ISLR 8.4 Conceptual

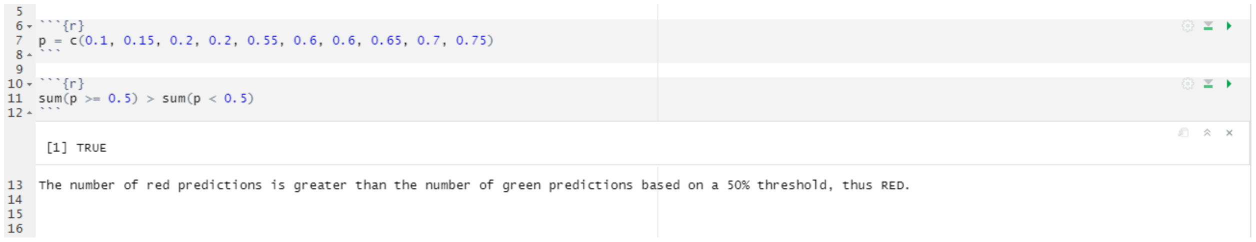
5. Suppose we produce ten bootstrapped samples from a data set containing red and green classes. We then apply a classification tree to each bootstrapped sample and, for a specific value of X, produce 10 estimates of P(Class is Red|X):

0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, and 0.75.

There are two common ways to combine these results together into a single class prediction. One is the majority vote approach discussed in this chapter. The second approach is to classify based on the average probability. In this example, what is the final classification under each of these two approaches?

p = c(0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, 0.75)

Majority approach:

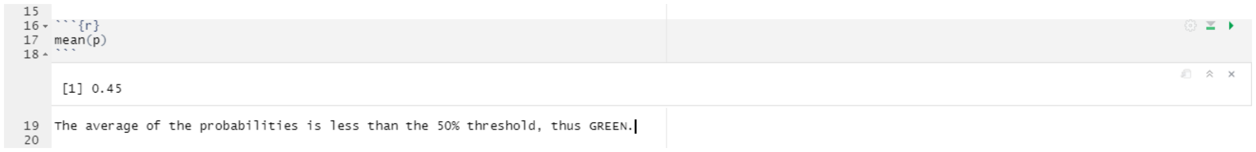


~~~
```{r}
p = c(0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, 0.75)
```

```{r}
sum(p >= 0.5) > sum(p < 0.5)
```
~~~

```
[1] TRUE
```

The number of red predictions is greater than the number of green predictions based on a 50% threshold, thus RED.

Average approach:



~~~
```{r}
mean(p)
```
~~~

```
[1] 0.45
```

The average of the probabilities is less than the 50% threshold, thus GREEN.

R Notebook

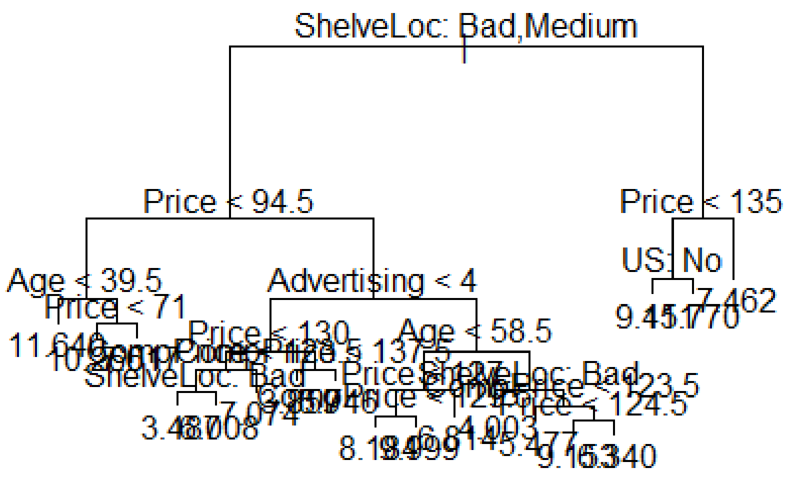
8. In the lab, a classification tree was applied to the Carseats data set after converting Sales into a qualitative response variable. Now we will seek to predict Sales using regression trees and related approaches, treating the response as a quantitative variable.

(a) Split the data set into a training set and a test set.

```
library(ISLR)
attach(Carseats)
set.seed(1)

train = sample(dim(Carseats)[1], dim(Carseats)[1]/2)
Carseats.train = Carseats[train, ]
Carseats.test = Carseats[-train, ]
```

(b) Fit a regression tree to the training set. Plot the tree, and interpret the results. What test MSE do you obtain?

```
library(tree)

## Warning: package 'tree' was built under R version 4.2.1

tree.carseats = tree(Sales ~ ., data = Carseats.train)
summary(tree.carseats)

##
## Regression tree:
## tree(formula = Sales ~ ., data = Carseats.train)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Age"         "Advertising" "CompPrice"
## [6] "US"
## Number of terminal nodes:  18
## Residual mean deviance:  2.167 = 394.3 / 182
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -3.88200 -0.88200 -0.08712  0.00000  0.89590  4.09900

plot(tree.carseats)
text(tree.carseats, pretty = 0)
```

