Homework2_sol

Lei Shi

12/6/2021

R Markdown

Question 1

(a) See Figure 1.

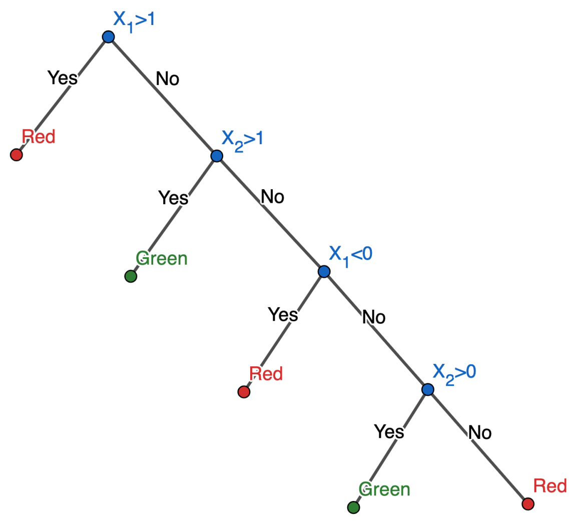


Figure 1: Tree

(b) See Figure 2.

(c) (2,1): $X_2 < -1.55? \xrightarrow{No} X_2 < 1.25? \xrightarrow{Yes} X_1 < 2.25? \xrightarrow{Yes} X_1 < -1.8? \xrightarrow{No}$ Classified to -1.

