Homework 4 Solutions

Problem 1

(a)

Pseudo code:

```
KFold = function(data, K = 5, model.name){
  n = dim(data)[1]
  fold.size = int(n / K)
  test.error = rep(0, K)
  for(k in 1:K){
    test.idx = seq((k-1) * fold.size + 1, k * fold.size)
    train.data = data[-test.idx, ]
    test.data = data[test.idx, ]
    fit = model.name(train.data)
    test.error[k] = predict(fit, test.data)
  }
  return(test.error)
}
```

(b)

- Advantage of k-fold over validation set:
  - Less variance. In k-fold we average test errors k times, but in validation we only take one.
  - Less bias. Same reason as above.
- Disadvantage of k-fold over validation set:
  - Computationally more expensive
- Advantage of k-fold over LOOCV:
  - Computationally less expensive (if there is no shortcut for LOOCV like least squares)
- Disadvantage of k-fold over LOOCV:
  - LOOCV has less bias because it uses more samples to train the model
  - K-fold may yield different results each time you run it, but LOOCV will always yield the same result.

Problem 2

```
library(ISLR)
library(boot)
set.seed(1)
```

(a)

```
glmFit = glm(default ~ income + balance, data=Default, family='binomial')
summary(glmFit)
```

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

(b)

```
val.method = function(trainInd){
  glm.fit.train = glm(default ~ income + balance, data=Default[trainInd,], family='binomial')
  glm.probs.val = predict(glm.fit.train, newdata=Default[-trainInd,], type="response")
  glm.pred.val = rep("No", n-length(trainInd))
  glm.pred.val[glm.probs.val>0.5] = "Yes"
  valError = mean(glm.pred.val!=as.character(Default[-trainInd,1]))
}
n = nrow(Default)
train.ind = sample(1:n, n*0.5)
val.method(train.ind)
```

(c)

```
sapply(1:3, function(i){val.method(sample(1:n, n*0.5))})
```

```
## [1] 0.0236 0.0280 0.0268
```

The validation set errors for different splits are close. Thus, we can use one of them as an estimator of test error.

(d)

```
val.method.student = function(trainInd){
  glm.fit.train = glm(default ~ income + balance + student, data=Default[trainInd,],family='binomial')
  glm.probs.val = predict(glm.fit.train, newdata=Default[-trainInd,], type="response")
  glm.pred.val = rep("No", n-length(trainInd))
  glm.pred.val[glm.probs.val>0.5]="Yes"
  valError = mean(glm.pred.val!=as.character(Default[-trainInd,1]))
}
sapply(1:3, function(i){val.method.student(sample(1:n, n*0.5))})
```

```
## [1] 0.0264 0.0246 0.0264
```

The validation set error did not decrease significantly. Thus, including student variable does not lead to a reduction in the test error.

Problem 3

(a)

```
glmFit = glm(default ~ income + balance, data=Default, family='binomial')
summary(glmFit)
```

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

(b)

```
boot.fn = function(dat, trainIndices){
  glmFit = glm(default ~ income + balance, data=dat[trainIndices,], family='binomial')
```

```
  glmFit$coefficients
}
```

(c)

```
boot.result = boot(Default, boot.fn, R=1000)
boot.result
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Default, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##          original        bias     std. error
## t1* -1.154047e+01 -7.590393e-03 4.253451e-01
## t2*  2.080898e-05  5.166665e-08 4.598642e-06
## t3*  5.647103e-03  2.013605e-06 2.272256e-04
```

(d)

The standard error estimates in glm and bootstrap are very close. We see that the bootstrap method does not require any information on the model but gives good estimates of standard errors.

Problem 4

(a)

```
library(MASS)
library(boot)
set.seed(1)
mean(Boston$crim)
```

```
## [1] 3.613524
```

(b)

```
sd(Boston$crim)/sqrt(length(Boston$crim))
```

```
## [1] 0.3823853
```

The standard error is about 10% of the mean, which means the estimate is fairly accurate.