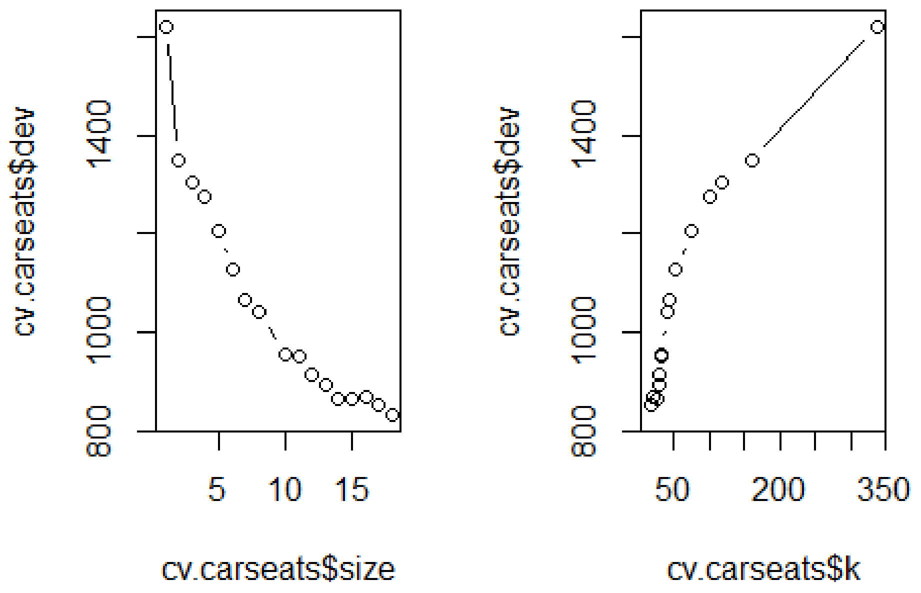


```
pred.carseats = predict(tree.carseats, Carseats.test)
mean((Carseats.test$Sales - pred.carseats)^2)

## [1] 4.922039
```

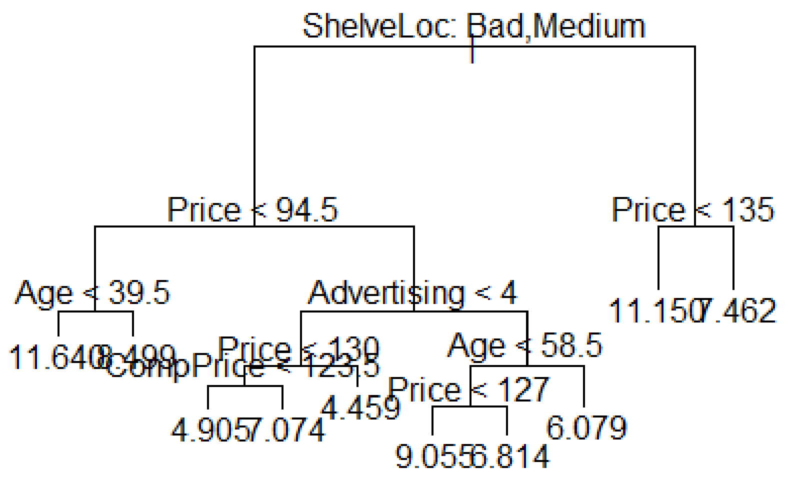
The test MSE is about 4.92.

(c) Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test MSE?

```
cv.carseats = cv.tree(tree.carseats, FUN = prune.tree)
par(mfrow = c(1, 2))
plot(cv.carseats$size, cv.carseats$dev, type = "b")
plot(cv.carseats$k, cv.carseats$dev, type = "b")
```



```
# Best size = 9
pruned.carseats = prune.tree(tree.carseats, best = 9)
par(mfrow = c(1, 1))
plot(pruned.carseats)
text(pruned.carseats, pretty = 0)
```



```
pred.pruned = predict(pruned.carseats, Carseats.test)
mean((Carseats.test$Sales - pred.pruned)^2)

## [1] 4.918134
```

Pruning the tree in this case increases the test MSE to 4.91.

(d) Use the bagging approach in order to analyze this data. What test MSE do you obtain? Use the importance() function to determine which variables are most important.

```
library(randomForest)

## Warning: package 'randomForest' was built under R version 4.2.1

## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

bag.carseats = randomForest(Sales ~ ., data = Carseats.train, mtry = 10,
ntree = 500,
    importance = T)
bag.pred = predict(bag.carseats, Carseats.test)
mean((Carseats.test$Sales - bag.pred)^2)

## [1] 2.657296

importance(bag.carseats)
```

```
##                %IncMSE IncNodePurity
## CompPrice   23.07909904    171.185734
## Income       2.82081527     94.079825
## Advertising 11.43295625     99.098941
## Population  -3.92119532     59.818905
## Price       54.24314632    505.887016
## ShelveLoc   46.26912996    361.962753
## Age         14.24992212    159.740422
## Education   -0.07662320     46.738585
## Urban        0.08530119      8.453749
## US           4.34349223     15.157608
```

Bagging improves the test MSE to 2.65. We also see that Price, ShelveLoc and Age are three most important predictors of Sale.

(e) Use random forests to analyze this data. What test MSE do you obtain? Use the importance() function to determine which variables are most important. Describe the effect of m, the number of variables considered at each split, on the error rate obtained.

```
rf.carseats = randomForest(Sales ~ ., data = Carseats.train, mtry = 5, ntree
= 500,
    importance = T)
rf.pred = predict(rf.carseats, Carseats.test)
mean((Carseats.test$Sales - rf.pred)^2)

## [1] 2.701665

importance(rf.carseats)

##               %IncMSE IncNodePurity
## CompPrice   19.8160444     162.73603
## Income       2.8940268     106.96093
## Advertising 11.6799573     106.30923
## Population  -1.6998805      79.04937
## Price       46.3454015     448.33554
## ShelveLoc   40.4412189     334.33610
## Age         12.5440659     169.06125
## Education    1.0762096      55.87510
## Urban        0.5703583      13.21963
## US           5.8799999      25.59797
```

In this case, random forest worsens the MSE on test set to 2.70. Changing m varies test MSE between 2.65 to 3. We again see that Price, ShelveLoc and Age are three most important predictors of Sale.

10. We now use boosting to predict Salary in the Hitters data set.

(a) Remove the observations for whom the salary information is unknown, and then log-transform the salaries.