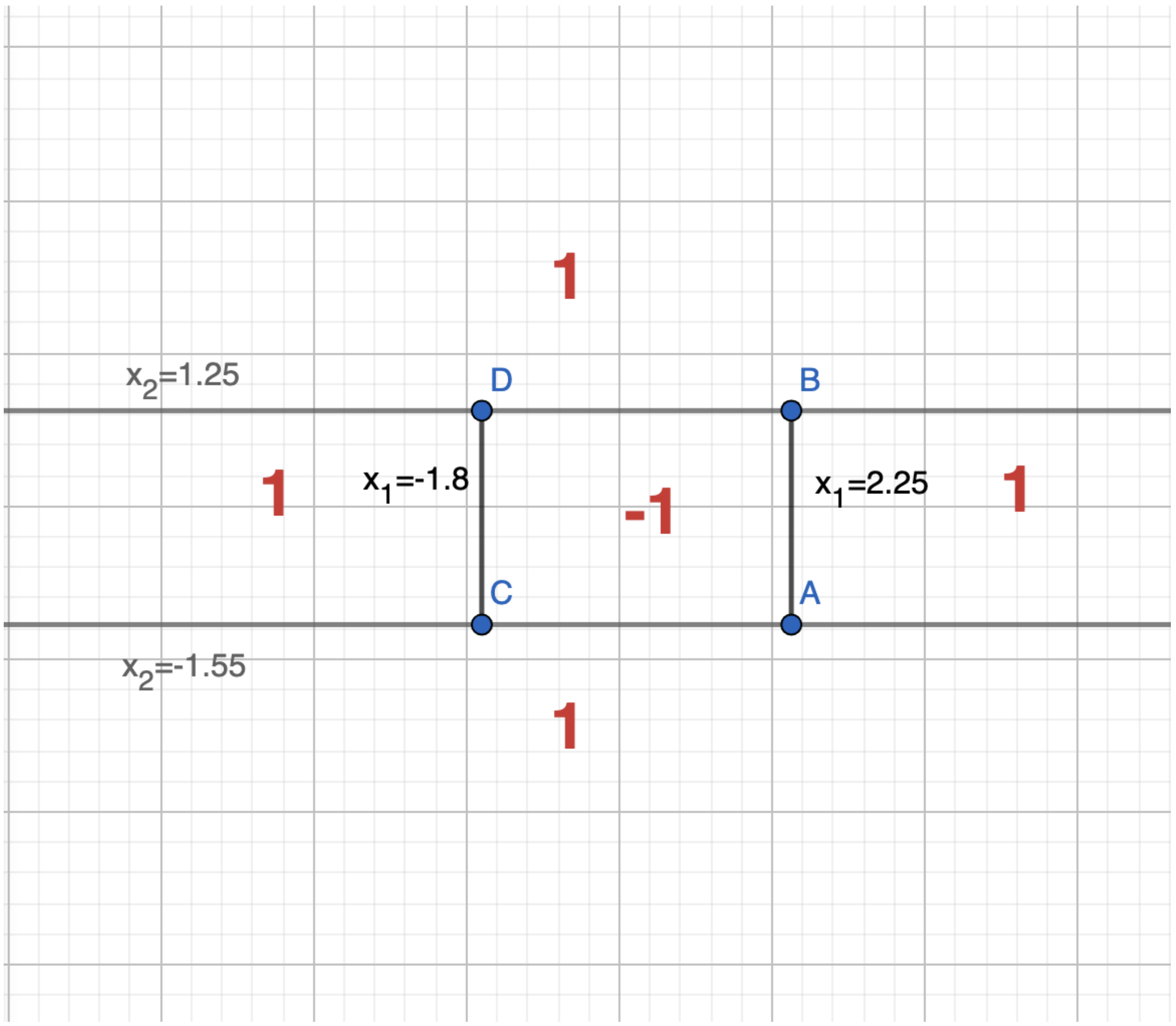


Figure 2: Tree

Question 2

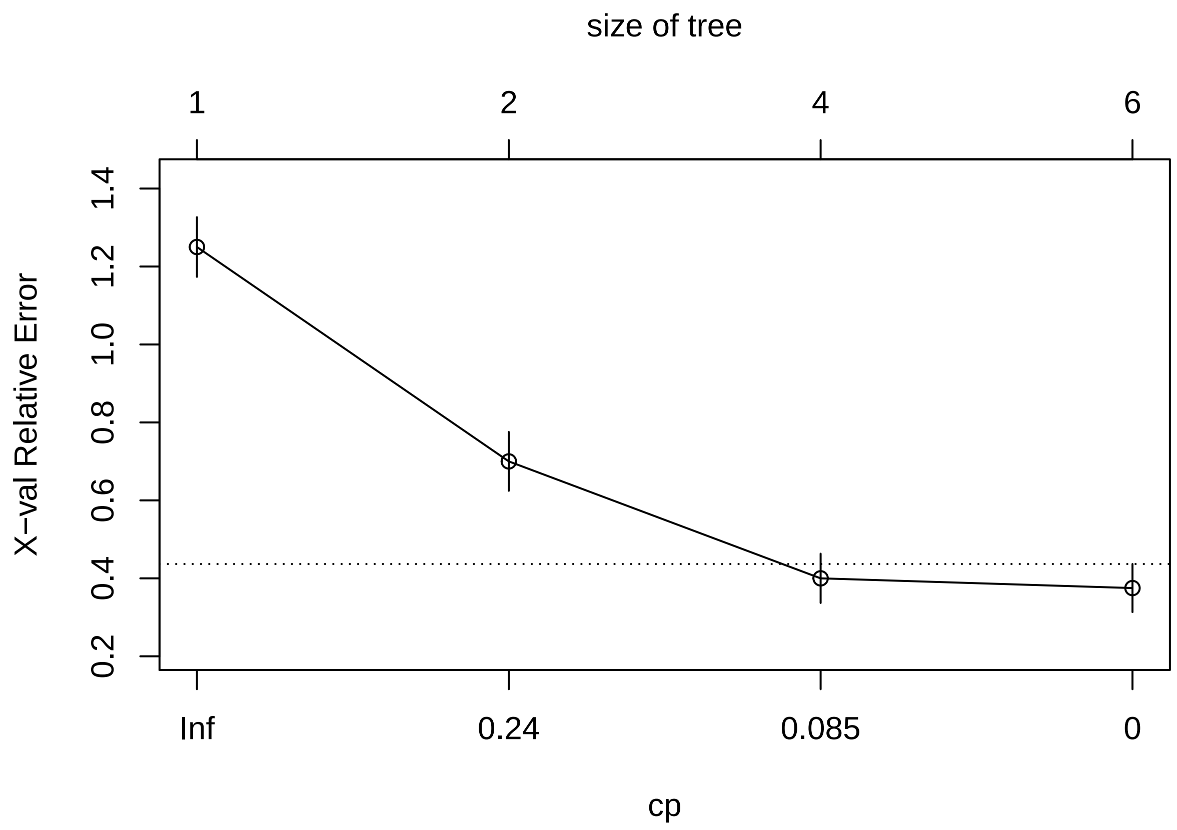
```
data <- MASS::crabs
set.seed(6789)

sex.levels <- levels(data$sex)
sp.levels <- levels(data$sp)

train <- data.frame()
test <- data.frame()

for (sex in sex.levels){
for (sp in sp.levels){
  stratum <- data %>% filter(sex == !!sex & sp == !!sp)
  train.index <- sample(seq_len(nrow(stratum)), size = floor(0.8*nrow(stratum)))
  train <- rbind(train, stratum[train.index, ])
  test <- rbind(test, stratum[-train.index, ])
}
}

train <- subset(train, select = -index)
test <- subset(test, select = -index)
```

