Cross-validation and bootstrapping

STAT5003

Justin Wishart

Semester 2, 2022

Libraries to load

```
library(mlbench)
library(caret)
library(MASS)
library(class)
library(pROC)
```

Create simulation dataset

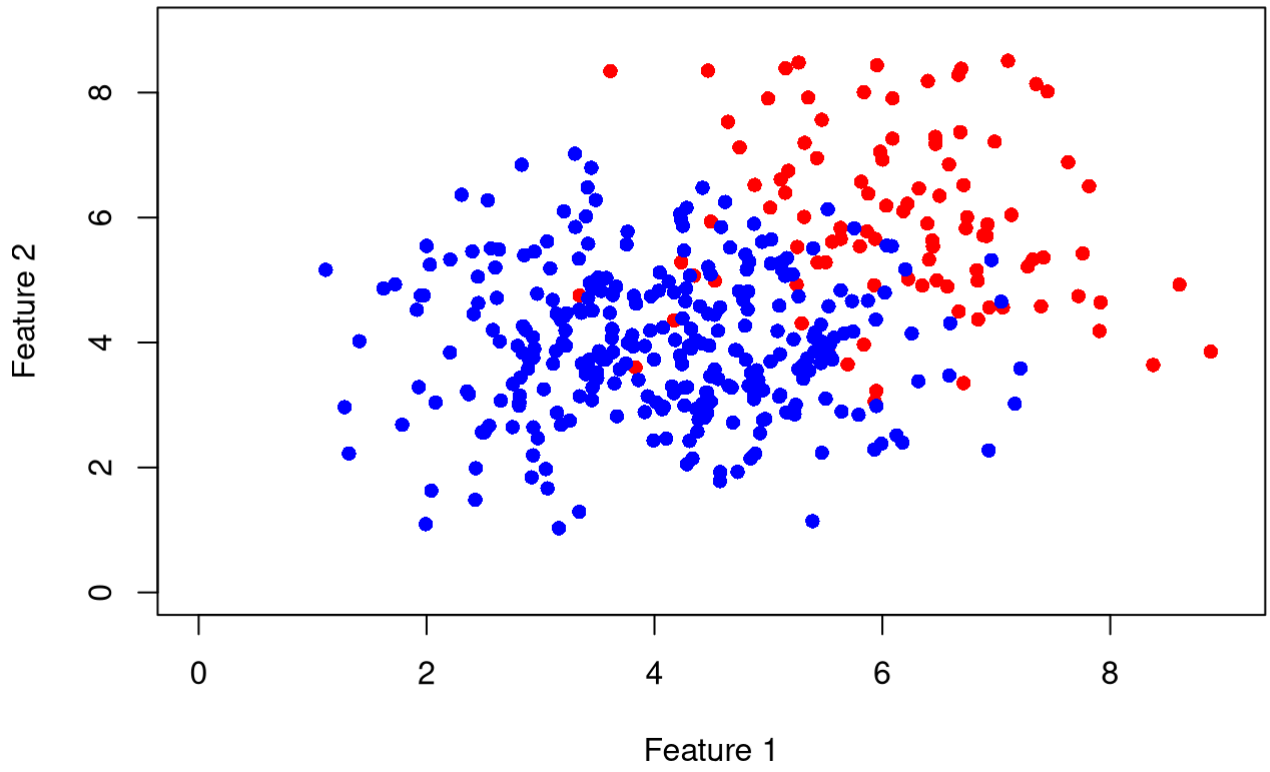
For the demonstration this week, we will first create a simulation dataset. Using `rnorm()`, we draw random features from a Gaussian distribution. We will simulate a dataset with two features - each feature will have a different mean but the same standard deviation for the two simulated samples.

```
# create positive class sample with 2 descriptive features
set.seed(1)
f1 <- rnorm(100, mean = 6, sd = 1.2)
set.seed(2)
f2 <- rnorm(100, mean = 6, sd = 1.2)
P.data <- cbind(f1, f2)

# create negative class sample with 2 descriptive features
set.seed(3)
f1 <- rnorm(300, mean = 4, sd = 1.2)
set.seed(4)
f2 <- rnorm(300, mean = 4, sd = 1.2)
N.data <- cbind(f1, f2)

# combine all samples
data.mat <- data.frame(rbind(P.data, N.data),
                       Class = rep(c(1, 0), times = c(nrow(P.data), nrow(N.data))))
rownames(data.mat) <- paste("s", 1:(nrow(P.data) + nrow(N.data)), sep = "")

# plot data
plot(subset(data.mat, Class == 1)[,-3], col = "red", pch = 16, ylim = c(0, 9), xlim =
c(0, 9), xlab = "Feature 1", ylab = "Feature 2")
points(subset(data.mat, Class == 0)[,-3], col = "blue", pch = 16)
```