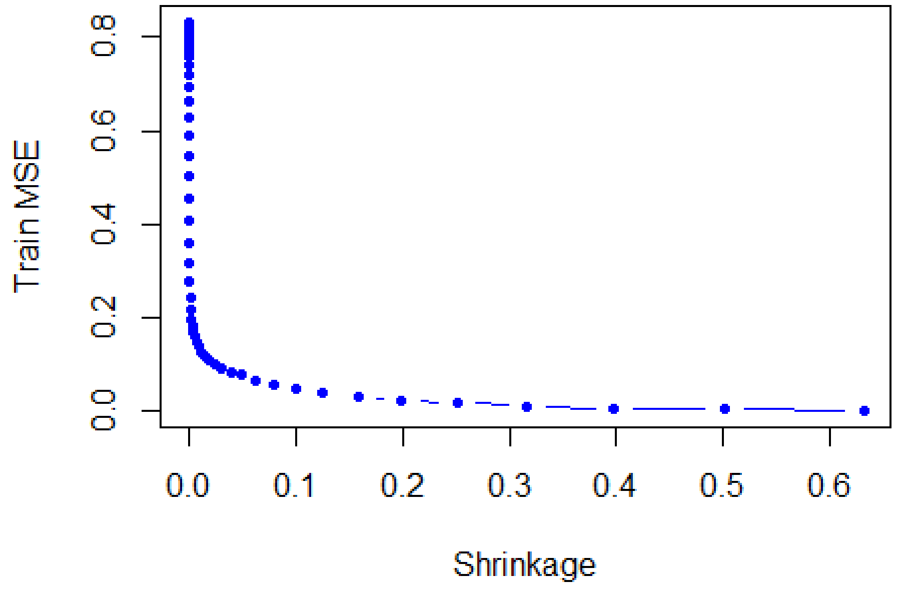
```
library(ISLR)
sum(is.na(Hitters$Salary))

## [1] 59

Hitters = Hitters[-which(is.na(Hitters$Salary)), ]
sum(is.na(Hitters$Salary))

## [1] 0

Hitters$Salary = log(Hitters$Salary)
```

(b) Create a training set consisting of the first 200 observations, and a test set consisting of the remaining observations.

```
train = 1:200
Hitters.train = Hitters[train, ]
Hitters.test = Hitters[-train, ]
```

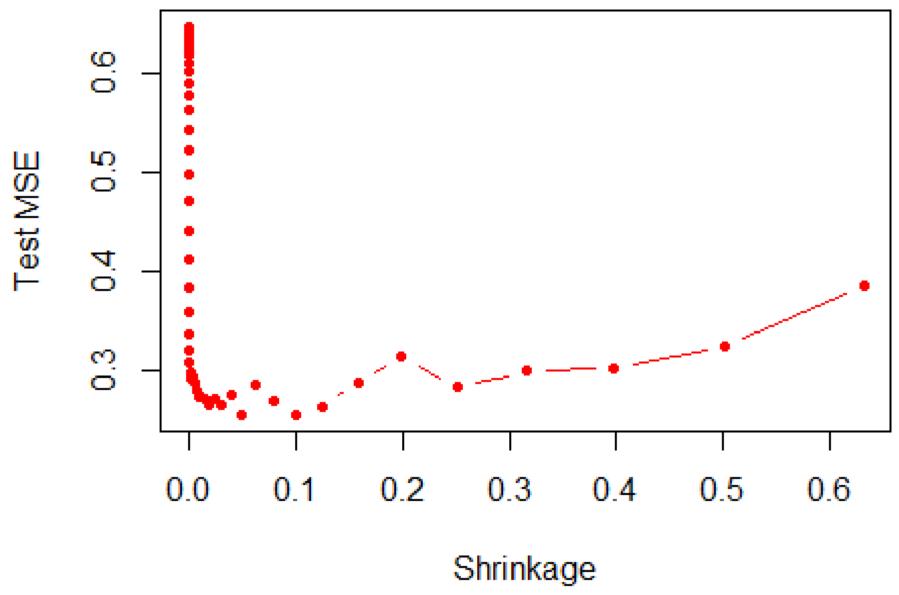
(c) Perform boosting on the training set with 1,000 trees for a range of values of the shrinkage parameter λ. Produce a plot with different shrinkage values on the x-axis and the corresponding training set MSE on the y-axis.

```
library(gbm)

## Warning: package 'gbm' was built under R version 4.2.1

## Loaded gbm 2.1.8

set.seed(103)
pows = seq(-10, -0.2, by = 0.1)
lambdas = 10^pows
length.lambdas = length(lambdas)
train.errors = rep(NA, length.lambdas)
test.errors = rep(NA, length.lambdas)
for (i in 1:length.lambdas) {
    boost.hitters = gbm(Salary ~ ., data = Hitters.train, distribution =
"gaussian",
        n.trees = 1000, shrinkage = lambdas[i])
    train.pred = predict(boost.hitters, Hitters.train, n.trees = 1000)
    test.pred = predict(boost.hitters, Hitters.test, n.trees = 1000)
    train.errors[i] = mean((Hitters.train$Salary - train.pred)^2)
    test.errors[i] = mean((Hitters.test$Salary - test.pred)^2)
}

plot(lambdas, train.errors, type = "b", xlab = "Shrinkage", ylab = "Train
MSE",
    col = "blue", pch = 20)
```



(d) Produce a plot with different shrinkage values on the x-axis and the corresponding test set MSE on the y-axis.

```
plot(lambdas, test.errors, type = "b", xlab = "Shrinkage", ylab = "Test MSE",
col = "red", pch = 20)
```



```
min(test.errors)

## [1] 0.2560507

lambdas[which.min(test.errors)]

## [1] 0.05011872
```

Minimum test error is obtained at λ=0.05.

(e) Compare the test MSE of boosting to the test MSE that results from applying two of the regression approaches seen in Chapters 3 and 6.