(a)

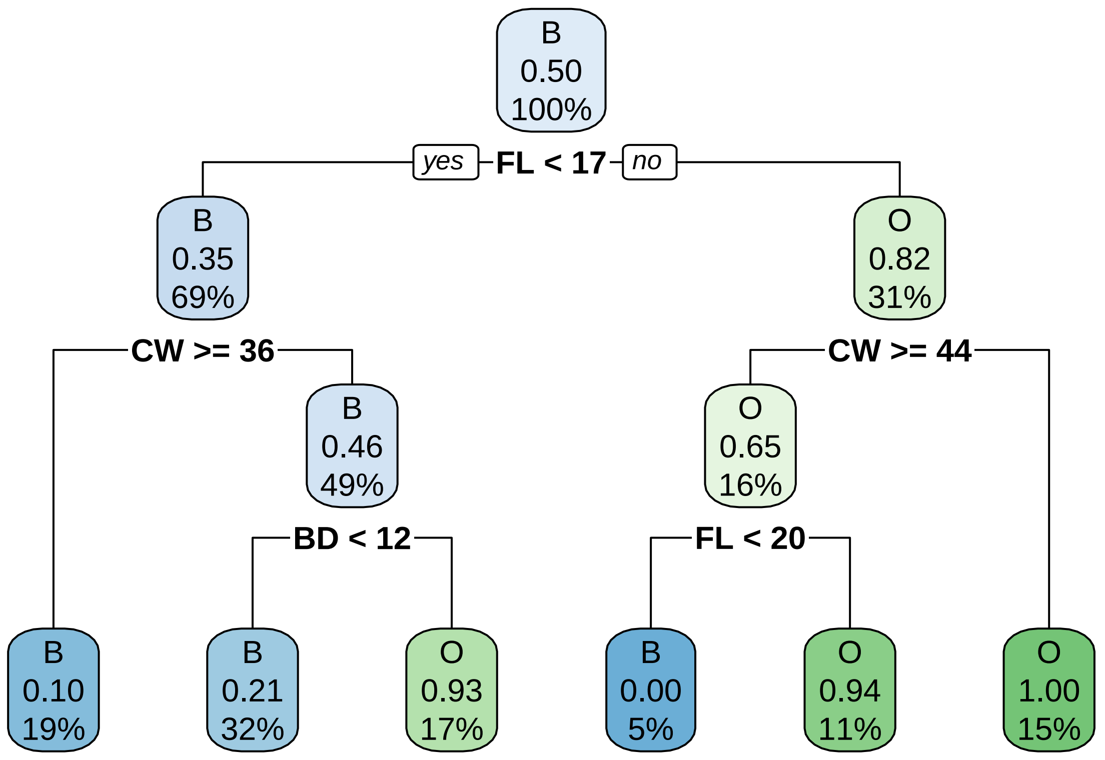
```
fit <- rpart(sp ~ ., method="class", data=train,
             control = rpart.control(minsplit =1,
                                     minbucket=1,
                                     xval = 10,
                                     cp=0,
                                     maxdepth = 3))
printcp(fit)
```

```
##
## Classification tree:
## rpart(formula = sp ~ ., data = train, method = "class", control = rpart.control(minsplit = 1,
##     minbucket = 1, xval = 10, cp = 0, maxdepth = 3))
##
## Variables actually used in tree construction:
## [1] BD CW FL
##
## Root node error: 80/160 = 0.5
##
## n= 160
##
##        CP nsplit rel error xerror     xstd
## 1 0.40000      0    1.0000  1.250 0.076547
## 2 0.14375      1    0.6000  0.700 0.075416
## 3 0.05000      3    0.3125  0.400 0.063246
## 4 0.00000      5    0.2125  0.375 0.061714
```

```
plotcp(fit)
```



```
rpart.plot(fit)
```