Classifier evaluation

Split the data into training (80%) and test set (20%)

We can use the caret package to do this.

```
# Load the "caret" package. This package is used to partition data, training and test
classification models etc.
set.seed(5003)
inTrain <- createDataPartition(data.mat$Class, p = 0.8)[[1]]
dataTrain <- data.mat[ inTrain, ]
dataTest  <- data.mat[-inTrain, ]
```

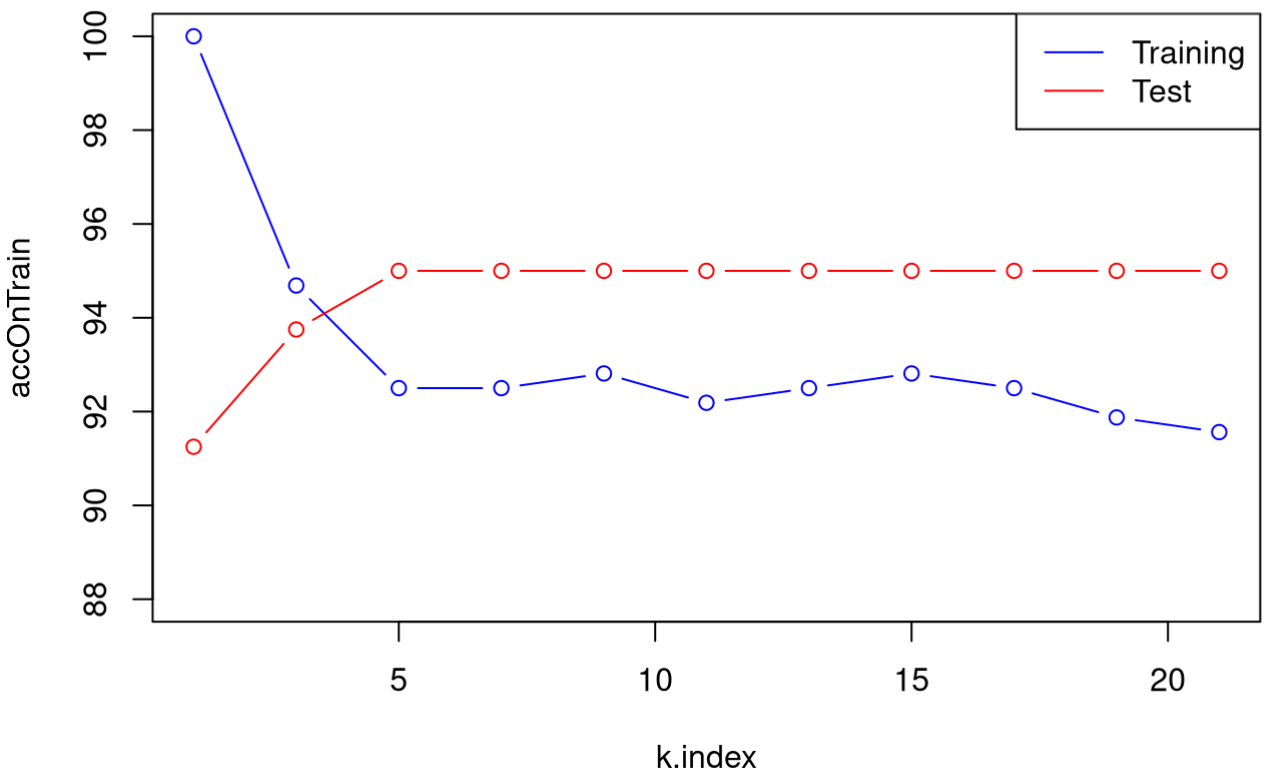
Demonstrate overfitting using test dataset

Here, let's see what the testing error is like if we first train and test using the same data and then if we only test on test set. The plot of the training and testing error shows that if we decrease k to lower than 5, testing error becomes higher than training error. i.e. test accuracy is lower than training accuracy. (Note that small k when used solely on training data can lead to overfitting)

```
accOnTrain <- accOnTest <- NULL

# Record the training and testing error for various parameters k
# Remember k represent how many neighbours we want to use to perform
# the classification.
k.index <- seq(from = 21, to = 1, by = -2)
for(k in k.index) {
  # Column 3 is the label here, so remove from training
  knnOnTrain <- knn(train=dataTrain[,-3], test=dataTrain[,-3], cl=dataTrain[,3], k=k)
  knnOnTest <- knn(train=dataTrain[,-3], test=dataTest[,-3], cl=dataTrain[,3], k=k)
  accOnTrain <- c(accOnTrain, sum(knnOnTrain == dataTrain[,3]) / nrow(dataTrain) * 10
0)
  accOnTest <- c(accOnTest, sum(knnOnTest == dataTest[,3]) / nrow(dataTest) * 100)
}

plot(k.index, accOnTrain, type="b", col="blue", ylim=c(88, 100))
lines(k.index, accOnTest, type="b", col="red")
legend("topright", c("Training", "Test"), col=c("blue", "red"), lty=c(1,1))
```



