
- Disadvantages:
    - If we were to attempt the validation set approach multiple times on the same dataset, the estimate of the “test error” may be very variable given the randomness inherent in choosing which observations fall into the “training” and “testing” sets.
    - Only the observations included in the “training set” are used to fit the model. This lowers the sample size, leading to a worse fitting model. The validation set approach in this case may actually overestimate the testing error.

Leave One Out Cross Validation (LOOCV) approach:

- Advantages:
    - Unlike the validation set approach, every observation in the dataset (as some point in the analysis) is used in the training set, and at some point in the testing set
    - There is no “random” component to LOOCV, given each “testing set” is just one observation and every observation will get it’s turn as the “testing set” at some point in the analysis
    - Because n-1 observations are being used to fit the model in “training”, these are more observations than you would have in the validation set approach. Therefore you don’t have this same loss in sample size of the training set you have in the validation set approach
- Disadvantages:
    - LOOCV is computationally expensive. We fit as many models as there are observations in the original dataset. These can lead to analyses that take an unreasonable or untenable amount of time to complete.

  - Because each training set is all but one of the observations, the training sets in each round of the LOOCV and resulting models that are fit on those training sets will be highly correlated. This will lead to high variance in the estimate of the average testing error among the CV rounds.

Problem #2:

Modified from exercise 5, section 5.4 of ITSL textbook

In HW#3, we used logistic regression to predict the probability of a suburb having a crime rate above the median crime rate in the "Boston" dataset using several covariates. We will now estimate the test error of this logistic regression model using the validation set approach.

(a) Fit a logistic regression model that uses all of the covariates in the dataset except "indus" to predict if a suburb has a crime rate above the median crime rate.

Answer:
There were a total of 506 observations included in fitting the regression model. Estimates of the recovered logistic regression parameters are shown below:

Multivariate Logistic Regression on outcome "above median crime rate"

	Regression Coefficient	p-value
(Intercept)	-32.746238	3.51e-07
zn	-0.081251	0.01279
chas	0.609636	0.40055
nox	44.974782	2.31e-11
rm	-0.432711	0.53817
age	0.022610	0.06267
dis	0.699559	0.00142
rad	0.718103	1.15e-06
tax	-0.007418	0.00290
ptratio	0.358451	0.00348
black	-0.012818	0.03807
lstat	0.032852	0.49355
medv	0.168093	0.01350

(b) Using the validation set approach, estimate the test error of this model.

Answer:
The 506 total observations were randomly split in a training set (388 observations) and a testing set (118 observations). The logistic regression model was fit on the training set, with the prediction error assessed on the test set. Using a probability cut-off of 0.5 to categorize the predicted outcome into "above" or "below" the median crime rate, the resulting confusion matrix on the testing set is show below:

		Actual	
		Below	Above
Predicted	Below	53	9
	Above	3	53

Therefore the test error on this particular randomly drawn testing set is:

$$(9+3)/_{118} \approx 0.1017$$

(c) Repeat the process in (b) three times, using three different splits of the observations into a training set and validation set. Comment on the results obtained.

Answer:

The dataset is split into a training and testing set three more times. The resulting three confusion matrices assessing the error on the testing set are shown below:

		Actual	
		Below	Above
Predicted	Below	54	6
	Above	11	56

$$test\ error = (6+11)/_{127} \approx 0.1339$$

		Actual	
		Below	Above
Predicted	Below	67	10
	Above	10	48

$$test\ error = (10+10)/_{135} \approx 0.1481$$

		Actual	
		Below	Above
Predicted	Below	54	6
	Above	1	65

$$test\ error = (1+6)/_{126} \approx 0.0556$$

Considering the results from part (b) as well, the average test error is approximately 0.1098, with standard deviation 0.0411.

(d) Now fit the model from (b), but include "indus" as a covariate in the model. Comment on whether or not including "indus" leads to a reduction in the test error rate.