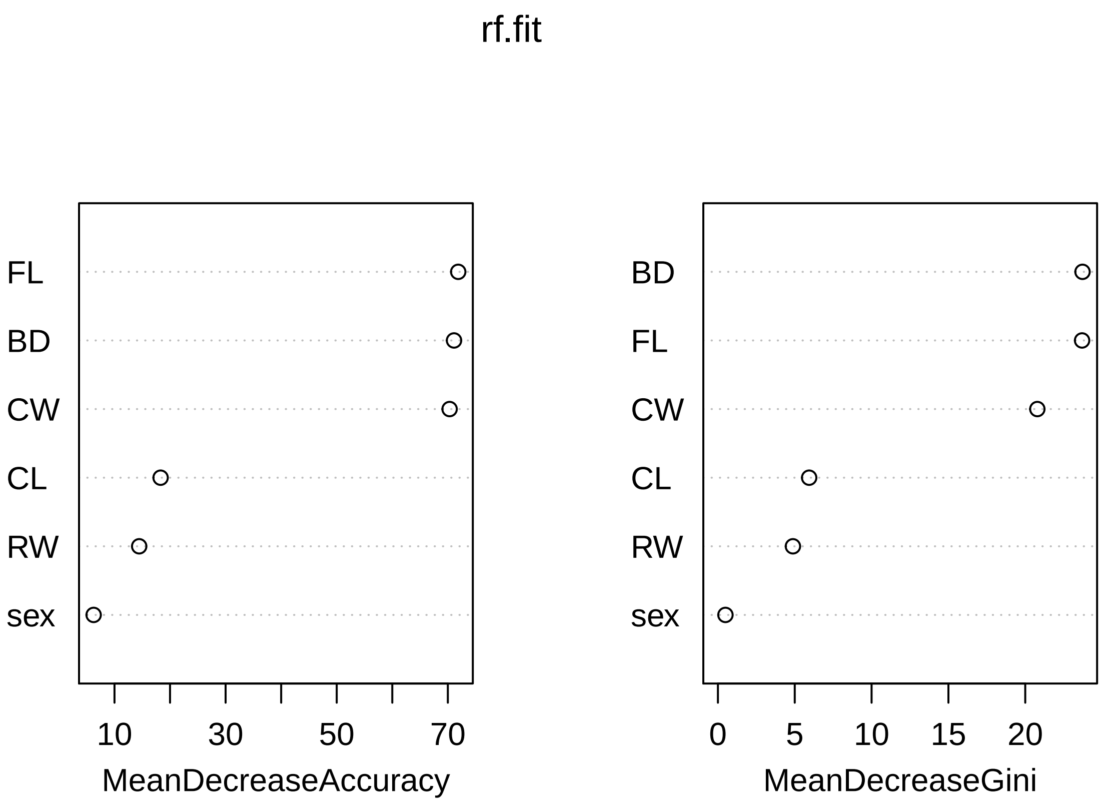
```
# Alternative method without "rpart.plot"
# plot(fit, uniform=TRUE, main="Classification Tree for Crabs")
# text(fit, use.n=TRUE, all=TRUE, cex=.8)

# training error and test error

prediction <- predict(fit, newdata = train, type = "class")
training.error <- sum(prediction!=train$sp)/length(prediction)

prediction <- predict(fit, newdata = test, type = "class")
test.error <- sum(prediction!=test$sp)/length(prediction)

data.frame(training.error, test.error)
```

```
##   training.error test.error
## 1        0.10625        0.1
```

The variables used in the tree: FL, CW and BD. The training error and testing error are 10.6% and 10.0% respectively.