Cross validation of classification

Here, we demonstrate using cross validation on the `Sonar` dataset. We will use 10-fold crossvalidation to compare between `lda` and `knn` methods.

```
# Sonar is a dataset in the mlbench package
data(Sonar)
dim(Sonar)
```

```
## [1] 208  61
```

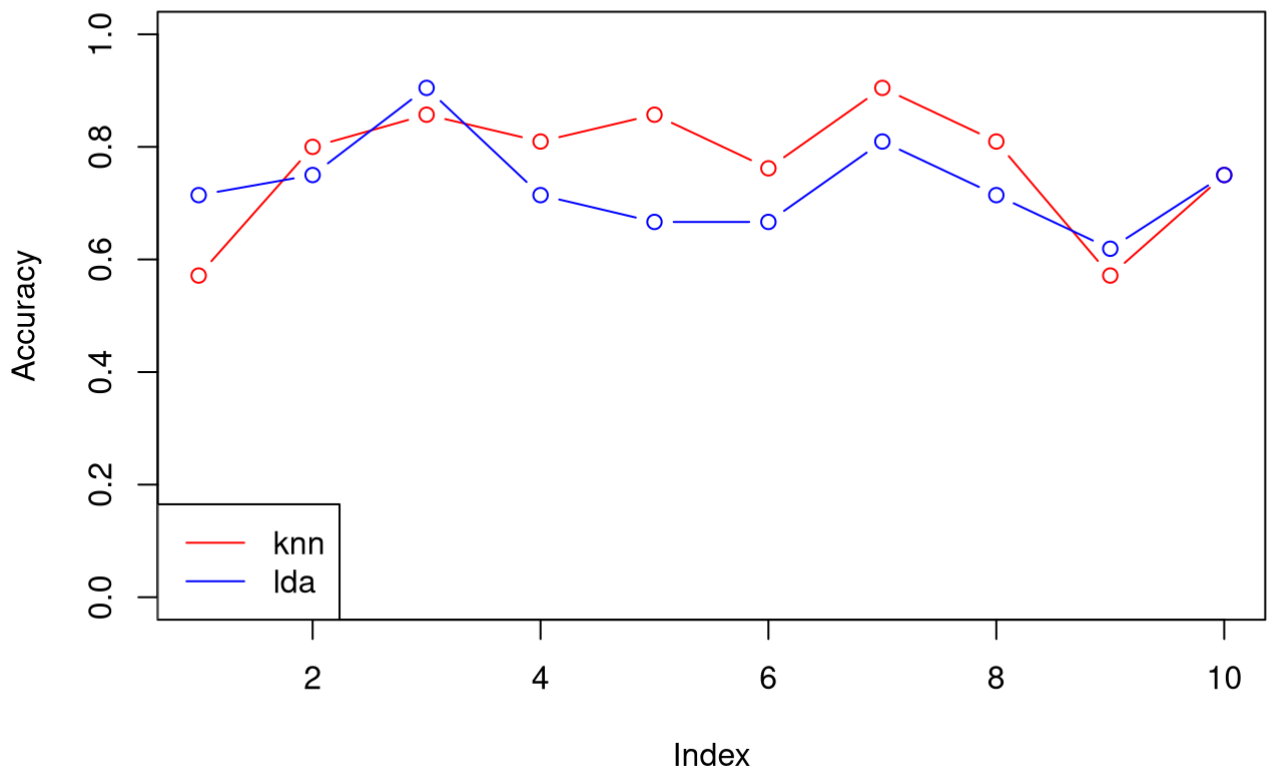
```
# create cross validation folds
library(caret)
set.seed(1)
folds <- createFolds(Sonar$Class, k=10)
# apply 10-fold cross-validation
knn.TP <- knn.TN <- knn.FP <- knn.FN <- c()
lda.TP <- lda.TN <- lda.FP <- lda.FN <- c()
knn.preds <- lapply(folds, function(f) knn(train = Sonar[-f,-61],
                                           test = Sonar[f,-61],
                                           cl = Sonar$Class[-f],
                                           k = 5))
lda.models <- lapply(folds, function(f) lda(Class ~ ., data = Sonar[-f, ]))
lda.probs <- mapply(function(model, f) predict(model,
                                               newdata = Sonar[f, -61])$posterior[,
"M"],
                    lda.models, folds)
lda.preds <- lapply(lda.probs, function(p) factor(ifelse(p > 0.5, "M", "R")))
# Determine the observed classes in each fold
obs <- lapply(folds, function(x) Sonar$Class[x])
# Compute the confusion matrix output for each fold
knn.metrics <- mapply(function(pred, obs) confusionMatrix(data = pred, reference = ob
s),
                      pred = knn.preds, obs = obs,
                      SIMPLIFY = FALSE)
lda.metrics <- mapply(function(pred, obs) confusionMatrix(data = pred, reference = ob
s),
                      pred = lda.preds, obs = obs,
                      SIMPLIFY = FALSE)
# Inspect the confusionMatrix output list structure
str(knn.metrics[[1]])
```

```
