1 PQHS 471 Notes

1.1 Doughnut experiment

We simulate data in a 2-dimensional doughnut shape. A 2D doughnut has two circles. Data that are between the circles have a high probability to be in class 1, and those inside the inner circle or outside the outer circle have a high probability to be in class 0. We compare the performance of various prediction models on this data.

You could similarly generate a high-dimensional doughnut data, although it would be difficult to visualize.

```r
library(splines)  ## ns()
library(gam)  ## gam() allowing multiple smoothing splines
library(tree)  ## tree()
library(randomForest)  ## randomForest()
library(xgboost)
library(e1071)  ## svm()
library(class)  ## knn()
library(rgl)  ## for 3D plots

## Grid for displaying the fitted prediction models
gridlen = 201
gridx1 = seq(-3, 3, length.out=gridlen)
gridx2 = gridx1
g1 = rep(gridx1, each=gridlen)
g2 = rep(gridx2, gridlen)
## 10D grid
testgrid = data.frame(x1=g1, x2=g2, r=sqrt(g1^2+g2^2),
                      x3=rnorm(gridlen^2), x4=rnorm(gridlen^2),
                      x5=rnorm(gridlen^2), x6=rnorm(gridlen^2),
                      x7=rnorm(gridlen^2), x8=rnorm(gridlen^2),
                      x9=rnorm(gridlen^2), x10=rnorm(gridlen^2))
## 2D grid
testgrid2 = testgrid[,1:2]
```

1.1.1 Generate data

Set seed and sample size. Define the 2D doughnut region by specifying the inner and outer radii. Assign a high probability to the doughnut region and a low probability outside this region. You may treak these numbers to see how much the performance would change for various methods. To generate the outcome deterministically, set `pin=1` and `pout=0`. To generate the outcome probabilistically, set these probabilities to be between 0 and 1.

```r
N = 500
Ntrain = 400
radius1 = 1; radius2 = 1.5  ## the radii that define our doughnut
pin = 1; pout = 0  ## the probabilities of class 1 for inside and outside the doughnut region
```

Generate some features and compute the radius using the first two features. Only the first two features determine the outcome.

```r
set.seed(2018)
for(i in 1:10)
  assign(paste("x", i, sep=''), rnorm(N))

r = sqrt(x1^2 + x2^2)
table(cut(r, c(0, radius1, radius2, Inf)))  ## check the distribution
```

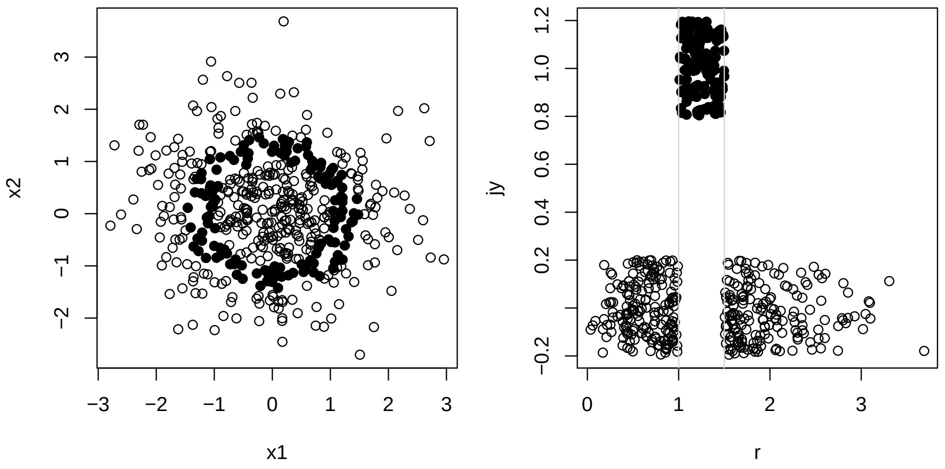
Generate the outcome. Plot the data.

```
p = ifelse(r>radius1 & r<radius2, pin, pout)

y = ifelse(runif(N)<p, 1, 0)  ## Generate the outcome according to the probabilities.
table(y, cut(r, c(0, radius1, radius2, Inf)))

jy = jitter(y)  ## for some plots

plot(x1, x2); points(x1[y==1], x2[y==1], pch=19)
plot(r, jy); points(r[y==1], jy[y==1], pch=19)
abline(v=c(radius1, radius2), col='lightgrey')
```



