HW4

Anish Mohan

October 18, 2015

1. Q1

- 1a.

```
  library(ISLR)

## Warning: package 'ISLR' was built under R version 3.2.2

  attach(Default)
  set.seed(2)
  glm.fit=glm(default~income+balance, family=binomial)
```

- 1b.

```
  train=sample(10000,6000)
  train_default=(Default[train,])
  test_default=(Default[-train,])
  glm.fit=glm(default~income+balance, train_default,family=binomial)
  glm.probs=predict(glm.fit,test_default,type="response")
  glm.pred=rep("No",4000)
  glm.pred[glm.probs>0.5]="Yes"
  table(glm.pred,test_default$default)

##
## glm.pred   No  Yes
##      No  3852   95
##      Yes   14   39

  print(paste0("Error rate = ", mean(glm.pred!=test_default$default)*100,
"%"))

## [1] "Error rate = 2.725%"
```

- 1c.

```
  train=sample(10000,5000)
  train_default=(Default[train,])
  test_default=(Default[-train,])
  glm.fit=glm(default~income+balance, train_default,family=binomial)
  glm.probs=predict(glm.fit,test_default,type="response")
  glm.pred=rep("No",5000)
  glm.pred[glm.probs>0.5]="Yes"
  table(glm.pred,test_default$default)
```

```
##
## glm.pred   No  Yes
##      No  4823  105
##      Yes   21   51

  print(paste0("Error rate = ", mean(glm.pred!=test_default$default)*100,
"%"))

## [1] "Error rate = 2.52%"

  train=sample(10000,500)
  train_default=(Default[train,])
  test_default=(Default[-train,])
  glm.fit=glm(default~income+balance, train_default,family=binomial)
  glm.probs=predict(glm.fit,test_default,type="response")
  glm.pred=rep("No",9500)
  glm.pred[glm.probs>0.5]="Yes"
  table(glm.pred,test_default$default)

##
## glm.pred   No  Yes
##      No  9141  223
##      Yes   45   91

  print(paste0("Error rate = ", mean(glm.pred!=test_default$default)*100,
"%"))

## [1] "Error rate = 2.82105263157895%"

  train=sample(10000,8000)
  train_default=(Default[train,])
  test_default=(Default[-train,])
  glm.fit=glm(default~income+balance, train_default,family=binomial)
  glm.probs=predict(glm.fit,test_default,type="response")
  glm.pred=rep("No",2000)
  glm.pred[glm.probs>0.5]="Yes"
  table(glm.pred,test_default$default)

##
## glm.pred   No  Yes
##      No  1937   31
##      Yes    8   24

  print(paste0("Error rate = ", mean(glm.pred!=test_default$default)*100,
"%"))

## [1] "Error rate = 1.95%"
```

- Fairly low error rates are observed even when using only 5% of given data for training the model. Logistic regression captures the underlying distribution well and is able to predict with a good accuracy.

- 1d.

```
train=sample(10000,5000)
train_default=(Default[train,])
test_default=(Default[-train,])
glm.fit=glm(default~income+balance+student, train_default,family=binomial
)
glm.probs=predict(glm.fit,test_default,type="response")
glm.pred=rep("No",5000)
glm.pred[glm.probs>0.5]="Yes"
table(glm.pred,test_default$default)

##
## glm.pred   No  Yes
##      No  4814  108
##      Yes   23   55

print(paste0("Error rate = ", mean(glm.pred!=test_default$default)*100,"%
"))

## [1] "Error rate = 2.62%"
```

  - Adding Student does not seem to a significant impact in reducing the test error.

2. Q2

- 2a.

```
set.seed(1)
glm.fit=glm(default~income+balance,Default,family=binomial)
summary(glm.fit)

##
## 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
```

- 2b.

```
boot.fn =function(data, index){
train_data=data[index,]
glm.fit=glm(default~income+balance,train_data,family=binomial)
return(glm.fit$coefficients)
}
```

- 2c.

```
library(boot)

## Warning: package 'boot' was built under R version 3.2.2

boot(data=Default, statistic = boot.fn, R=1000)

##
## ORDINARY NONPARAMETRIC BOOTSTRAP
```

```
##
##
## Call:
## boot(data = Default, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##           original        bias     std. error
## t1* -1.154047e+01 -8.008379e-03 4.239273e-01
## t2*  2.080898e-05  5.870933e-08 4.582525e-06
## t3*  5.647103e-03  2.299970e-06 2.267955e-04
```

```
#change R=1000 for final submission; copy value of R=100 for reference.

original        bias     std. error
```

t1* -1.154047e+01 -8.008379e-03 4.239273e-01 t2* 2.080898e-05 5.870933e-08 4.582525e-06 t3* 5.647103e-03 2.299970e-06 2.267955e-04

- 2d.
  - The standard error estimate for the coefficient ie. income and balance are same/very close (3 and 4 significant digits for R=1000) to the estimates we got from logistic regression.