## List of 6
##  $ positive: chr "M"
##  $ table   : 'table' int [1:2, 1:2] 10 1 8 2
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ Prediction: chr [1:2] "M" "R"
##   .. ..$ Reference : chr [1:2] "M" "R"
##  $ overall : Named num [1:7] 0.571 0.113 0.34 0.782 0.524 ...
##   ..- attr(*, "names")= chr [1:7] "Accuracy" "Kappa" "AccuracyLower" "AccuracyUppe
r" ...
##  $ byClass : Named num [1:11] 0.909 0.2 0.556 0.667 0.556 ...
##   ..- attr(*, "names")= chr [1:11] "Sensitivity" "Specificity" "Pos Pred Value" "N
eg Pred Value" ...
##  $ mode    : chr "sens_spec"
##  $ dots    : list()
##  - attr(*, "class")= chr "confusionMatrix"
```

```
# Extract the elements of the table
knn.TP <- vapply(knn.metrics, function(cm) cm[["table"]][1, 1], numeric(1))
knn.FP <- vapply(knn.metrics, function(cm) cm[["table"]][1, 2], numeric(1))
knn.TN <- vapply(knn.metrics, function(cm) cm[["table"]][2, 2], numeric(1))
knn.FN <- vapply(knn.metrics, function(cm) cm[["table"]][2, 1], numeric(1))
knn.accuracy <- vapply(knn.metrics, function(cm) cm[["overall"]]["Accuracy"], numeric
(1))
# Verify the computations
ft <- knn.metrics[[1]]$table
(ft[1, 1] + ft[2, 2])/(sum(ft))
```

```
## [1] 0.5714286
```

```
# Compute the same for the LDA model
lda.TP <- vapply(lda.metrics, function(cm) cm[["table"]][1, 1], numeric(1))
lda.FP <- vapply(lda.metrics, function(cm) cm[["table"]][1, 2], numeric(1))
lda.TN <- vapply(lda.metrics, function(cm) cm[["table"]][2, 2], numeric(1))
lda.FN <- vapply(lda.metrics, function(cm) cm[["table"]][2, 1], numeric(1))
lda.accuracy <- vapply(lda.metrics, function(cm) cm[["overall"]]["Accuracy"], numeric
(1))
# Plot the accuracy in each of the 10 folds
plot(knn.accuracy, type = "b", col = "red", ylim = c(0,1), ylab = "Accuracy")
lines(lda.accuracy, type = "b", col = "blue")
legend("bottomleft", c("knn", "lda"), col = c("red", "blue"), lty = c(1,1))
```