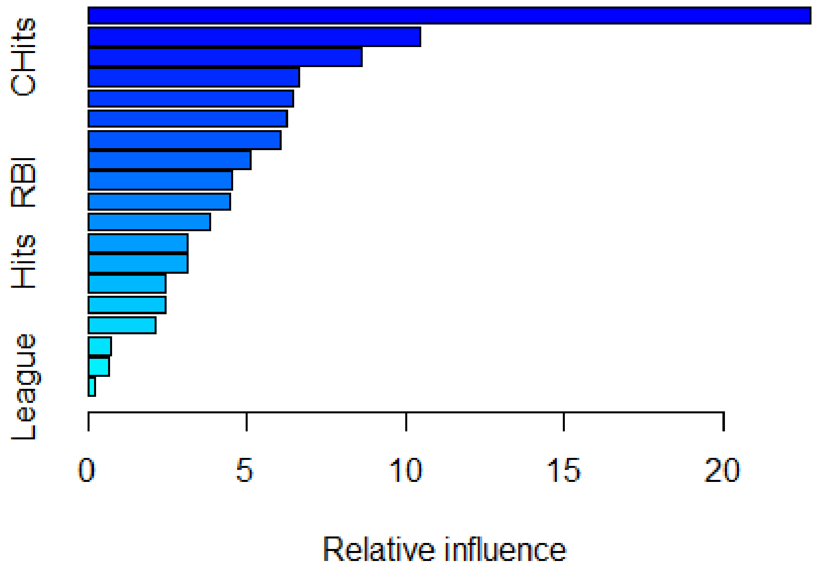
```
lm.fit = lm(Salary ~ ., data = Hitters.train)
lm.pred = predict(lm.fit, Hitters.test)
mean((Hitters.test$Salary - lm.pred)^2)

## [1] 0.4917959

library(glmnet)

## Warning: package 'glmnet' was built under R version 4.2.1

## Loading required package: Matrix

## Loaded glmnet 4.1-4

set.seed(134)
x = model.matrix(Salary ~ ., data = Hitters.train)
y = Hitters.train$Salary
```

```
x.test = model.matrix(Salary ~ ., data = Hitters.test)
lasso.fit = glmnet(x, y, alpha = 1)
lasso.pred = predict(lasso.fit, s = 0.01, newx = x.test)
mean((Hitters.test$Salary - lasso.pred)^2)

## [1] 0.4700537
```

Both linear model and regularization like Lasso have higher test MSE than boosting.

(f) Which variables appear to be the most important predictors in the boosted model?

```
boost.best = gbm(Salary ~ ., data = Hitters.train, distribution = "gaussian",
    n.trees = 1000, shrinkage = lambdas[which.min(test.errors)])
summary(boost.best)
```



```
##                 var    rel.inf
## CAtBat       CAtBat 22.7562681
## CWalks       CWalks 10.4279674
## CHits         CHits  8.6198109
## PutOuts     PutOuts  6.6159325
## Years         Years  6.4611683
## Walks         Walks  6.2331148
## CRBI           CRBI  6.0926744
## CHmRun       CHmRun  5.1076104
## RBI             RBI  4.5321678
## CRuns         CRuns  4.4728132
## Assists     Assists  3.8366575
## HmRun         HmRun  3.1554038
## Hits              Hits  3.1229284
## AtBat            AtBat  2.4338530
## Errors          Errors  2.4324185
## Runs              Runs  2.1425481
## Division      Division  0.7041949
## NewLeague NewLeague  0.6675446
## League          League  0.1849234
```

CAtBat, CRBI and CWalks are three most important variables in that order.

(g) Now apply bagging to the training set. What is the test set MSE for this approach?

```
library(randomForest)

set.seed(21)
rf.hitters = randomForest(Salary ~ ., data = Hitters.train, ntree = 500, mtry
= 19)
rf.pred = predict(rf.hitters, Hitters.test)
mean((Hitters.test$Salary - rf.pred)^2)

## [1] 0.2303919
```

Test MSE for bagging is about 0.23, which is slightly lower than the best test MSE for boosting.

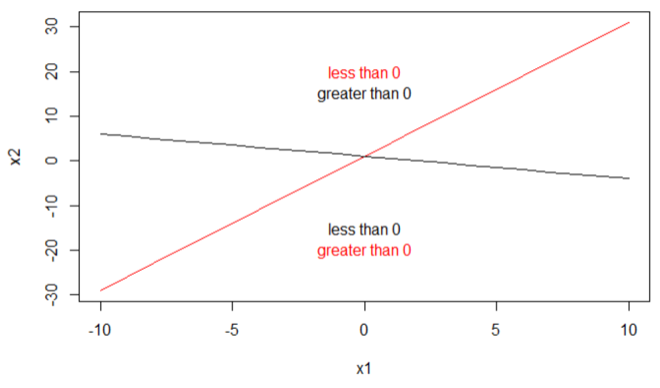
ISLR 9.7 Conceptual

1. This problem involves hyperplanes in two dimensions.

(a) Sketch the hyperplane 1 + 3X1 − X2 = 0. Indicate the set of points for which 1 + 3X1 − X2 > 0, as well as the set of points for which 1 + 3X1 − X2 < 0.

(b) On the same plot, sketch the hyperplane −2 + X1 + 2X2 = 0. Indicate the set of points for which −2 + X1 + 2X2 > 0, as well as the set of points for which −2 + X1 + 2X2 < 0.

~~~
```{r}
x1 = -10:10
x2 = 1 + 3 * x1
plot(x1, x2, type = "l", col = "red")
text(c(0), c(-20), "greater than 0", col = "red")
text(c(0), c(20), "less than 0", col = "red")
lines(x1, 1 - x1/2)
text(c(0), c(-15), "less than 0")
text(c(0), c(15), "greater than 0")
```
~~~

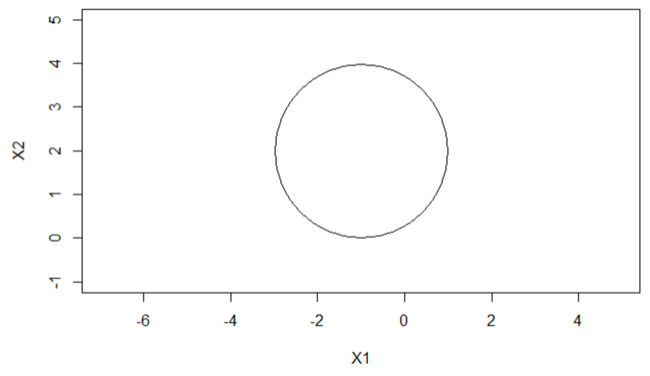


2. We have seen that in p = 2 dimensions, a linear decision boundary takes the form β0+β1X1+β2X2 = 0. We now investigate a non-linear decision boundary.