3. Q3

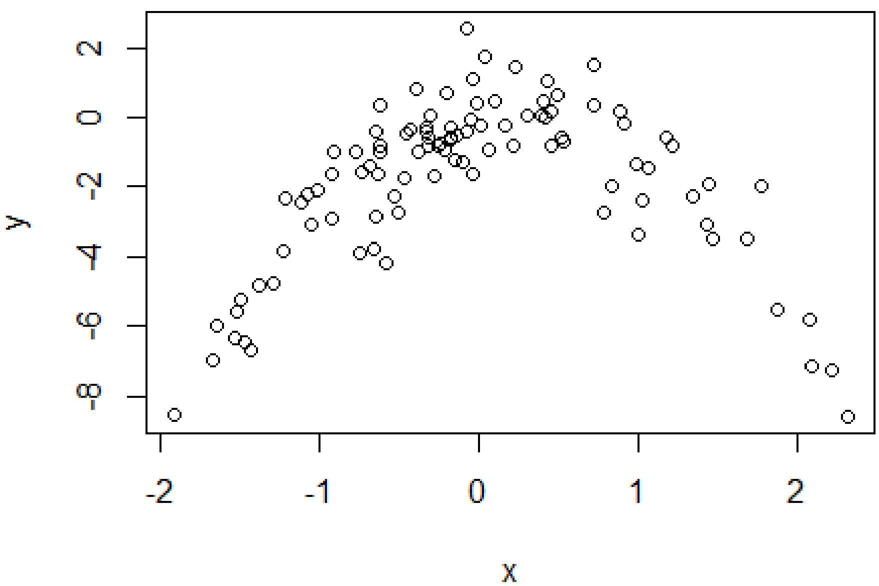
- 3a.

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

  - n=100; p=2
  - $y = x - 2 * x^2 + \epsilon$ where ϵ is random noise that is normally distributed with mean of 0 and std deviation of 1

- 3b.

```
plot(x,y)
```



  - There is non-linear(quadratic) relation between x and y. x has values between -2 and 2.4 and y hass between -8.7 and 2.6

- 3c.
  - i

```
xydat=data.frame(y,x)

set.seed(1)
glm.fit=glm(y~x, data=xydat)
```

```r
cv.err=cv.glm(xydat,glm.fit)
cv.err$delta[1]

## [1] 5.890979
```

- ii

```r
glm.fit=glm(y~poly(x,2), data=xydat)
cv.err=cv.glm(xydat,glm.fit)
cv.err$delta[1]

## [1] 1.086596
```

- iii

```r
glm.fit=glm(y~poly(x,3), data=xydat)
cv.err=cv.glm(xydat,glm.fit)
cv.err$delta[1]

## [1] 1.102585
```

- iv

```r
glm.fit=glm(y~poly(x,4), data=xydat)
cv.err=cv.glm(xydat,glm.fit)
cv.err$delta[1]

## [1] 1.114772
```

- 3d.

```r
set.seed(2)
glm.fit=glm(y~x, data=xydat)
cv.err=cv.glm(xydat,glm.fit)
cv.err$delta[1]

## [1] 5.890979

glm.fit=glm(y~poly(x,2), data=xydat)
cv.err=cv.glm(xydat,glm.fit)
cv.err$delta[1]

## [1] 1.086596

glm.fit=glm(y~poly(x,3), data=xydat)
cv.err=cv.glm(xydat,glm.fit)
cv.err$delta[1]

## [1] 1.102585

glm.fit=glm(y~poly(x,4), data=xydat)
cv.err=cv.glm(xydat,glm.fit)
cv.err$delta[1]
```

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

- Yes, the results are same as in part 3c. This is expected because we are running LOOCV on the full set of data and in both cases all 'n-1' subsets are evaluated.

- 3e. Quadratic model has the lowet error (1.0866). This is expected because true model is quadratic as well.

- 3f.

```
summary(glm.fit)

##
## Call:
## glm(formula = y ~ poly(x, 4), data = xydat)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8914  -0.5244   0.0749   0.5932   2.7796
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.8277     0.1041 -17.549   <2e-16 ***
## poly(x, 4)1   2.3164     1.0415   2.224   0.0285 *
## poly(x, 4)2 -21.0586     1.0415 -20.220   <2e-16 ***
## poly(x, 4)3  -0.3048     1.0415  -0.293   0.7704
## poly(x, 4)4  -0.4926     1.0415  -0.473   0.6373
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.084654)
##
##     Null deviance: 552.21  on 99  degrees of freedom
## Residual deviance: 103.04  on 95  degrees of freedom
## AIC: 298.78
##
## Number of Fisher Scoring iterations: 2
```

- In all trials in Part(c), the t-value for the coefficients of linear and quadratic terms are high (and p values are <0.05) indicating that they fit with least error. Yes these align with LOOCV as we found the lowest error in the quadratic model.