##ROC curve

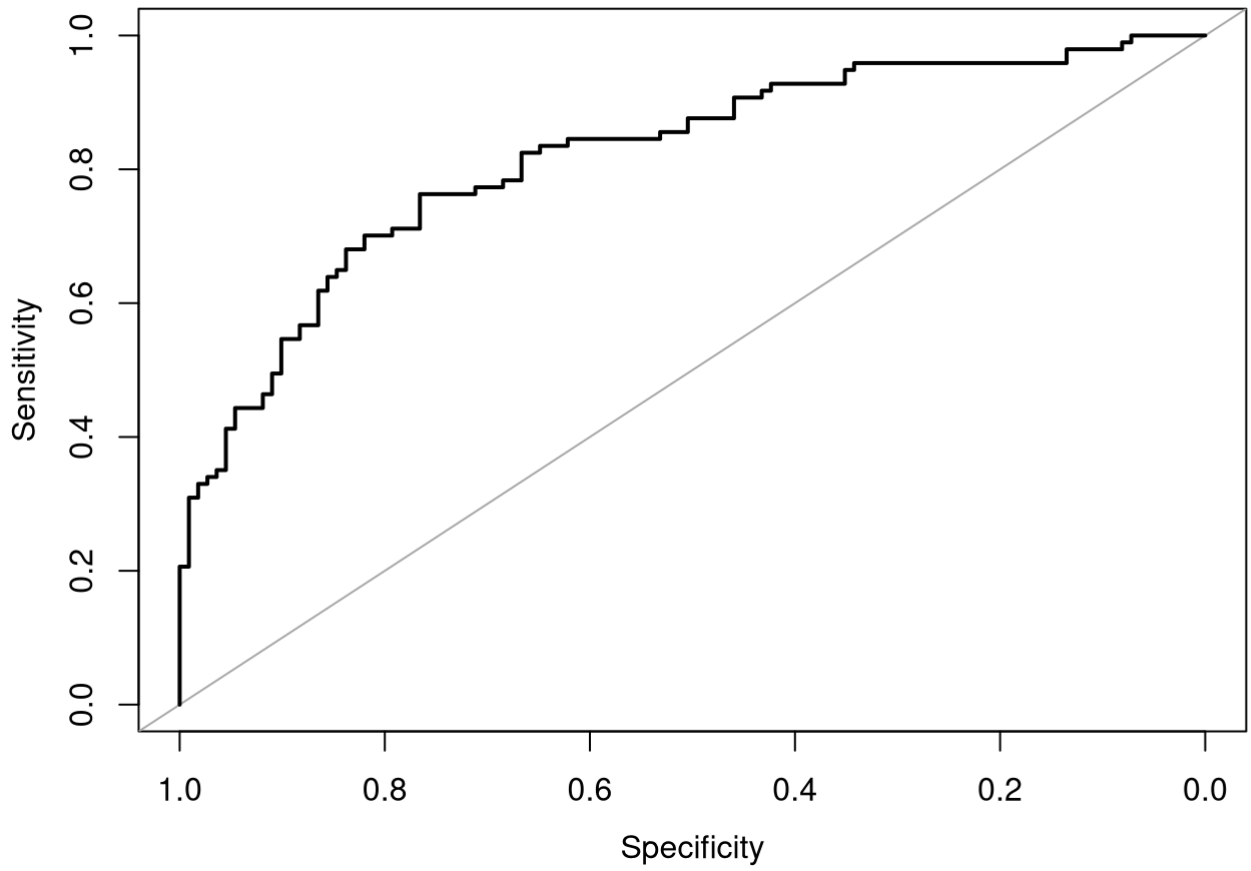
```
# Train LDA model with cross validation with the caret package
# Saves the prediction in each fold and the class probabilities
train_control <- trainControl(method = "cv", number = 10, savePredictions = "all", cl
assProbs = TRUE)
lda.model <- train(Class ~ ., data = Sonar, trControl = train_control, method = "lda"
)

lda.roc <- roc(lda.model$pred$obs,lda.model$pred$R)
```

```
## Setting levels: control = M, case = R
```

```
## Setting direction: controls < cases
```

```
plot.roc(lda.roc, asp=NA)
```