As in parts (b) and (c), the dataset was split into four different "testing" and "test" sets (i.e., I ran 4-fold cross validation). Using the same random-seed, I used the same split of testing and test sets as in parts (b) and (c). The resulting confusion matrices and test error rates are shown below:

Answer:

		Actual	
		Below	Above
Predicted	Below	53	5
	Above	3	57

$$test\ error = {(5+3)}/{118} \approx 0.0678$$

		Actual	
		Below	Above
Predicted	Below	55	5
	Above	10	57

$$test\ error = {(10+5)}/{127} \approx 0.1181$$

		Actual	
		Below	Above
Predicted	Below	68	10
	Above	9	48

$$test\ error = {(9+10)}/{135} \approx 0.1407$$

		Actual: Below	Actual: Above
Predicted	Below	54	6
	Above	1	65

$$test\ error = {(1+6)}/{126} \approx 0.0556$$

The average test error is approximately 0.0955, with standard deviation 0.0405. As compared to the results from part “c”, we can see the average test error (as well as the estimated standard deviation) decreased when including “indus” as a covariate in the model. So I would conclude the inclusion of “indus” as a covariate in the model leads to a reduction in the test error rate.

R Code:

```
################
## Problem #2 ##
################
library(MASS)
library(dplyr)
library(psych)
set.seed(10815657)
df <- Boston

## explore the Boston dataset
dim(df)
head(df)
colnames(df)
?Boston
summary(df)

## specify the median of the crime variable
median_cr <- median(df$crim)

## specify the binary outcome
df$outcome <- 0
df$outcome[df$crim>median_cr] <- 1
table(df$outcome, exclude=NULL)

############
## Part A ##
############
summary(glm(outcome ~ zn + chas + nox + rm + age + dis + rad + tax + ptratio + black + lstat +
medv, data=df, family="binomial"))

############
## Part B ##
############
k <- 4
df$CV_id <- sample(1:k, dim(df)[1], replace=TRUE)
table(df$CV_id, exclude=NULL)
CV_vector <- rep(NA, k)

df_train <- dplyr::filter(df, CV_id!=1)
df_test <- dplyr::filter(df, CV_id==1)

df_test$p_outcome <- predict(glm(outcome ~ zn + chas + nox + rm + age + dis + rad + tax + ptratio
+ black + lstat + medv, data=df_train, family="binomial"), df_test, type="response")
df_test$p_outcome_class <- 0
df_test$p_outcome_class[df_test$p_outcome>0.5] <- 1
```

```
table(df_test$p_outcome_class, df_test$outcome)
CV_vector[1] <- (dim(dplyr::filter(df_test, outcome!=p_outcome_class))[1]) / (dim(df_test)[1])

rm(df_train, df_test)

############
## Part C ##
############

for(i in 2:k){
  df_train <- dplyr::filter(df, CV_id!=i)
  df_test <- dplyr::filter(df, CV_id==i)

  df_test$p_outcome <- predict(glm(outcome ~ zn + chas + nox + rm + age + dis + rad + tax +
ptratio + black + lstat + medv, data=df_train, family="binomial"), df_test, type="response")
  df_test$p_outcome_class <- 0
  df_test$p_outcome_class[df_test$p_outcome>0.5] <- 1

  table(df_test$p_outcome_class, df_test$outcome)
  CV_vector[i] <- (dim(dplyr::filter(df_test, outcome!=p_outcome_class))[1]) / (dim(df_test)[1])

  rm(df_train, df_test)
}

############
## Part D ##
############

CV_vector_partD <- rep(NA, k)
for(i in 1:k){
  df_train <- dplyr::filter(df, CV_id!=i)
  df_test <- dplyr::filter(df, CV_id==i)

  df_test$p_outcome <- predict(glm(outcome ~ zn + indus+ chas + nox + rm + age + dis + rad + tax
+ ptratio + black + lstat + medv, data=df_train, family="binomial"), df_test, type="response")
  df_test$p_outcome_class <- 0
  df_test$p_outcome_class[df_test$p_outcome>0.5] <- 1

  table(df_test$p_outcome_class, df_test$outcome)
  CV_vector_partD[i] <- (dim(dplyr::filter(df_test, outcome!=p_outcome_class))[1]) /
(dim(df_test)[1])

  rm(df_train, df_test)
}
```

Problem #3:

Modified from exercise 6, section 5.4 of ITSL textbook

We continue to consider logistic regression to predict the probability of a suburb having a crime rate above the median crime rate in the "Boston" dataset, but now with only predictors "chas" and "rm".