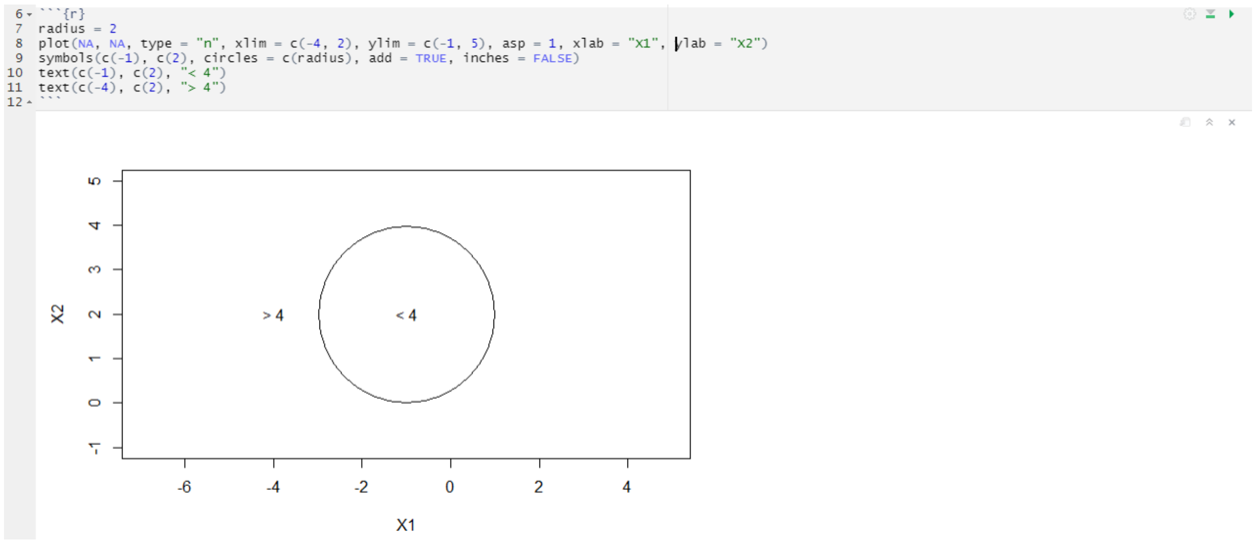
(a) Sketch the curve (1 + X1) 2 + (2 − X2) 2 = 4.

(1+X1)2+(2−X2)2=4(1+X1)2+(2−X2)2=4 is a circle with radius 2 and center (-1, 2).

~~~
```{r}
radius = 2
plot(NA, NA, type = "n", xlim = c(-4, 2), ylim = c(-1, 5), asp = 1, xlab = "X1", ylab = "X2")
symbols(c(-1), c(2), circles = c(radius), add = TRUE, inches = FALSE)
```
~~~



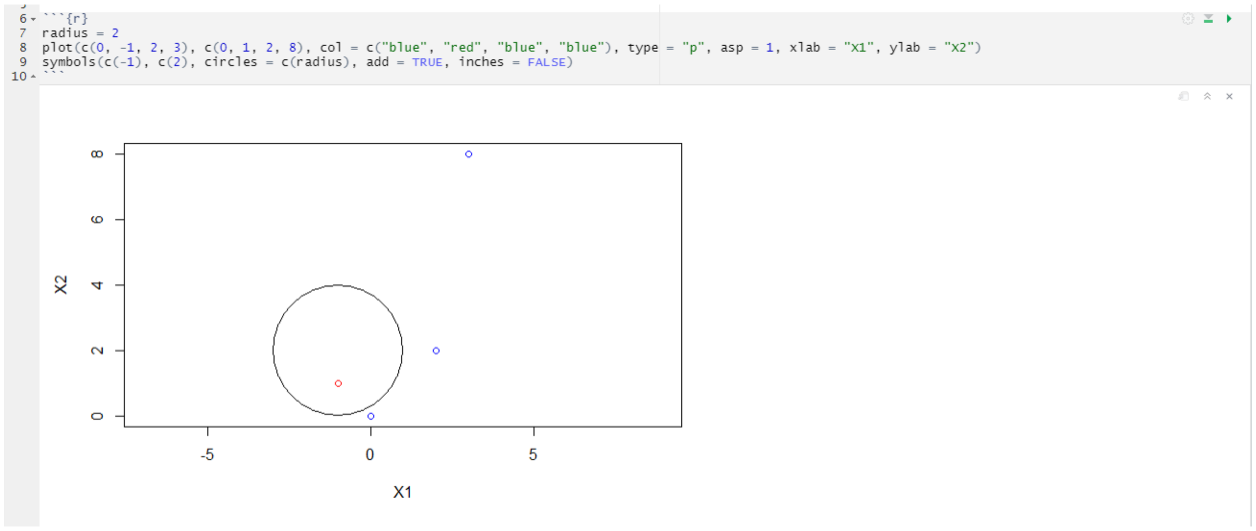
(b) On your sketch, indicate the set of points for which (1 + X1) 2 + (2 − X2) 2 > 4, as well as the set of points for which (1 + X1) 2 + (2 − X2) 2 ≤ 4.



```{r}
radius = 2
plot(NA, NA, type = "n", xlim = c(-4, 2), ylim = c(-1, 5), asp = 1, xlab = "X1", ylab = "X2")
symbols(c(-1), c(2), circles = c(radius), add = TRUE, inches = FALSE)
text(c(-1), c(2), "< 4")
text(c(-4), c(2), "> 4")
```

(c) Suppose that a classifier assigns an observation to the blue class if (1 + X1) 2 + (2 − X2) 2 > 4, and to the red class otherwise. To what class is the observation (0, 0) classified? (−1, 1)? (2, 2)? (3, 8)?