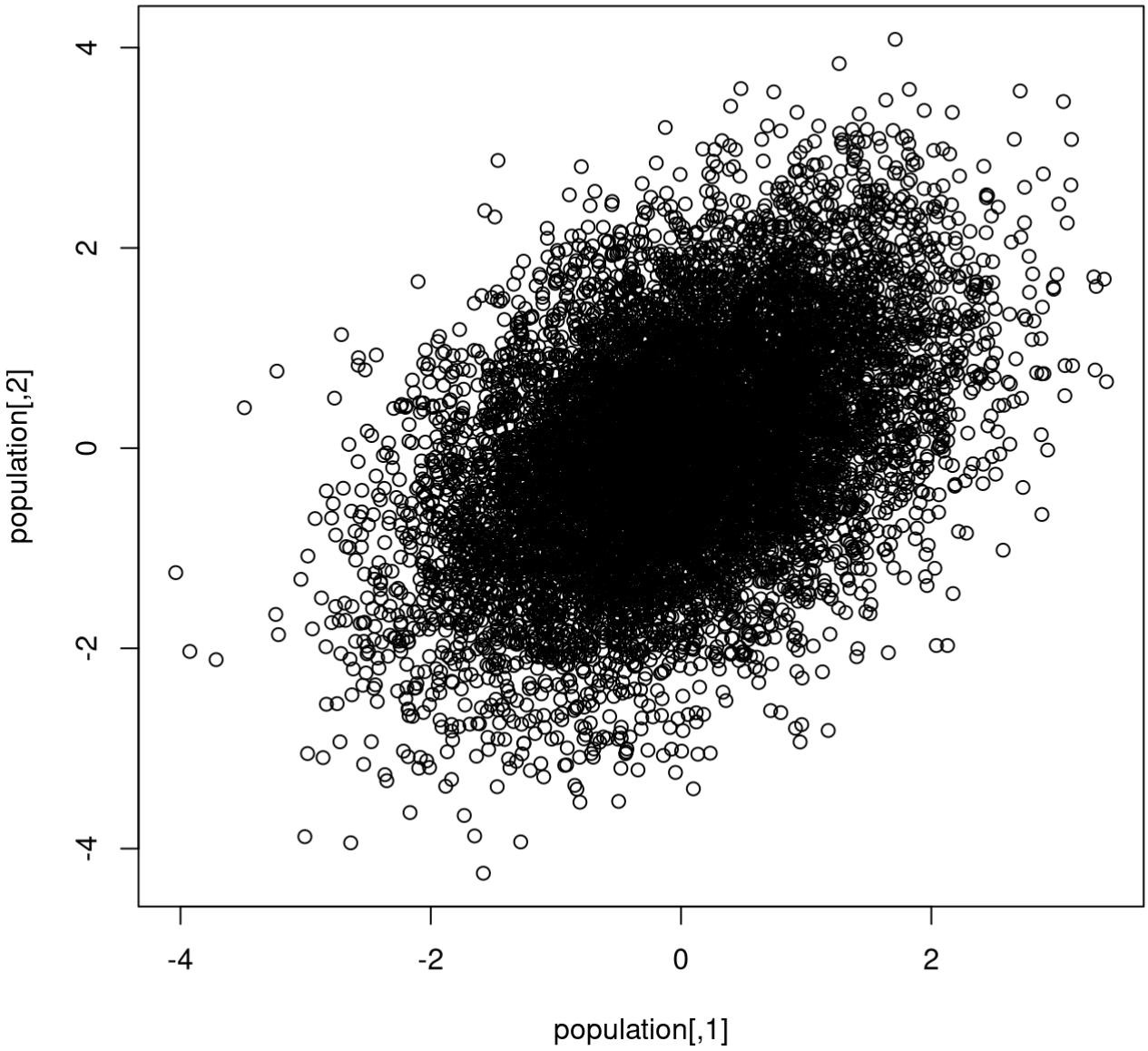



```
# Compute the AUC
auc(lda.roc)
```

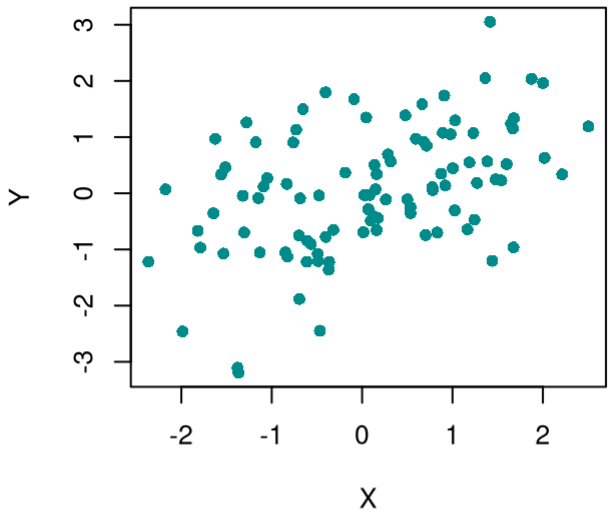
```
## Area under the curve: 0.8191
```

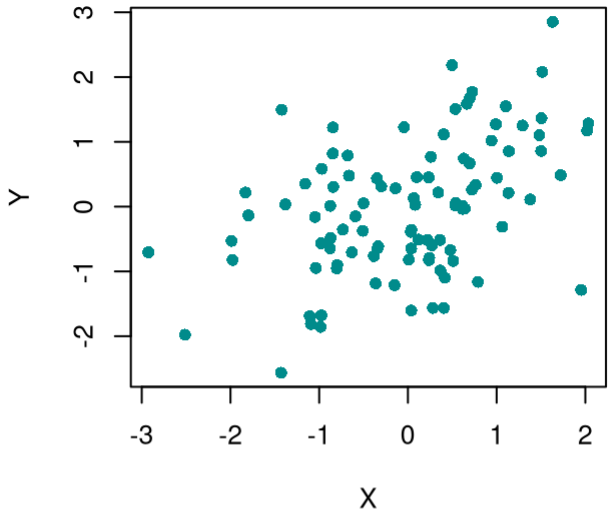
Demonstration of bootstrap sampling for parameter estimation