(a) Using the "summary()" and "glm()" functions, determine the estimated standard errors for the coefficients associated with "chas" and "rm" in a multiple logistic regression model that uses both predictors.

Answer:

Multivariate Logistic Regression on outcome "above median crime rate"

	Regression Coefficient	Standard Error	p-value
(Intercept)	3.0346	0.8546	0.000384
chas	0.7087	0.3720	0.056767
rm	-0.4906	0.1358	0.000301

As shown in the table above, from the glm function:

- The estimated standard error for the "chas" regression coefficient is 0.3720
- The estimated standard error for the "rm" regression coefficient is 0.1358

(b) Write a function "boot.fn()" that takes as input the "Boston" dataset as well as an index of the observations, and that outputs the coefficient estimates for "chas" and "rm" in the multiple logistic regression model.

Answer:

Shown below is the R-code for the "boot.fn()" function:

```
############
## Part B ##
############
boot.fn <- function(data, index){
  return(coef(glm(outcome ~ chas + rm, data=data, family="binomial", subset=index)))
}

boot.fn(df, 1:(dim(df)[1]))
```

The regression coefficient estimates output from the "boot.fn()" function are:

- Estimated regression coefficient for "chas" = 0.7087487
- Estimated regression coefficient for "rm" = -0.4906151

(c) Use the “boot()” function together with your “boot.fn()” function to estimate the standard errors of the logistic regression coefficients for “chas” and “rm”.

Answer:

The standard errors from the bootstrap are:

- Estimated bootstrap se for the regression coefficient for “chas” = 0.4040677
- Estimated bootstrap se for the regression coefficient for “rm” = 0.1482745

(d) Comment on the estimated standard errors obtained using the “glm()” function and using your bootstrap function.

Answer:

Comparing the MLE standard error estimates in part (a) with the bootstrap standard error estimates in part (c), we can see the bootstrap se estimates are larger for the estimated regression coefficients of both “chas” and “rm” as compared to their MLE counterparts recovered from the “glm” function. This is not surprising, as we would expect the standard error estimates of the bootstrap to be more conservative than the MLE standard error estimates.

<u>R Code:</u>

```
################
## Problem #3 ##
################
library(MASS)
library(dplyr)
library(psych)
set.seed(10815657)
df <- Boston

## explore the Boston dataset
dim(df)
head(df)
colnames(df)
?Boston
summary(df)

## specify the median of the crime variable
median_cr <- median(df$crim)

## specify the binary outcome
df$outcome <- 0
df$outcome[df$crim>median_cr] <- 1
table(df$outcome, exclude=NULL)

############
## Part A ##
############
summary(glm(outcome ~ chas + rm, family="binomial", data=df))

############
## Part B ##
############
boot.fn <- function(data, index){
  return(coef(glm(outcome ~ chas + rm, data=data, family="binomial", subset=index)))
}

boot.fn(df, 1:(dim(df)[1]))


############
## Part C ##
############
library(boot)
boot(df, boot.fn, 1000)
```

Problem #4:

Modified from exercise 9, section 5.4 of ITSL textbook

Consider the "Boston" housing dataset.

(a) Based on this data set, provide an estimate for the population mean of "crim". Call this estimate $\hat{\mu}$.

Answer:
The estimate for the population mean of "crim" is $\hat{\mu} = 3.613524$

(b) Provider an estimate of the standard error of $\hat{\mu}$. Interpret this result.

Answer:
The estimate for the standard error for the population mean of "crim" is $SE(\hat{\mu}) = 0.3823853$

We would interpret this as, assuming the observations in the "crim" variable are iid, the SE estimate (i.e. the standard deviation of the sampling distribution) is 0.3823853. We are able to make this assessment from the Central Limit Theorem (CLT).

(c) Now estimate the standard error of $\hat{\mu}$ using the bootstrap. How does this compare to your answer from (b)?

Answer:
The bootstrap estimate for the standard error for the population mean of "crim" is $SE_{boot}(\hat{\mu}) = 0.3892534$

We can see that the bootstrapped standard error estimate is larger than the SE estimated in part (b). This is expected, as we would expect the bootstrap SE estimate to be more conservative than the estimate in part (b).