To restate the boundary, outside the circle is blue, inside and on is red.



```{r}
radius = 2
plot(c(0, -1, 2, 3), c(0, 1, 2, 8), col = c("blue", "red", "blue", "blue"), type = "p", asp = 1, xlab = "X1", ylab = "X2")
symbols(c(-1), c(2), circles = c(radius), add = TRUE, inches = FALSE)
```

(d) Argue that while the decision boundary in (c) is not linear in terms of X1 and X2, it is linear in terms of X1, X2 1 , X2, and X2 2 .

The decision boundary is a sum of quadratic terms when expanded.

$(1+X_1)_2+(2-X_2)_2>4$

$1+2X_1+X_{21}+4-4X_2+X_{22}>4$

$5+2X_1-4X_2+X_{21}+X_{22}>4$

R Notebook

5. We have seen that we can fit an SVM with a non-linear kernel in order to perform classification using a non-linear decision boundary. We will now see that we can also obtain a non-linear decision boundary by performing logistic regression using non-linear transformations of the features.

a. Generate a data set with n = 500 and p = 2, such that the observations belong to two classes with a quadratic decision boundary between them. For instance, you can do this as follows: > x1 <- runif (500) - 0.5 > x2 <- runif (500) - 0.5 > y <- 1 * (x1^2 - x2^2 > 0)

```
set.seed(421)
x1 = runif(500) - 0.5
x2 = runif(500) - 0.5
y = 1 * (x1^2 - x2^2 > 0)
```

b. Plot the observations, colored according to their class labels. Your plot should display X1 on the x-axis, and X2 on the yaxis.

```
plot(x1[y == 0], x2[y == 0], col = "red", xlab = "X1", ylab = "X2", pch =
"+")
points(x1[y == 1], x2[y == 1], col = "blue", pch = 4)
```

The plot clearly shows non-linear decision boundary.

c. Fit a logistic regression model to the data, using X1 and X2 as predictors.

```
lm.fit = glm(y ~ x1 + x2, family = binomial)
summary(lm.fit)

##
## Call:
## glm(formula = y ~ x1 + x2, family = binomial)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.278  -1.227   1.089   1.135   1.175
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.11999    0.08971   1.338    0.181
## x1          -0.16881    0.30854  -0.547    0.584
## x2          -0.08198    0.31476  -0.260    0.795
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 691.35  on 499  degrees of freedom
## Residual deviance: 690.99  on 497  degrees of freedom
## AIC: 696.99
```

```
##
## Number of Fisher Scoring iterations: 3
```

Both variables are insignificant for predicting y.

d. Apply this model to the training data in order to obtain a predicted class label for each training observation. Plot the observations, colored according to the predicted class labels. The decision boundary should be linear

```
data = data.frame(x1 = x1, x2 = x2, y = y)
lm.prob = predict(lm.fit, data, type = "response")
lm.pred = ifelse(lm.prob > 0.52, 1, 0)
data.pos = data[lm.pred == 1, ]
data.neg = data[lm.pred == 0, ]
plot(data.pos$x1, data.pos$x2, col = "blue", xlab = "X1", ylab = "X2", pch =
"+")
points(data.neg$x1, data.neg$x2, col = "red", pch = 4)
```

With the given model and a probability threshold of 0.5, all points are classified to single class and no decision boundary can be shown. Hence we shift the probability threshold to 0.52 to show a meaningful decision boundary. This boundary is linear as seen in the figure.

e. Now fit a logistic regression model to the data using non-linear functions of X1 and X2 as predictors (e.g. X2 1 , X1×X2, log(X2), and so forth).

We use squares, product interaction terms to fit the model.

```
lm.fit = glm(y ~ poly(x1, 2) + poly(x2, 2) + I(x1 * x2), data = data, family
= binomial)

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

f. Apply this model to the training data in order to obtain a predicted class label for each training observation. Plot the observations, colored according to the predicted class labels. The decision boundary should be obviously non-linear. If it is not, then repeat (a)-(e) until you come up with an example in which the predicted class labels are obviously non-linear.