(b)

```
set.seed(6789)
rf.fit <- randomForest(
  sp ~ sex + FL + RW + CL + CW + BD,
  data=train,
  ntree=1000,
  mtry=5,
  importance = T
  )

varImpPlot(rf.fit)
```



```
prediction <- predict(rf.fit, newdata = train, type = "class")
training.error <- sum(prediction!=train$sp)/length(prediction)

prediction <- predict(rf.fit, newdata = test, type = "class")
test.error <- sum(prediction!=test$sp)/length(prediction)

data.frame(training.error, test.error)
```

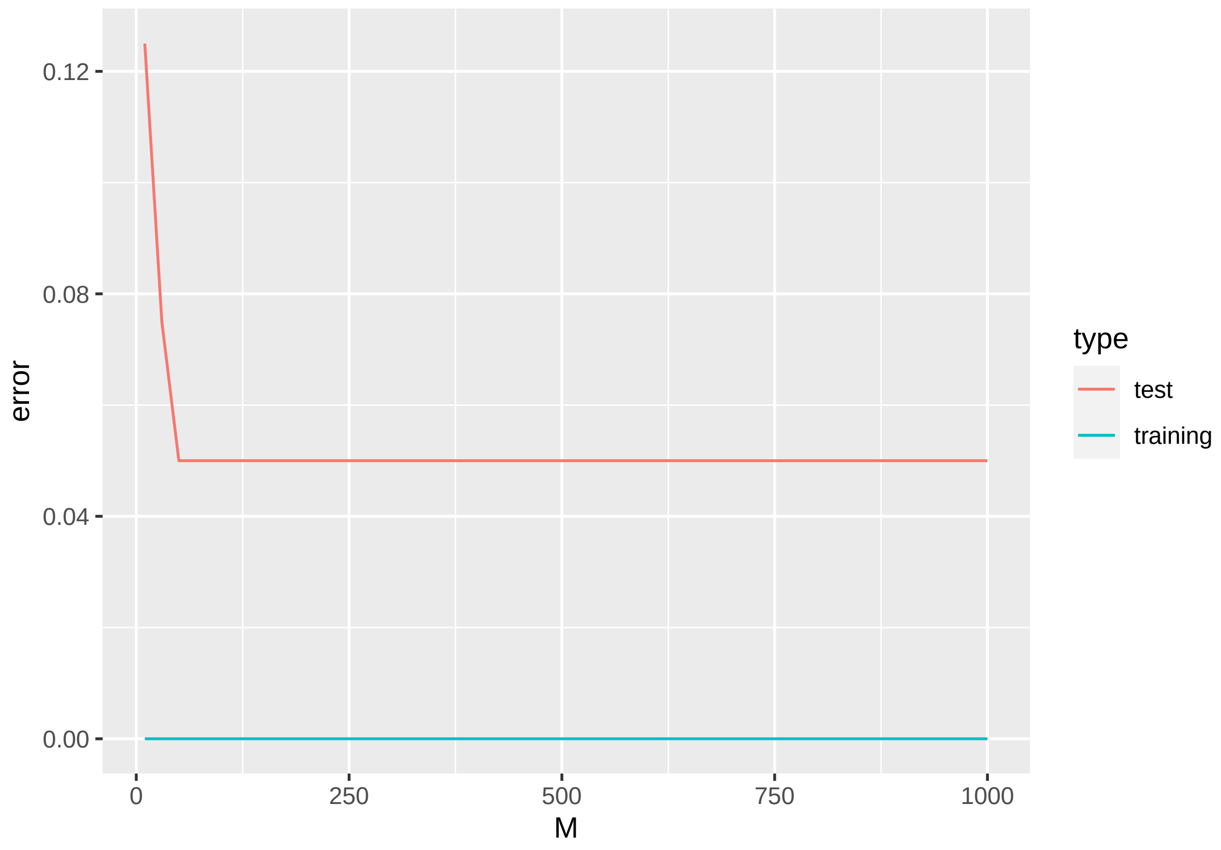
```
##   training.error test.error
## 1              0      0.125
```

(c)

```
set.seed(6789)
train.error <- c()
test.error <- c()

M_candidate <- c(10, 20, 30, 50, 70, 100, 200, 300, 500, 700, 1000)

for (M in M_candidate){
  fit <- boosting(sp ~ .,data=train, mfinal=M)

  # Compute training error
  pred.train <- predict(fit, newdata = train, type = "class")
  train.error <- c(train.error, sum(pred.train$class != train$sp)/length(train$sp))

  # Compute testing error
  pred.test <- predict(fit, newdata = test, type = "class")
  test.error <- c(test.error, sum(pred.test$class != test$sp)/length(test$sp))

  cat("Running M=", M, "\n")
}
```