(c)

```
resample.mean = function(x){
  mean(sample(x, replace=TRUE))
}
boot.mean = sapply(1:1000, function(b){resample.mean(Boston$crim)})
sd(boot.mean)
```

```
## [1] 0.3761714
```

The standard error estimates from (b) and (c) are very close.

(d)

```
quantile(boot.mean, c(.025, .975))
```

```
##     2.5%    97.5%
## 2.918871 4.367247
```

```
t.test(Boston$crim)
```

```
##
##  One Sample t-test
##
## data:  Boston$crim
## t = 9.45, df = 505, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  2.862262 4.364786
## sample estimates:
## mean of x
##  3.613524
```

The confidence interval from bootstrap is a little bit wider because t-test assummes normality of the data.

(e)

```
median(Boston$crim)
```

```
## [1] 0.25651
```

(f)

```
resample.median = function(x){
  median(sample(x, replace=TRUE))
}
boot.median = sapply(1:1000, function(b){resample.median(Boston$crim)})
sd(boot.median)
```

```
## [1] 0.03802493
```

We get a reasonable estimate of standard error without any formula for computing the standard error of the median. The median is more robust to outliers than the mean, and we observe that its standard error is indeed smaller than the standard error of the mean.

(g)

```
quantile(Boston$crim, c(.1))
```

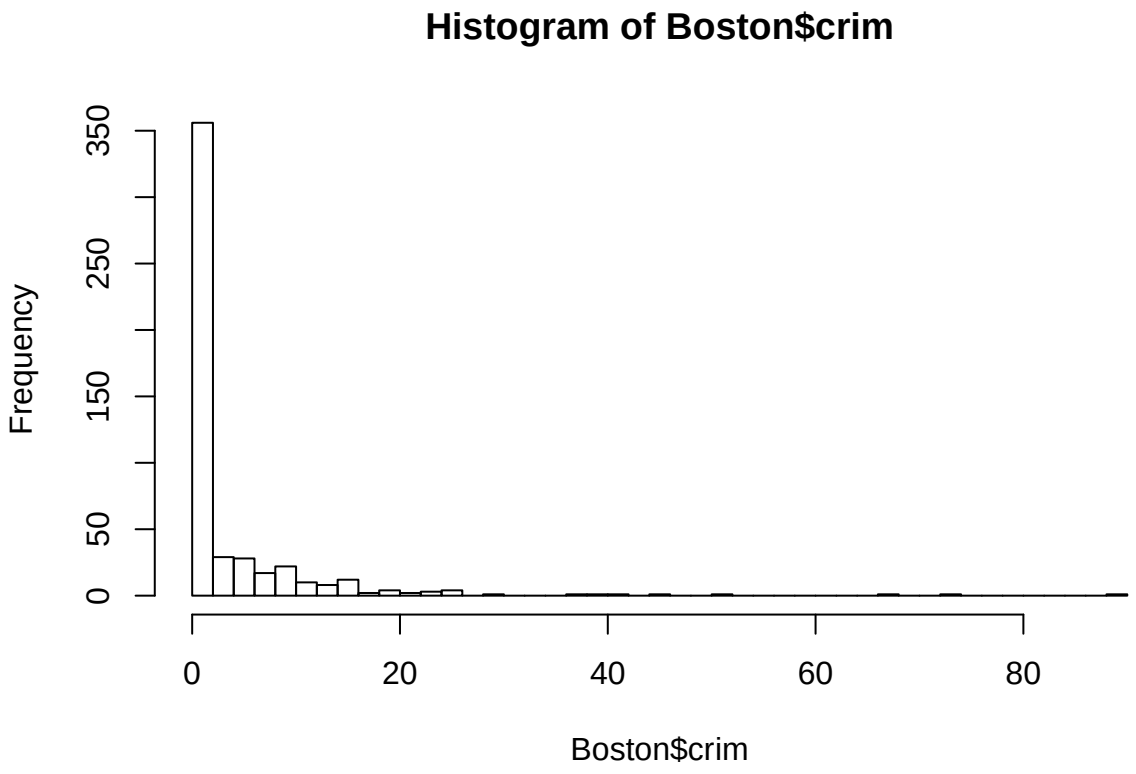
```
##      10%
## 0.038195
```

(h)

```
resample.quantile = function(x, p){
  quantile(sample(x, replace=TRUE),p)
}
boot.quantile = sapply(1:1000, function(b){resample.quantile(Boston$crim, 0.1)})
sd(boot.quantile)
```

```
## [1] 0.003194698
```

```
hist(Boston$crim, 50)
```

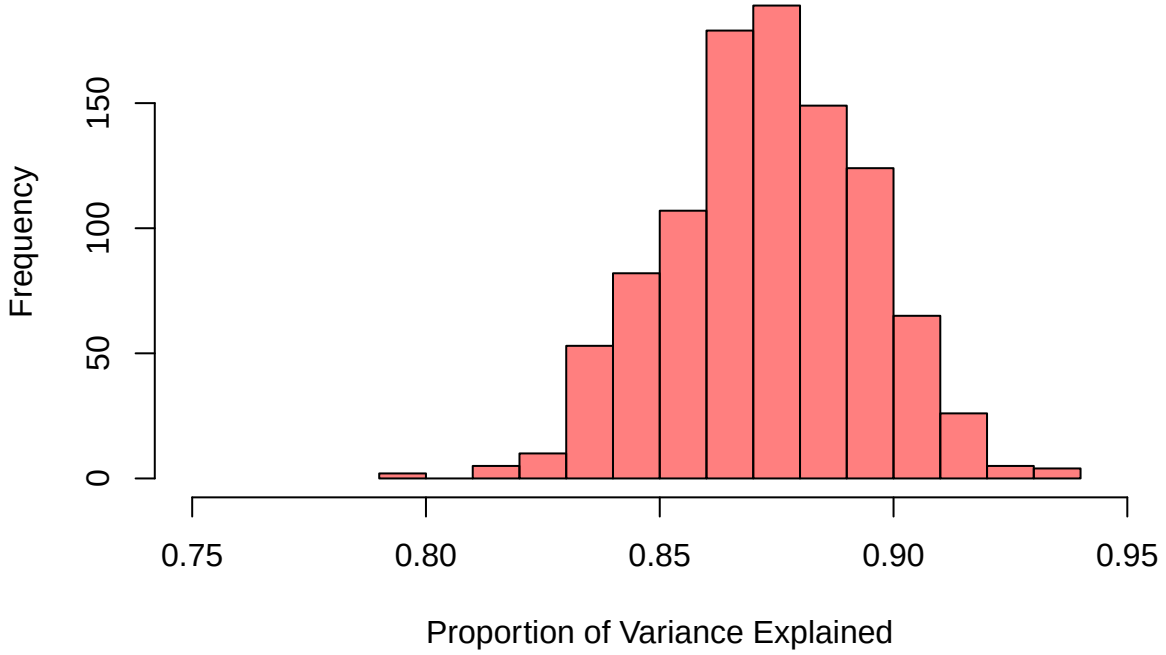


The estimate of standard error of .1 quantile is very small. (This can happen because the distribution of the data is concentrated on the small values. In this case, the estimate of the small quantile is accurate. However, if you try the 0.9 or 0.99 quantiles, the standard error should be very large.)