Now split the data into training and test sets.

```
train = sample(1:N, Ntrain)

trainy1 = 1:N %in% train & y==1
testy1 = !1:N %in% train & y==1

plot(x1, x2, type='n')
points(x1[train], x2[train])
points(x1[trainy1], x2[trainy1], pch=19)
points(x1[-train], x2[-train], col=2)
points(x1[testy1], x2[testy1], pch=19, col=2)

## Data including the noise features
mydata = data.frame(x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, y)
mydata.test = data.frame(mydata, r=r)[-train,]

## Data with only the true predictors
mydata2 = data.frame(x1, x2, y)
mydata2.test = mydata2[-train,]

y.test = y[-train]
```

1.1.2 Logistic regression

Only linear terms in x1 and x2.

```
mod1 = glm(y ~ x1 + x2, mydata, subset=train, family=binomial)
summary(mod1)
```

```r
plot(predict(mod1, mydata.test, type="response"), jitter(y.test))
cor(predict(mod1, mydata.test, type="response"), y.test)

## obtain predicted probabilities for the grid for plotting
mod1.pred = matrix(predict(mod1, testgrid, type='response'), gridlen)
## make various plots
plot(x1, x2)
points(x1[y==1], x2[y==1], pch=19)
contour(gridx1, gridx2, mod1.pred, add=T, col=2, lwd=3, levels=c(0.25,0.5,0.75))

persp(gridx1, gridx2, mod1.pred, theta=30, phi=30)
plot3d(g1, g2, mod1.pred)
```

Now add quadratic terms in x1 and x2

```r
## without the x1*x2 term
mod2 = glm(y ~ x1 + x2 + I(x1^2) + I(x2^2), mydata, subset=train, family=binomial)
## with the x1*x2 term
mod2 = glm(y ~ x1 + x2 + I(x1^2) + I(x2^2) + I(x1*x2), mydata, subset=train, family=binomial)
summary(mod2)

plot(predict(mod2, mydata.test, type="response"), jitter(y.test))
cor(predict(mod2, mydata.test, type="response"), y.test)

mod2.pred = matrix(predict(mod2, testgrid, type='response'), gridlen)
plot(x1, x2)
points(x1[y==1], x2[y==1], pch=19)
contour(gridx1, gridx2, mod2.pred, add=T, col=2, lwd=3, levels=c(0.25,0.5,0.75))

plot3d(g1, g2, mod2.pred)
```

Logistic regression with splines on x1 and x2. You may treak the df parameters.

```r
## Natural splines on x1 and x2.
mod3 = glm(y ~ ns(x1, df=5) + ns(x2, df=5), mydata, subset=train, family=binomial)
## Smoothing splines on x1 and x2.
mod3 = gam(y ~ s(x1, df=5) + s(x2, df=5), mydata, subset=train, family=binomial)
summary(mod3)

plot(predict(mod3, mydata.test, type="response"), jitter(y.test))
cor(predict(mod3, mydata.test, type="response"), y.test)

mod3.pred = matrix(predict(mod3, testgrid, type='response'), gridlen)
plot(x1, x2)
points(x1[y==1], x2[y==1], pch=19)
contour(gridx1, gridx2, mod3.pred, add=T, col=2, lwd=3, levels=c(0.25,0.5,0.75))

plot3d(g1, g2, mod3.pred)
```

Using splines on the radius $r = \sqrt{x_1^2 + x_2^2}$. You may treak the df parameters.

```r
mod4 = glm(y[train] ~ ns(r, df=4), data=data.frame(y,r)[train,], family=binomial)
summary(mod4)

plot(predict(mod4, data.frame(y,r)[-train,], type="response"), jitter(y.test))
cor(predict(mod4, data.frame(y,r)[-train,], type="response"), y.test)

mod4.pred = matrix(predict(mod4, testgrid, type='response'), gridlen)
```

```r
plot(x1, x2)
points(x1[y==1], x2[y==1], pch=19)
contour(gridx1, gridx2, mod4.pred, add=T, col=2, lwd=3, levels=c(0.25,0.5,0.75))

plot3d(g1, g2, mod4.pred)
```

1.1.3 Classification trees

```r
tree1 = tree(factor(y) ~ ., data=mydata, subset=train)
summary(tree1)

tree1.test = predict(tree1, mydata.test)[,2]

plot(tree1.test, jitter(y.test))
cor(tree1.test, y.test)
table(tree1.test>.5, y.test)
sum(as.numeric(tree1.test>.5) == y.test) / length(y.test)

## to prune the tree
tree1 = prune.misclass(tree1, best=9)
tree1.test = predict(tree1, mydata.test)[,2]
table(tree1.test>.5, y.test)

tree1.pred = matrix(predict(tree1, testgrid)[,2], gridlen)

plot(x1, x2)
points(x1[y==1], x2[y==1], pch=19)
contour(gridx1, gridx2, tree1.pred, add=T, col=2, lwd=3,levels=c(0.25,0.5,0.75))

plot3d(g1, g2, tree1.pred)
```