```
lm.prob = predict(lm.fit, data, type = "response")
lm.pred = ifelse(lm.prob > 0.5, 1, 0)
data.pos = data[lm.pred == 1, ]
data.neg = data[lm.pred == 0, ]
plot(data.pos$x1, data.pos$x2, col = "blue", xlab = "X1", ylab = "X2", pch =
"+")
points(data.neg$x1, data.neg$x2, col = "red", pch = 4)
```

This non-linear decision boundary closely resembles the true decision boundary.

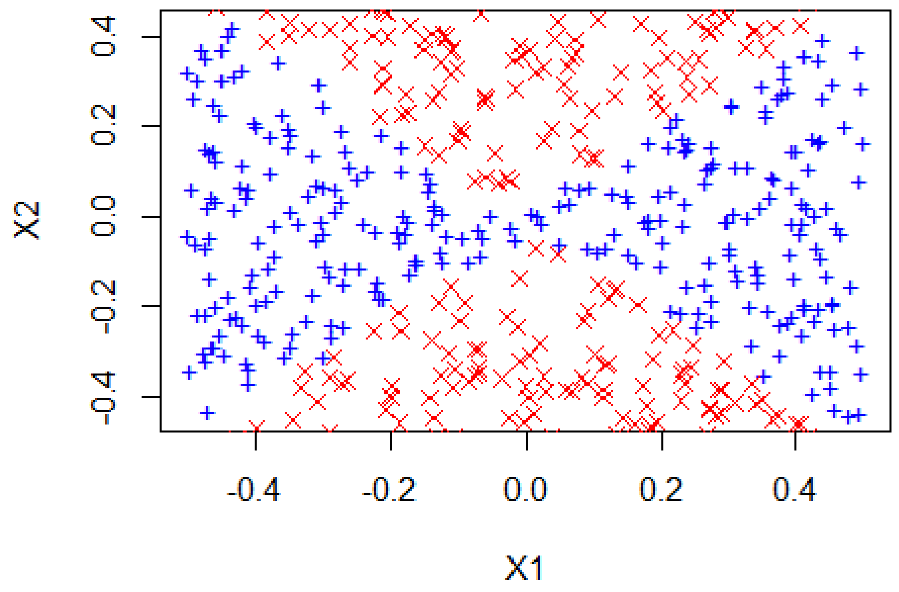
g. Fit a support vector classifier to the data with X1 and X2 as predictors. Obtain a class prediction for each training observation. Plot the observations, colored according to the predicted class labels.

```
library(e1071)

## Warning: package 'e1071' was built under R version 4.2.1

svm.fit = svm(as.factor(y) ~ x1 + x2, data, kernel = "linear", cost = 0.1)
svm.pred = predict(svm.fit, data)
data.pos = data[svm.pred == 1, ]
data.neg = data[svm.pred == 0, ]
plot(data.pos$x1, data.pos$x2, col = "blue", xlab = "X1", ylab = "X2", pch =
"+")
points(data.neg$x1, data.neg$x2, col = "red", pch = 4)
```

A linear kernel, even with low cost fails to find non-linear decision boundary and classifies all points to a single class.

h. Fit a SVM using a non-linear kernel to the data. Obtain a class prediction for each training observation. Plot the observations, colored according to the predicted class labels.

```
svm.fit = svm(as.factor(y) ~ x1 + x2, data, gamma = 1)
svm.pred = predict(svm.fit, data)
data.pos = data[svm.pred == 1, ]
data.neg = data[svm.pred == 0, ]
plot(data.pos$x1, data.pos$x2, col = "blue", xlab = "X1", ylab = "X2", pch =
"+")
points(data.neg$x1, data.neg$x2, col = "red", pch = 4)
```



Again, the non-linear decision boundary on predicted labels closely resembles the true decision boundary.

i. Comment on your results.

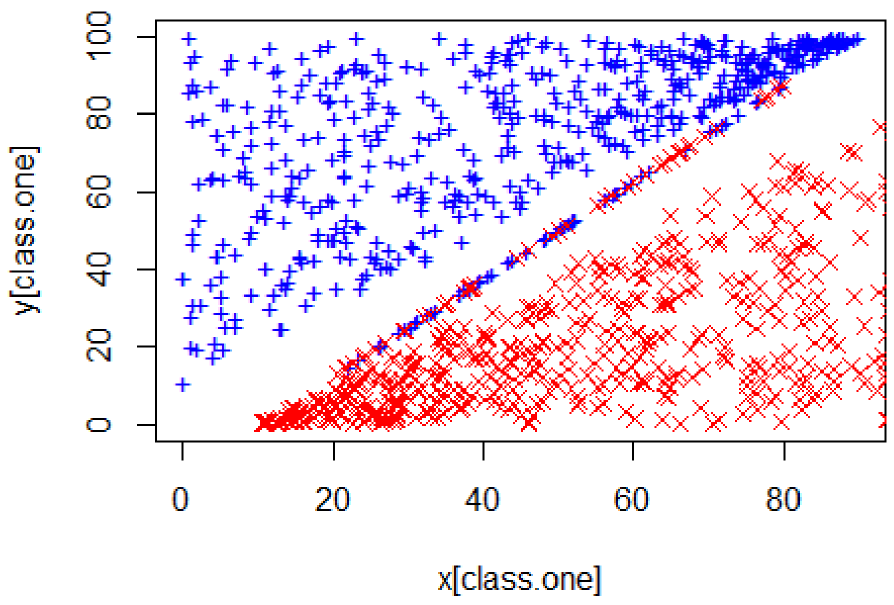
This experiment demonstrates the effectiveness of SVMs with non-linear kernels for locating non-linear boundaries. SVMs using linear kernels and logistic regression with no interactions both fall short in locating the decision border. Logistic regression appears to have the same power as radial-basis kernels when interaction factors are included. However, choosing the proper interaction terms requires some manual work and fine adjustment. With a lot of features, this endeavor may be impossible. On the other hand, radial basis kernels just need to have the gamma value tuned, which can be done quickly via cross-validation.

6. At the end of Section 9.6.1, it is claimed that in the case of data that is just barely linearly separable, a support vector classifier with a small value of cost that misclassifies a couple of training observations may perform better on test data than one with a huge value of cost that does not misclassify any training observations. You will now investigate this claim.

a. Generate two-class data with p = 2 in such a way that the classes are just barely linearly separable.

We randomly generate 1000 points and scatter them across line x=y with wide margin. We also create noisy points along the line 5x−4y−50=0. These points make the classes barely separable and also shift the maximum margin classifier.