(d) Based on your bootstrap estimate from (c), provide a 95% CI for the mean of "crim". Compare it to the results obtained using "t.test(Boston$crim).

Answer:
The bootstrap 95% CI is:

- $\hat{\mu} \pm 2SE_{boot}(\hat{\mu}) = 3.613524 \pm (2 * 0.3892534) = (2.835, 4.392)$

The 95% CI from the "t.test" function is:

- $(2.862, 4.365)$

Comparing the 95% CIs above, we can see the CIs produced using the bootstrapped SE estimates are wider than the CIs recovered using the "t.test" function. This is again expected as we would expect the bootstrapped SE estimates to be conservative, thus leading to wider 95% CIs.

(e) Based on this data set, provide an estimate, $\hat{\mu}_{med}$, for the median value of "crim" in the population.

Answer:
The estimate for the population median of "crim" is $\hat{\mu}_{med} = 0.25651$

(f) We now would like to estimate the standard error of $\hat{\mu}_{med}$. Unfortunately, there is no simple formula for the standard error of the median. Instead, estimate the standard error of the median using the bootstrap. Comment on your findings.

Answer:
The bootstrap estimate for the standard error for the population median of "crim" is $SE_{boot}(\hat{\mu}_{med}) = 0.03702524$

Using the bootstrap to create a consistent empirical reconstruction of the sampling distribution for μ_{med}, we estimate the standard deviation of this sampling distribution (i.e. the SE of μ_{med}) to be 0.03702524

(g) Based on this data set, provide an estimate for the tenth percentile of "crim" in Boston suburbs. Call this quantity $\hat{\mu}_{0.1}$.

Answer:
The estimate for the population tenth percentile of "crim" is $\hat{\mu}_{0.1} = 0.038195$

(h) Use the bootstrap to estimate the standard error of $\hat{\mu}_{0.1}$. Comment on your findings.

Answer:
The bootstrap estimate for the standard error for the population tenth percentile of "crim" is $SE_{boot}(\hat{\mu}_{0.1}) = 0.003128331$

Using the bootstrap to create a consistent empirical reconstruction of the sampling distribution for $\mu_{0.1}$, we estimate the standard deviation of this sampling distribution (i.e. the SE of $\mu_{0.1}$) to be 0.003128331

R Code:

```
################
## Problem #4 ##
################
library(MASS)
library(dplyr)
library(psych)
set.seed(10815657)
df <- Boston

## explore the Boston dataset
dim(df)
head(df)
colnames(df)
?Boston
summary(df)

############
## Part A ##
############
mean(df$crim)

############
## Part B ##
############
se_estimate <- sd(df$crim) / sqrt(dim(df)[1])
se_estimate

############
## Part C ##
############
boot.part_c <- function(data, index){
  x <- data$crim[index]
  return(mean(x))
}

library(boot)
part_c_results <- boot(df, boot.part_c, 1000)
part_c_results

############
## Part D ##
############
t.test(df$crim)


############
## Part E ##
############
median(df$crim)

############
## Part F ##
############
boot.part_f <- function(data, index){
  x <- data$crim[index]
  return(median(x))
}

part_f_results <- boot(df, boot.part_f, 1000)
part_f_results
```

```
############
## Part G ##
############
quantile(df$crim, probs = 0.1)

############
## Part H ##
############
boot.part_h <- function(data, index){
  x <- data$crim[index]
  return(quantile(x, probs = 0.1))
}

part_h_results <- boot(df, boot.part_h, 1000)
part_h_results
```