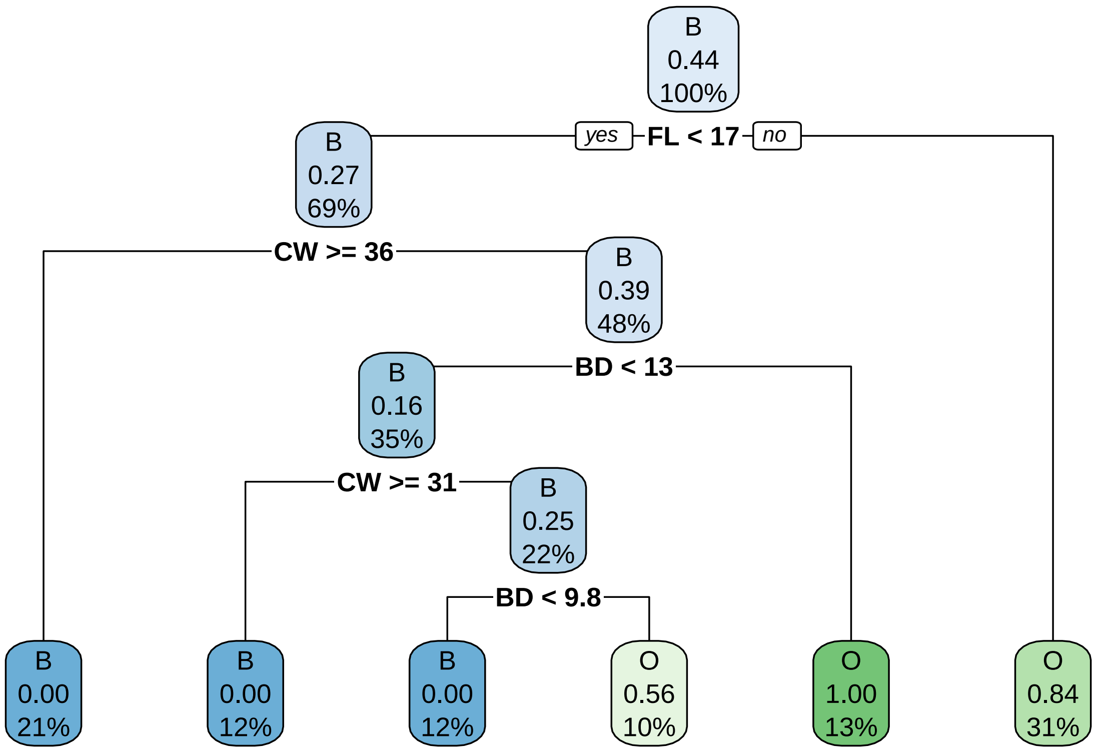
```
## Running M= 10
## Running M= 20
## Running M= 30
## Running M= 50
## Running M= 70
## Running M= 100
## Running M= 200
## Running M= 300
## Running M= 500
## Running M= 700
## Running M= 1000
```

```
error.df <- data.frame(M = rep(M_candidate, 2),
                       error = c(train.error, test.error),
                       type = rep(c("training","test"), each=length(M_candidate)))
ggplot(error.df, aes(x=M, y=error, col=type)) + geom_line()
```



```
# with M = 30
fit <- boosting(sp ~ .,data=train, mfinal = 30)
rpart.plot(fit$trees[[1]])
```

```
## Warning: Cannot retrieve the data used to build the model (so cannot determine roundint and is.binar
## To silence this warning:
##     Call rpart.plot with roundint=FALSE,
##     or rebuild the rpart model with model=TRUE.
```



```
data.frame(train.error[3], test.error[3])
```