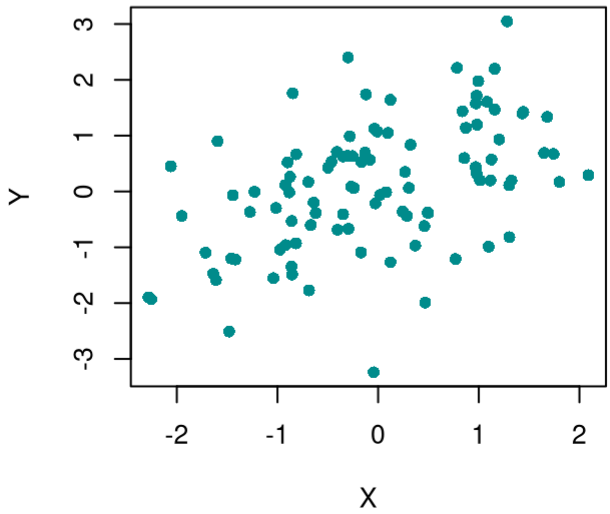
```
library(MASS)
# suppose the true population is as follows:
set.seed(2)
population <- mvrnorm(10000, mu = c(0,0), Sigma = matrix(c(1, 0.5,
                                                           0.5, 1.25), ncol = 2), emp
irical = TRUE)
plot(population)
```

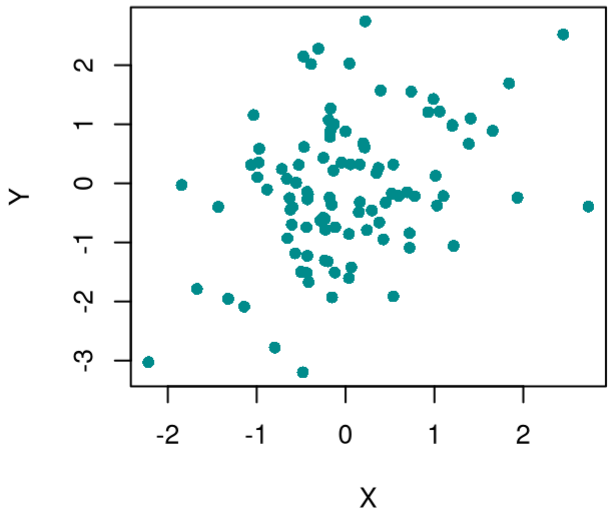



```
# sampling from the population
par(mfrow=c(2,2))
for (i in 1:4) {
  s <- population[sample(x=1:nrow(population), size=100, replace=TRUE),]
  plot(s, pch=16, col="cyan4", xlab="X", ylab="Y")
}
```