1.1.4 Random forest

RF seems to be worse than a classification tree for a small N but better for a large N. You may treak the `ntree` and `mtry` parameters.

```r
rf1 = randomForest(factor(y) ~ ., data=mydata, subset=train,
                   ntree=500, mtry=3, importance=TRUE)
table(predict(rf1, mydata.test), y.test)
sum(predict(rf1, mydata.test) == y.test) / length(y.test)

rf1.pred = matrix(predict(rf1, testgrid), gridlen)

plot(x1, x2)
points(x1[y==1], x2[y==1], pch=19)
contour(gridx1, gridx2, rf1.pred, add=T, col=2, lwd=3, levels=c(0.25,0.5,0.75))

image(gridx1, gridx2, matrix(as.numeric(rf1.pred), gridlen))

persp(gridx1, gridx2, rf1.pred, theta=30, phi=70)

plot3d(g1, g2, rf1.pred)
```

1.1.5 SVM

Only x1 and x2 are considered. You may play with the cost parameter.

```
svm1 = svm(factor(y) ~ x1 + x2, data=mydata, subset=train,
           kernel="polynomial", cost=1, scale=FALSE)
table(predict(svm1, mydata.test), y.test)
sum(predict(svm1, mydata.test) == y.test) / length(y.test)

svm1.pred = matrix(predict(svm1, testgrid), gridlen)

plot(x1, x2)
points(x1[y==1], x2[y==1], pch=19)
contour(gridx1, gridx2, svm1.pred, add=T, col=2, lwd=3, levels=c(0.25,0.5,0.75))

plot3d(g1, g2, svm1.pred)
```

Now add quadratic terms.

```
svm2 = svm(factor(y) ~ x1 + x2 + I(x1^2) +I(x2^2),
           data=mydata, subset=train,
           kernel="linear", cost=1, scale=FALSE)
svm2 = svm(factor(y) ~ x1 + x2 + I(x1^2) +I(x2^2) + I(x1*x2),
           data=mydata, subset=train,
           kernel="linear", cost=1, scale=FALSE)

table(predict(svm2, mydata.test), y.test)
sum(predict(svm2, mydata.test) == y.test) / length(y.test)

svm2.pred = matrix(predict(svm2, testgrid), gridlen)

plot(x1, x2)
points(x1[y==1], x2[y==1], pch=19)
contour(gridx1, gridx2, svm2.pred, add=T, nlevels=2, col=2, lwd=3)

plot3d(g1, g2, svm2.pred)
```

Using the radial kernel.

```
svm3 = svm(factor(y) ~ x1 + x2, data=mydata, subset=train,
           kernel="radial", cost=1, scale=FALSE)
table(predict(svm3, mydata.test), y.test)
sum(predict(svm3, mydata.test) == y.test) / length(y.test)

## obtain predicted values for the grid for plotting
svm3.pred = matrix(predict(svm3, testgrid), gridlen)

plot(x1, x2)
points(x1[y==1], x2[y==1], pch=19)
contour(gridx1, gridx2, svm3.pred, add=T, nlevels=2, col=2, lwd=3)

plot3d(g1, g2, svm3.pred)
```

1.1.6 KNN

KNN using only x1 and x2. You may treak the parameter `k`.

```
table(knn(mydata2[train,1:2], mydata2.test[,1:2], y[train], k=5), y.test)
sum(knn(mydata2[train,1:2], mydata2.test[,1:2], y[train], k=5) == y.test)

knn1.pred = matrix(knn(mydata2[train,1:2], testgrid2, y[train], k=5), gridlen)

plot(x1, x2)
points(x1[y==1], x2[y==1], pch=19)
contour(gridx1, gridx2, knn1.pred, add=T, nlevels=2, col=2, lwd=3)

persp(gridx1, gridx2, knn1.pred, theta=30, phi=50)
```

KNN using all features.

```
table(knn(mydata[train,1:10], mydata.test[,1:10], y[train], k=5), y.test)
sum(knn(mydata[train,1:10], mydata.test[,1:10], y[train], k=5) == y.test)

knn2.pred = matrix(knn(mydata[train,1:10], testgrid[,-3], y[train], k=5), gridlen)

plot(x1, x2)
points(x1[y==1], x2[y==1], pch=19)
contour(gridx1, gridx2, knn2.pred, add=T, nlevels=2, col=2, lwd=3)

persp(gridx1, gridx2, knn2.pred, theta=30, phi=50)
```

1.1.7 Boosting

1.1.8 NN?