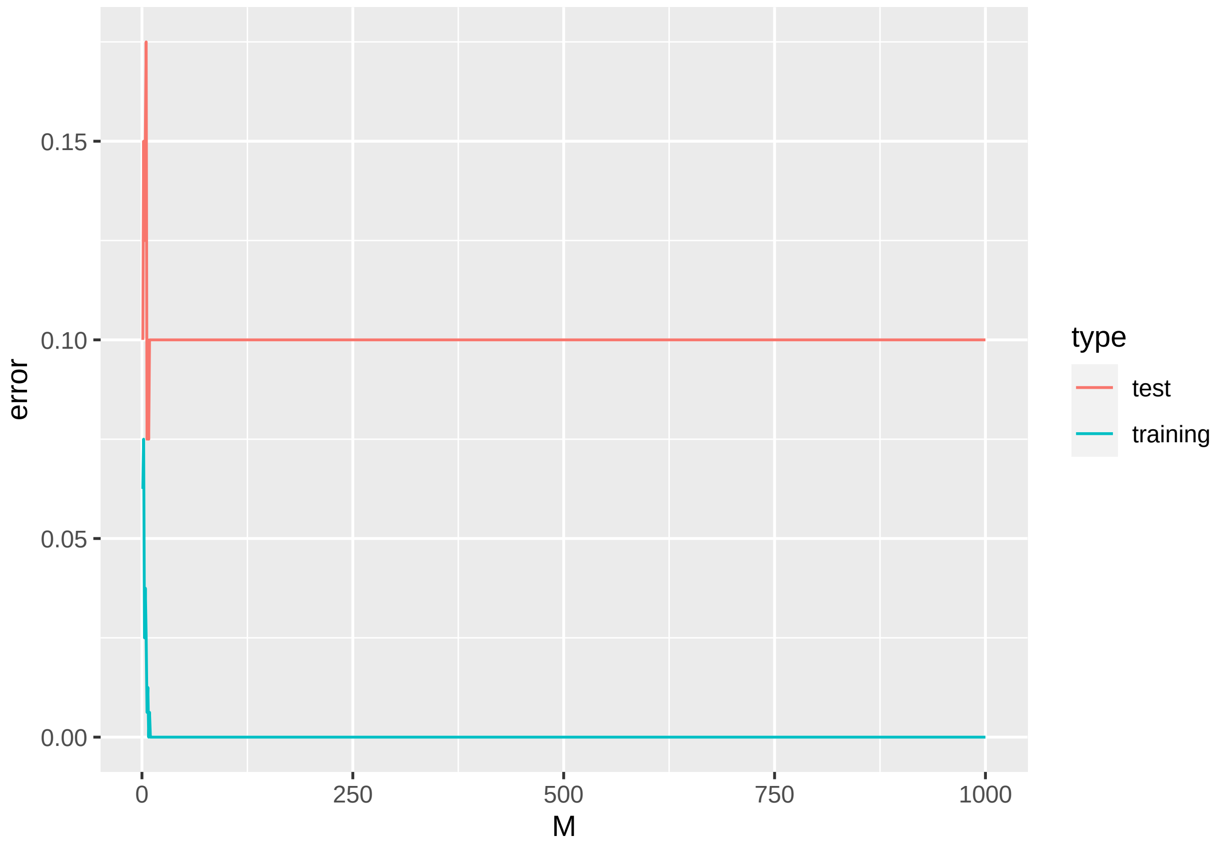
```
##   train.error.3. test.error.3.
## 1              0         0.075
```

Alternatively we can use another package:

```
set.seed(6789)
train.error <- c()
test.error <- c()

M_candidate <- c(1:10, 20, 30, 50, 70, 100, 200, 300, 500, 700, 1000)

for (M in M_candidate){
  fit <- adaboost(sp ~ ., data=train, nIter=M)

  # Compute training error
  pred.train <- predict(fit, newdata = train, type = "class")
  train.error <- c(train.error, sum(pred.train$class != train$sp)/length(train$sp))

  # Compute testing error
  pred.test <- predict(fit, newdata = test, type = "class")
  test.error <- c(test.error, sum(pred.test$class != test$sp)/length(test$sp))

  cat("Running M =", M, "\n")
}
```

```
## Running M = 1
## Running M = 2
## Running M = 3
## Running M = 4
## Running M = 5
## Running M = 6
## Running M = 7
## Running M = 8
## Running M = 9
## Running M = 10
## Running M = 20
## Running M = 30
## Running M = 50
## Running M = 70
## Running M = 100
## Running M = 200
## Running M = 300
## Running M = 500
## Running M = 700
## Running M = 1000
```

```
error.df <- data.frame(M = rep(M_candidate, 2),
                       error = c(train.error, test.error),
                       type = rep(c("training","test"), each=length(M_candidate)))
ggplot(error.df, aes(x=M, y=error, col=type)) + geom_line()
```



Choose M = 10(or 20, 30, 50) works well in this problem, since larger M won't lead to significant increase in the training and testing accuracy but will render the algorithm more time-consuming.