4. Q4

- 4a.

```
  library(MASS)
  attach(Boston)
  set.seed(1)
  mean(medv)

## [1] 22.53281
```

- 4b.

```
   sd(medv)/sqrt(length(medv))

## [1] 0.4088611
```

- 4c.

```
set.seed(1)
boot.fn2 = function(inputdata,index){
Boston_temp=inputdata[index,]
return (mean(Boston_temp$medv))
  }

  boot(data=Boston, statistic = boot.fn2, R=1000)

##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston, statistic = boot.fn2, R = 1000)
##
##
## Bootstrap Statistics :
##     original       bias    std. error
## t1* 22.53281 0.008517589   0.4119374
```

  - Standard Error of Mean from (b) was 0.409 and answer from bootstrap is 0.412. So the answer are very close (within 0.003) to each other

- 4d.

```
  t.test(Boston$medv)

##
##  One Sample t-test
##
## data:  Boston$medv
## t = 55.111, df = 505, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
```

```
## 95 percent confidence interval:
##  21.72953 23.33608
## sample estimates:
## mean of x
##  22.53281

  print(paste0("95% confidence from t.test is between 21.729 and 23.336")
)

## [1] "95% confidence from t.test is between 21.729 and 23.336"

  print(paste0("95% confidence interval from Bootstrap is between ", 22.5
32-2*0.412," and ", 22.532+2*0.412))

## [1] "95% confidence interval from Bootstrap is between 21.708 and 23.3
56"
```

- The values for the 95% confidence intervals are fairly close; same to 3 significant digits

- 4e.

```
  median(medv)

## [1] 21.2
```

- 4f.

```
set.seed(1)
bootmedian.fn = function(inputdata,index){
Boston_temp=inputdata[index,]
return (median(Boston_temp$medv))
  }

boot(data=Boston, statistic = bootmedian.fn, R=1000)

##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston, statistic = bootmedian.fn, R = 1000)
##
##
## Bootstrap Statistics :
##     original   bias    std. error
## t1*     21.2 -0.01615   0.3801002
```

- Using bootstrap to get standar error of median gives a value of 0.380 which is very small. With 95% confidence we can say that the true median is between 21.86 and 20.34

- 4g.

```
  quantile(medv,c(0.1))

##   10%
## 12.75
```

- 4h

```
set.seed(1)
boot_1Quantile.fn = function(inputdata,index){
Boston_temp=inputdata[index,]
return (quantile(Boston_temp$medv,c(0.1)))
  }

boot(data=Boston, statistic = boot_1Quantile.fn, R=1000)

##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston, statistic = boot_1Quantile.fn, R = 1000)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*    12.75 0.01005    0.505056
```

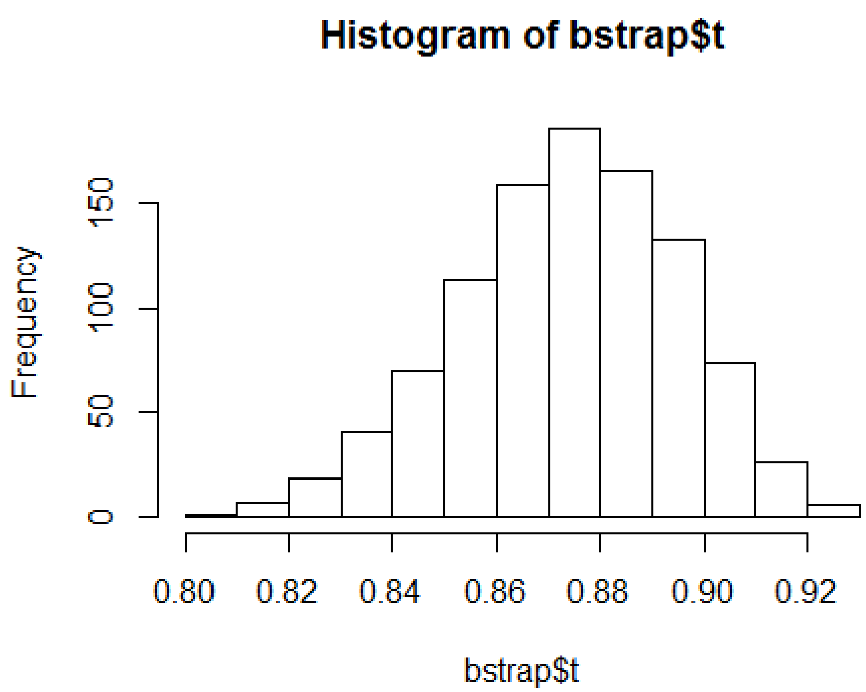
  - Using bootstrap to get standard error of 10th percentile gives a value of 0.505 which is very small. With 95% confidence we can say that the true 10th percentile is between 11.74 and 13.76