Problem 5

(a)

```
resample.pve = function(){
dat = USArrests[sample(1:nrow(USArrests), replace=TRUE),]
pr.out = prcomp(dat, scale=TRUE)
pr.var = pr.out$sdev^2
pr.pve = pr.var/sum(pr.var)
sum(pr.pve[c(1,2)])
}
boot.pve = t(sapply(1:1000, function(b){resample.pve()}))
hist(boot.pve, col=rgb(1,0,0,0.5),xlim=c(0.75,0.95), main="", xlab="Proportion of Variance Explained")
```



(b)

```
sd(boot.pve)
```

```
## [1] 0.02139831
```

```
quantile(boot.pve, c(.025, .975))
```

```
##      2.5%     97.5%
## 0.8320835 0.9133166
```

(c)

Suppose ϕ is the first principal component vector derived from the true covariance matrix. Then $-\phi$ is also a valid first principal component. If we compute this vector from bootstrap samples, each time we will get a slightly perturbed version of either ϕ or $-\phi$, as the numerical algorithm implemented in prcomp can lead to

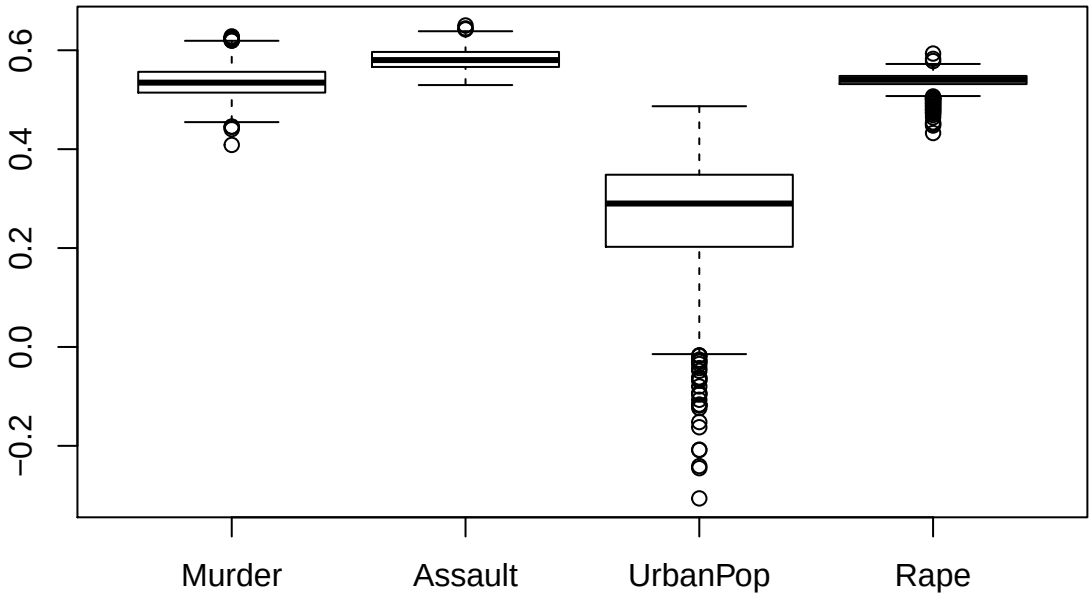
either result. Thus if we took the standard deviation of the bootstrapped loadings, we might overestimate the standard error of each loading.

(d)

```
resample.signed.loadings = function(i){
dat = USArrests[sample(1:nrow(USArrests), replace=TRUE),]
pr.out = prcomp(dat, scale=TRUE)
first.loading = pr.out$rotation[,1]
fl.sign = sign(first.loading[i])
first.signed.loading = first.loading*fl.sign
}
```

(e)

```
pr.out = prcomp(USArrests, scale=TRUE)
first.loading = pr.out$rotation[,1]
i = which.max(abs(first.loading))
boot.fsLoading = t(sapply(1:1000, function(b){resample.signed.loadings(i)}))
boxplot(boot.fsLoading)
```



(f)

The assumption is that the sign of the element with largest magnitude indicates the sign of the principal component vector in every bootstrap sample. In this example, Assault is chosen as the reference, and we see that the loadings of Assault fall in a narrow range. The directions are indeed specified for each bootstrap sample. This assumption may hold for the first few principal components, but it will break for the last principal components which tend to have larger standard errors.