```
set.seed(3154)
# Class one
x.one = runif(500, 0, 90)
y.one = runif(500, x.one + 10, 100)
x.one.noise = runif(50, 20, 80)
y.one.noise = 5/4 * (x.one.noise - 10) + 0.1

# Class zero
x.zero = runif(500, 10, 100)
y.zero = runif(500, 0, x.zero - 10)
x.zero.noise = runif(50, 20, 80)
y.zero.noise = 5/4 * (x.zero.noise - 10) - 0.1

# Combine all
class.one = seq(1, 550)
x = c(x.one, x.one.noise, x.zero, x.zero.noise)
y = c(y.one, y.one.noise, y.zero, y.zero.noise)

plot(x[class.one], y[class.one], col = "blue", pch = "+", ylim = c(0, 100))
points(x[-class.one], y[-class.one], col = "red", pch = 4)
```



The plot shows that classes are barely separable. The noisy points create a fictitious boundary 5x−4y−50=0.

b. Compute the cross-validation error rates for support vector classifiers with a range of cost values. How many training errors are misclassified for each value of cost considered, and how does this relate to the cross-validation errors obtained?

We create a z variable according to classes.

```
library(e1071)

set.seed(555)
z = rep(0, 1100)
z[class.one] = 1
data = data.frame(x = x, y = y, z = z)
tune.out = tune(svm, as.factor(z) ~ ., data = data, kernel = "linear", ranges
= list(cost = c(0.01,
    0.1, 1, 5, 10, 100, 1000, 10000)))
summary(tune.out)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost
##  10000
##
```

```
## - best performance: 0
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-02 0.05818182 0.01926091
## 2 1e-01 0.04727273 0.01648663
## 3 1e+00 0.04636364 0.01738137
## 4 5e+00 0.05000000 0.01975516
## 5 1e+01 0.04909091 0.01973190
## 6 1e+02 0.04818182 0.02012359
## 7 1e+03 0.04181818 0.03005963
## 8 1e+04 0.00000000 0.00000000
```

```r
data.frame(cost = tune.out$performances$cost, misclass =
tune.out$performances$error *
    1100)
```

```
##    cost misclass
## 1 1e-02       64
## 2 1e-01       52
## 3 1e+00       51
## 4 5e+00       55
## 5 1e+01       54
## 6 1e+02       53
## 7 1e+03       46
## 8 1e+04        0
```

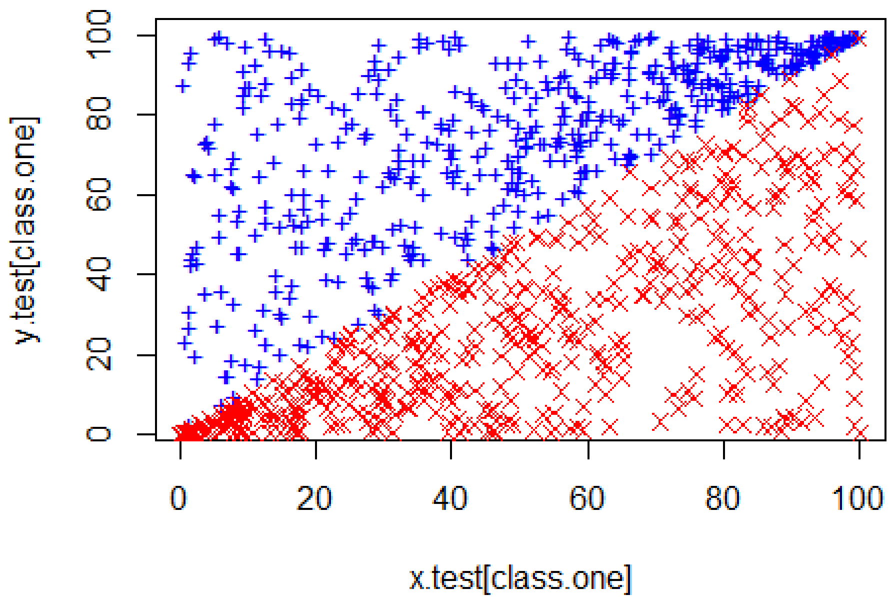
The table above shows train-misclassification error for all costs. A cost of 10000 seems to classify all points correctly. This also corresponds to a cross-validation error of 0.

c. Generate an appropriate test data set, and compute the test errors corresponding to each of the values of cost considered. Which value of cost leads to the fewest test errors, and how does this compare to the values of cost that yield the fewest training errors and the fewest cross-validation errors?

We now generate a random test-set of same size. This test-set satisfies the true decision boundary x=y.

```r
set.seed(1111)
x.test = runif(1000, 0, 100)
class.one = sample(1000, 500)
y.test = rep(NA, 1000)
# Set y > x for class.one
for (i in class.one) {
    y.test[i] = runif(1, x.test[i], 100)
}
# set y < x for class.zero
for (i in setdiff(1:1000, class.one)) {
    y.test[i] = runif(1, 0, x.test[i])
}
```

```
plot(x.test[class.one], y.test[class.one], col = "blue", pch = "+")
points(x.test[-class.one], y.test[-class.one], col = "red", pch = 4)
```



We now make same predictions using all linear svms with all costs used in previous part.

```
set.seed(30012)
z.test = rep(0, 1000)
z.test[class.one] = 1
all.costs = c(0.01, 0.1, 1, 5, 10, 100, 1000, 10000)
test.errors = rep(NA, 8)
data.test = data.frame(x = x.test, y = y.test, z = z.test)
for (i in 1:length(all.costs)) {
    svm.fit = svm(as.factor(z) ~ ., data = data, kernel = "linear", cost =
all.costs[i])
    svm.predict = predict(svm.fit, data.test)
    test.errors[i] = sum(svm.predict != data.test$z)
}

data.frame(cost = all.costs, `test misclass` = test.errors)

##    cost test.misclass
## 1 1e-02            51
## 2 1e-01            14
## 3 1e+00             6
## 4 5e+00             0
## 5 1e+01             0
## 6 1e+02           183
## 7 1e+03           205
## 8 1e+04           205
```

cost=10 seems to be performing better on test data, making the least number of classification errors. This is much smaller than optimal value of 10000 for training data.

d. Discuss your results.

We observe overfitting for linear kernel once more. The train data is overfit because a huge cost tries to fit accurately classify noisy-points. However, a modest cost works better on test data and makes a few mistakes on the noisy test points.