5. Q5

- 5.1

```
set.seed(1)
library(boot)
attach(USArrests)
boot_pca.fn = function(input, index){
temp_input=input[index,]
pca_set=prcomp(temp_input, center= TRUE, scale=TRUE)
pca_var=pca_set$sdev^2
prop_Var=(pca_var[1]+pca_var[2])/sum(pca_var)
return(prop_Var)
}

bstrap= boot(data=USArrests, statistic = boot_pca.fn, R=1000)
hist(bstrap$t)
```



- 5.2 Standard Error=0.0213

```
print(paste0("95% confidence interval for proportion of variance explaine
d by first 2 components is ", 0.8675-2*0.0213," and ", 0.8675+2*0.0213, "
or between 82.5% to 91.1%"))

## [1] "95% confidence interval for proportion of variance explained by f
irst 2 components is 0.8249 and 0.9101 or between 82.5% to 91.1%"
```

- 5.3

```
  boot_pca2.fn = function(input, index){
  temp_input=input[index,]
  pca_set=prcomp(temp_input,center=TRUE, scale=TRUE)
  pca_pc1=pca_set$rotation[,1]
  return(pca_pc1)
}

boot(data=USArrests, statistic = boot_pca2.fn, R=1000)

##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = USArrests, statistic = boot_pca2.fn, R = 1000)
##
##
## Bootstrap Statistics :
##       original    bias    std. error
## t1* -0.5358995 0.5751711   0.5367148
## t2* -0.5831836 0.6253797   0.5812652
## t3* -0.2781909 0.2922925   0.2851274
## t4* -0.5434321 0.5817628   0.5373784
```

  - This is problmatic because the loadings can be +ve or -ve i.e there can be 180 degree variation direction of the principal component. This change in direction does not impact the variance of the data, but when sampling 1000 times, the -ve and +ve loading muddle the findings for mean principal component.

- 5.4

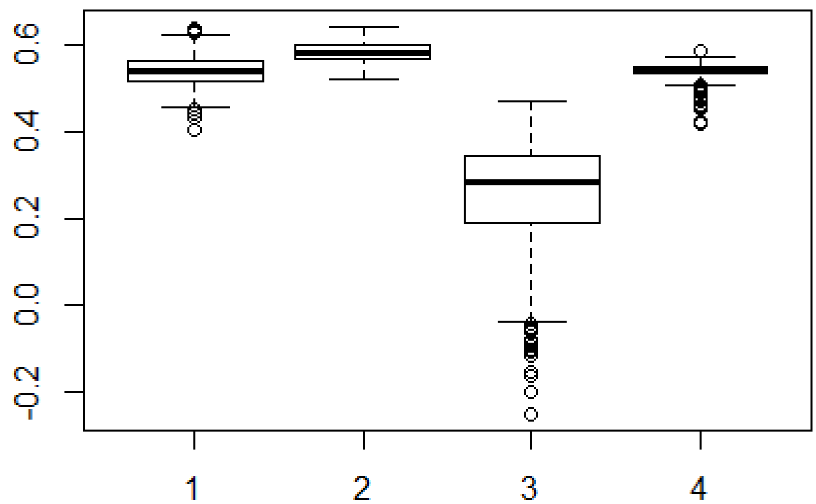
```
  runbstrap=function(inp_data){
    boot_pca3.fn = function(input, index){
      temp_input=input[index,]
      pca_set=prcomp(temp_input,center=TRUE, scale=TRUE)
       ind=which.max(abs(pca_set$rotation[,1]))
      v=sign(pca_set$rotation[ind,1])
      pca_signpc1=v*pca_set$rotation[,1]
      return(pca_signpc1)
    }

    bstrap=boot(data=inp_data, statistic = boot_pca3.fn, R=1000)

    boxplot(bstrap$t)
  }
```

- 5.5

```
  runbstrap(USArrests)
```



- 5.6
  - The function given in 5.4 finds the sign of the largest coefficient (i,e the variable that has most impact on the principal component) and changes direction of the principal component by 180 deg if the largest loading is -ve. This relies on the assumption that changing the direction of the principal component by 180 deg does impact the variance of data projected to that principal component.
  - Large absolute loadings indicate that the variable has a strong impact on that principal component. As we go towards lower/less important principal components, the absolute value of loadings is expected to decrease and hence the swings of the component across zero due to changing direction of principal component would have minimal impact in bootstrap. Hence it would not help much to improve the estimates for principal components beyond the first few.