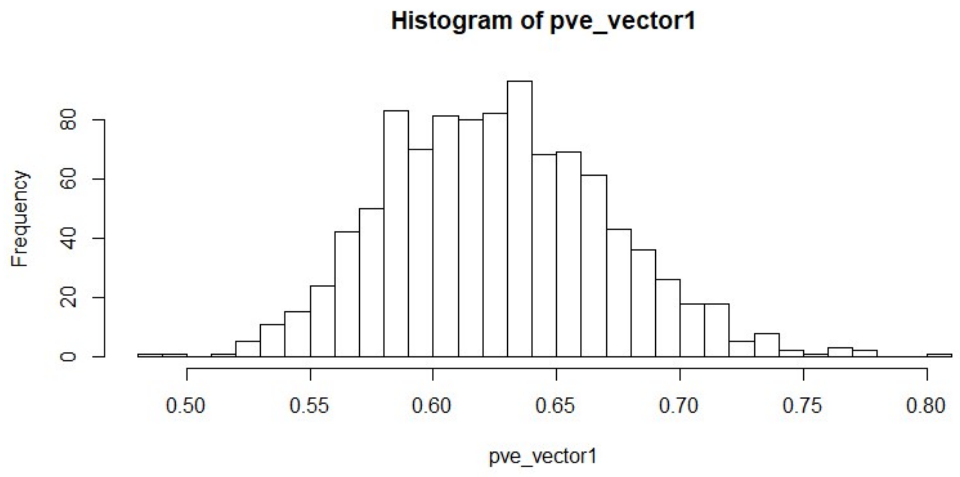
Problem #5:

The PCA algorithm uses the sample variance-covariance matrix. This is a statistic derived from a finite sample; therefore, it can deviate from the true covariance matrix of the variables. Because of this, every quantity estimated in PCA has some error.

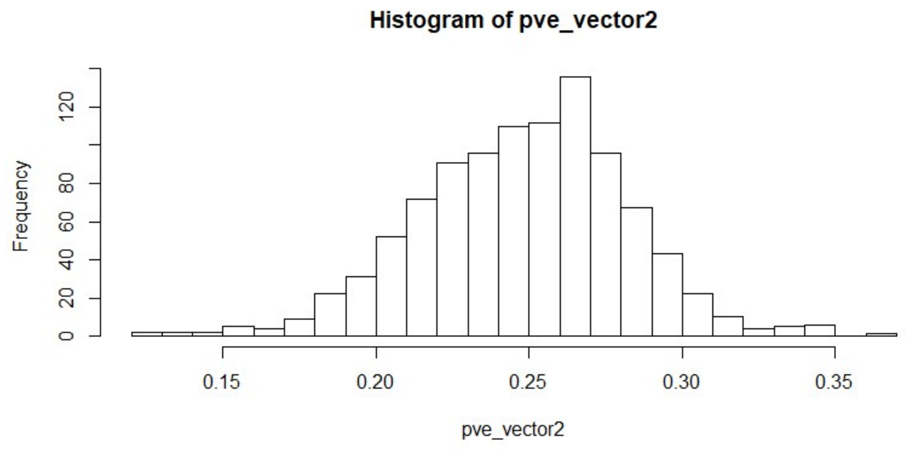
1. Using the "USArrests" dataset, plot a histogram of the proportion of variance explained by the first 2 principal components in 1000 Bootstrap resamplings of the data.

   Answer:

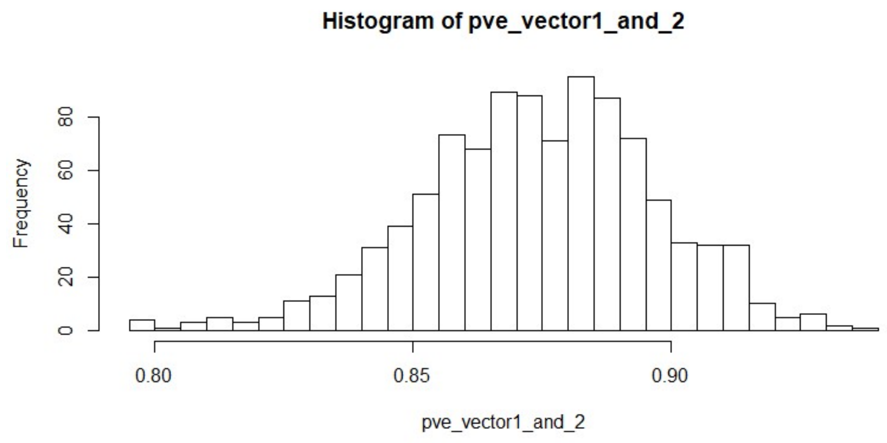
   Below is a histogram of the proportion of variance explained by the 1st principal component in each of the 1000 bootstrapped samples:



   Below is a histogram of the proportion of variance explained by the 2nd principal component in each of the 1000 bootstrapped samples:



Below is a histogram of the sum of the proportion of variance explained by the 1st and 2nd principal components in each of the 1000 bootstrapped samples:



2. Estimate a standard error and 95% CIs for the proportion of variance explained by the first 2 principal components.

   Answer:
   For proportion of variance explained by the 1st principal component:
   - Estimated standard error = 0.04546522
   - Estimated 95% CI ≈ $0.62006039 \pm 2(0.04546522) = (0.52913\,, 0.7109908)$

   For proportion of variance explained by the 2nd principal component:
   - Estimated standard error = 0.03391394
   - Estimated 95% CI ≈ $0.2474413 \pm 2(0.03391394) = (0.1796134\,, 0.3152692)$

   For sum of proportion of variance explained by the 1st and 2nd principal components:
   - Estimated standard error = 0.02262878
   - Estimated 95% CI ≈ $0.8675017 \pm 2(0.02262878) = (0.8222441\,, 0.9127593)$

3. Suppose we compute the first principal component from each of 1000 Bootstrap resamplings of the data. Using the resulting 1000 vectors, we estimate the standard error of each entry or loading using Eq. 5.8 in the textbook. Explain why this would be problematic.

   Answer:
   In each bootstrapped sample, we perform a PCA and recover the principal components (PCs). For a specified PC, there are two directions the PC can "point-in". The direction the PCs "point-in" in the orthogonal hyperdimensional parameter space is somewhat arbitrary with regards to the "tail" and "head" of that principal component. Each PC score for an observation is an additive linear combination of the features used to recover the PCs. Whichever way the PC is labeled "pointing" (and hence signifying one end of the PC vector as the "head" and the other the "tail"), the associated loadings of the PCs will reflect this. Hence as long as we keep our coordinate system consistent, using the PCs for any type of analysis (whether it be

dimensionality reduction, etc), it will not matter which end of the PC is treated as the "tail" and which the "head".

However, if we are recovering the PCs in each bootstrapped dataset and using them to assess the standard error of each loading of each PC, there is a potential issue. We are assuming two things:

- If there are "p" features being included in the PCA, then the analysis will produce "p" PCs from each bootstrapped dataset. In terms of ordering these "p" PCs in each bootstrapped dataset by the proportion of variance they explain (from largest to least), we are assuming that order will always be the same from bootstrapped set to set. This may not be the case, particularly when we get to some of the last PCs that account for little of the variance. Due to the inherent random component in generating the bootstrapped datasets, particularly some of the last PCs may switch ordering.
- Even if the PCs are arranged in the same order from bootstrap set to set, we are assuming which end of the PC was chosen as the "tail" and which the "head" is the same in each PCA analysis. This may not be the case (again, because the decision is somewhat arbitrary)

4. There is a way around the problem alluded to in part 3. Write a function in R which, given a data.frame:
    - Compute the vector of loadings for the first principal component and define "i" to be the index of the element with highest absolute value.
    - For each of 1000 bootstrap resamplings of the data.frame, compute the vector of loadings for the first principal component and multiplies it by the sign of its ith element to generate *signed loadings*.

Answer:

R Code:

```
part_4 <- function(df){
  PC1_matrix <- matrix(NA, dim(df)[2], 1000)

  for(i in 1:1000){
    df_matrix <- as.matrix(df)
    bs_matrix <- df_matrix[(sample(dim(df)[1], dim(df)[1], replace=T)),]
    PR <- prcomp(bs_matrix, scale=T)
    PC1 <- PR$rotation[,1]
    largest_value <- max(PC1)
    if(largest_value < 0){
      PC1 <- PC1*-1
    }
    PC1_matrix[,i] <- PC1

    rm(df_matrix, bs_matrix, PR, PC1, largest_value)
  }
  return(PC1_matrix)
}

PC1_matrix <- part_4(df)
```

5. Apply the function to "USArrests".

   Answer:
   I applied my function to the USArrests dataset. The resulting loadings of the 1st PC, along with the associated bootstrapped estimated standard errors, are show below:

	PC1 loadings	Bootstrapped SE
Murder	0.5358995	0.1425142
Assault	0.5831836	0.1475317
UrbanPop	0.2781909	0.1086261
Rape	0.5434321	0.1262481

6. The function described in part 4 yield estimates of the standard error for each loading of the first principal component. On what assumption does this method rely? Would this give good standard error estimates for principal components beyond the first few? Explain.