Y
X
-3
-2
-1
0
1
2
3


Y
X
-3
-2
-1
0
1
2
3

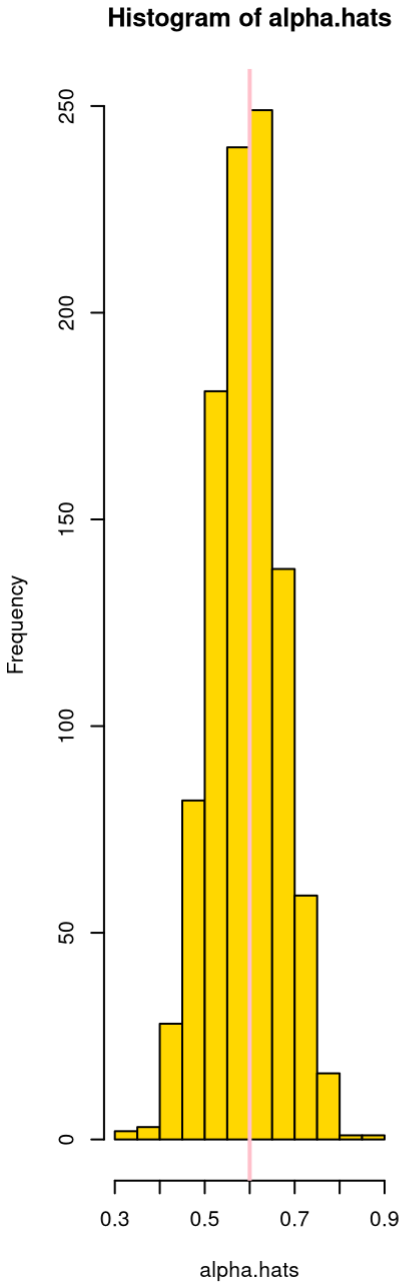

Y
X
-3
-2
-1
0
1
2
3

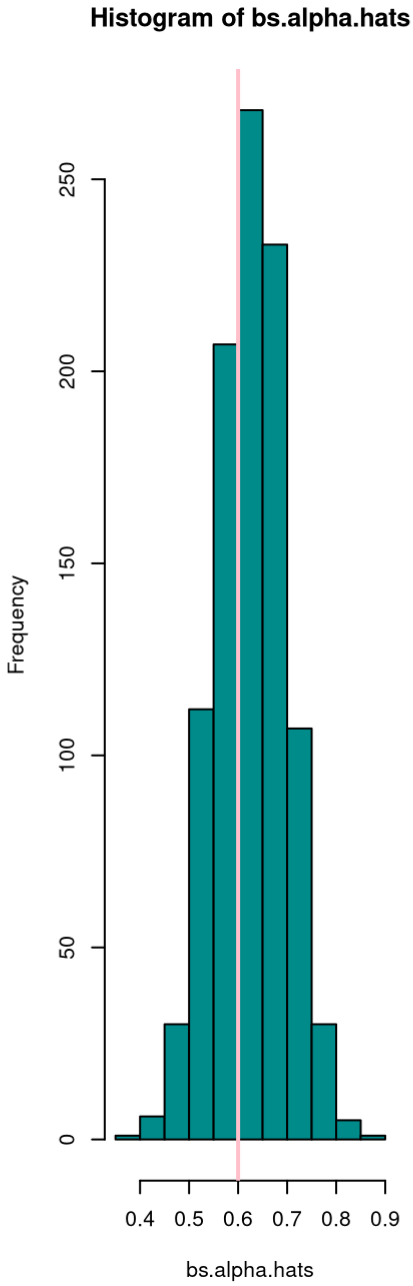

Y
X
-3
-2
-1
0
1
2

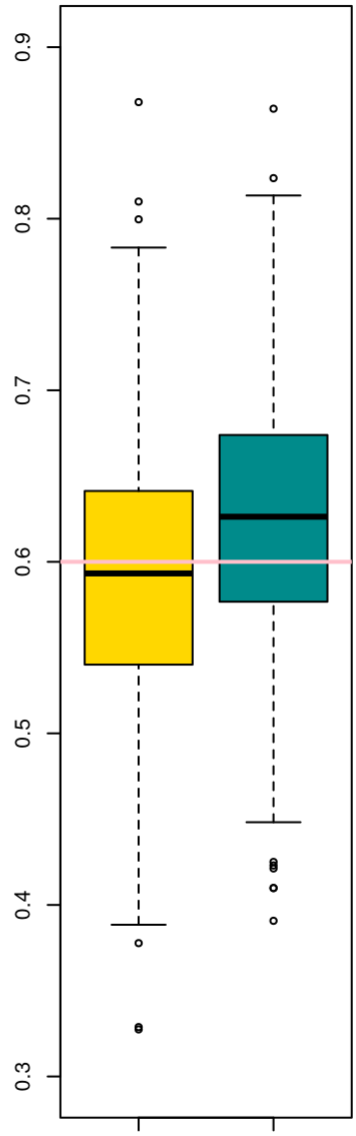
```
# sampling 1000 times from the population and estimating alpha
alpha.hats <- c()
for(i in 1:1000) {
  s <- population[sample(x=1:nrow(population), size=100, replace=TRUE),]
  sigma.hats <- apply(s, 2, var)
  cor.hat <- cor(s[,1], s[,2])
  alpha.hats <- c(alpha.hats,  (sigma.hats[2] - cor.hat) / (sigma.hats[1] + sigma.hat
s[2] - 2*cor.hat))
}

# suppose now we only have one sample from the population and now need to rely on boo
tstrap approach
s <- population[sample(x=1:nrow(population), size=100, replace=TRUE),]
# bootstrap sampling from the single sample data and estimating alpha
bs.alpha.hats <- c()
for(i in 1:1000) {
  bs <- s[sample(x=1:nrow(s), size=nrow(s), replace=TRUE),]
  bs.sigma.hats <- apply(bs, 2, var)
  bs.cor.hat <- cor(bs[,1], bs[,2])
  bs.alpha.hats <- c(bs.alpha.hats,  (bs.sigma.hats[2] - bs.cor.hat) / (bs.sigma.hats
[1] + bs.sigma.hats[2] - 2*bs.cor.hat))
}

# plot and compare alpha estimated from sampling from true population and bootstrap s
ampling from a single sample data
par(mfrow=c(1, 3))
hist(alpha.hats, col="gold")
abline(v=0.6, col="pink", lwd=2)
hist(bs.alpha.hats, col="cyan4")
abline(v=0.6, col="pink", lwd=2)
boxplot(alpha.hats, bs.alpha.hats, ylim=c(0.3, 0.9), col=c("gold", "cyan4"))
abline(h=0.6, col="pink", lwd=2)
```


Histogram of alpha.hats
Frequency
0
50
100
150
200
250
0.3
0.5
0.7
0.9
alpha.hats


Histogram of bs.alpha.hats
Frequency
0
50
100
150
200
250
0.4
0.5
0.6
0.7
0.8
0.9
bs.alpha.hats


0.3
0.4
0.5
0.6
0.7
0.8
0.9