   I already addressed the answer to this question in my answer to "part 4" of this question. I am copying my response below:

   Answer:
   If we are recovering the PCs in each bootstrapped dataset and using them to assess the standard error of each loading of each PC, there is a potential issue. We assume:

   - If there are "p" features being included in the PCA, then the analysis will produce "p" PCs from each bootstrapped dataset. In terms of ordering these "p" PCs in each bootstrapped dataset by the proportion of variance they explain (from most to least), we are assuming that order will always be the same from bootstrapped set to set. This may not be the case, particularly when we get to some of the last PCs that account for little of the variance. Due to the inherent random component in generating the bootstrapped datasets, particularly some of the last PCs may switch ordering.

   This issue is not likely to be problematic for the first or second PCs. Given the high amount of variance they explain, they are likely to be the same PCs being captured in each bootstrapped dataset and assigned as "PC1" and "PC2" accordingly when ranking the PCs from highest to lowest (in terms of the proportion of variance the explain). This may be an issue for the last two PCs. From bootstrap set to set, the ordering of PC3 and PC4 may switch given the low amount of variance they explain in the original dataset (between 4-6%) and the random component with bootstrapping. Therefore using the methodology we've established here (use bootstrapped PCs ordered from largest to smallest in terms of variance explained to assess the standard error of the PCs) for PC3 and PC4 may be ill advised.

<u>R Code:</u>

```
################
## Problem #5 ##
################
library(MASS)
library(dplyr)
library(psych)
set.seed(10815657)
df <- USArrests
head(df)

############
## Part 1 ##
############
pve_vector1 <- rep(NA, 1000)
pve_vector2 <- rep(NA, 1000)
pve_vector1_and_2 <- rep(NA, 1000)

for(i in 1:1000){
  df_matrix <- as.matrix(df)
  bs_matrix <- df_matrix[(sample(dim(df)[1], dim(df)[1], replace=T)),]

  PR <- prcomp(bs_matrix, scale=T)
  pve <- ((PR$sdev)^2) / sum((PR$sdev)^2)

  pve_vector1[i] <-  pve[1]
  pve_vector2[i] <-  pve[2]
  pve_vector1_and_2[i] <- pve[1] + pve[2]

  rm(df_matrix, bs_matrix, PR, pve)
}

hist(pve_vector1, breaks=25)
hist(pve_vector2, breaks=25)
hist(pve_vector1_and_2, breaks=25)


############
## Part 2 ##
############
PR <- prcomp(df, scale=T)
pve <- ((PR$sdev)^2) / sum((PR$sdev)^2)

se_1 <- sd(pve_vector1)
se_2 <- sd(pve_vector2)
se_1_and_2 <- sd(pve_vector1_and_2)

## 95% CI for pve_vector1
pve[1]
pve[1] - (2*se_1)
pve[1] + (2*se_1)


## 95% CI for pve_vector2
pve[2]
pve[2] - (2*se_2)
pve[2] + (2*se_2)

## 95% CI for pve_vector1_and_2
pve[1] + pve[2]
(pve[1]+pve[2]) - (2*se_1_and_2)
(pve[1]+pve[2]) + (2*se_1_and_2)
```

```
############
## Part 4 ##
############
part_4 <- function(df){
  PC1_matrix <- matrix(NA, dim(df)[2], 1000)

  for(i in 1:1000){
    df_matrix <- as.matrix(df)
    bs_matrix <- df_matrix[(sample(dim(df)[1], dim(df)[1], replace=T)),]
    PR <- prcomp(bs_matrix, scale=T)
    PC1 <- PR$rotation[,1]
    largest_value <- max(PC1)
    if(largest_value < 0){
      PC1 <- PC1*-1
    }
    PC1_matrix[,i] <- PC1

    rm(df_matrix, bs_matrix, PR, PC1, largest_value)
  }
  return(PC1_matrix)
}

PC1_matrix <- part_4(df)


############
## Part 5 ##
############
sd(PC1_matrix[1,])
sd(PC1_matrix[2,])
sd(PC1_matrix[3,])
sd(PC1_matrix[4,])

PR <- prcomp(df, scale=T)
PR$rotation
```