(d) A table summarizing the accuracy:

Points	Training	Testing
tree	0.10625	0.1
rF	0	0.125
adaboost	0	0.075

Based on our results, we can see that Adaboost performs the best, with a testing error of 7.5% as opposed to 10.0% with the Classification tree and 12.5% with the Random Forest. In terms of variable important, the same variables are considered the most important by all methods in classifying the crab species.: FL, BD and CW.

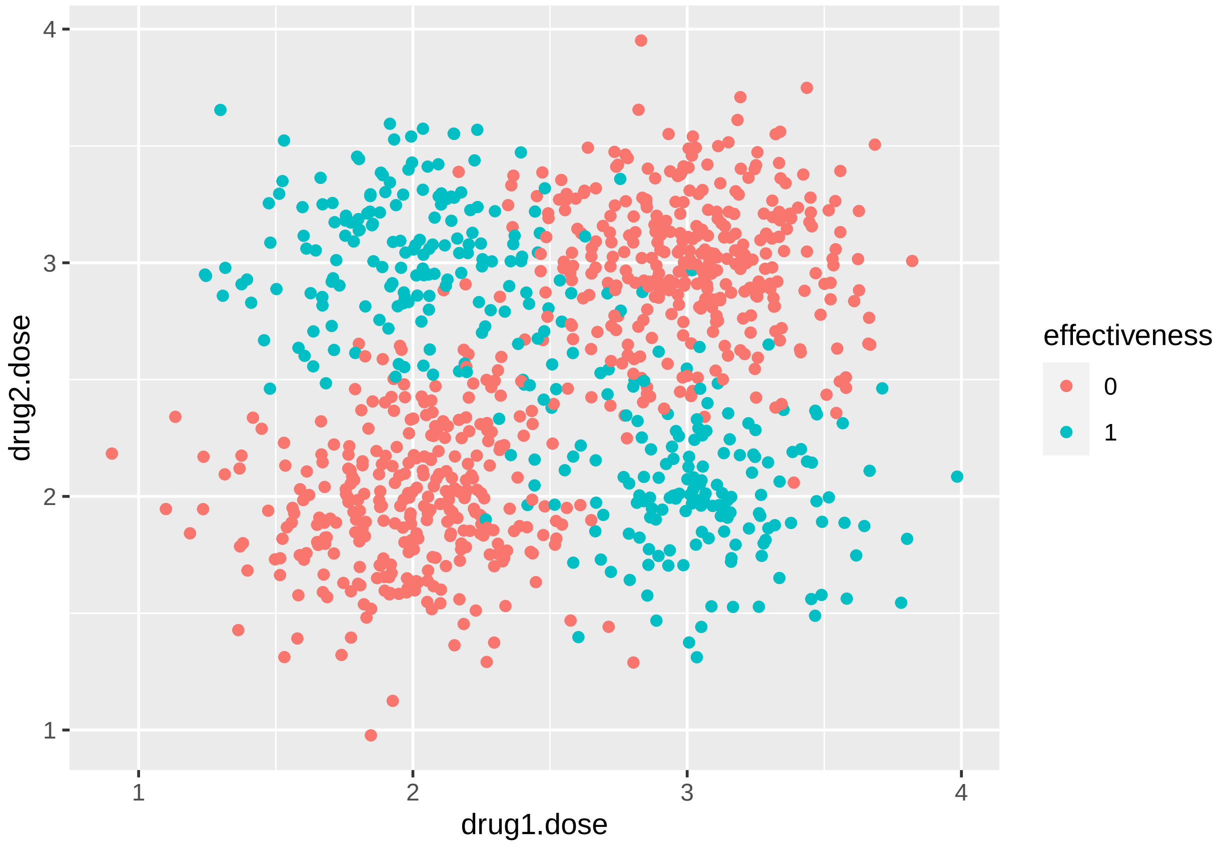
Question 3

Data importing and preprocessing

```
load("dataHW2.Rda")
data <- drug.effectiveness %>% rename(effectiveness = effectivness)
data$effectiveness <- as.factor(data$effectiveness)

train_index <- sample(nrow(data), size = floor(0.8*nrow(data)))
train <- data[train_index, ]
test <- data[-train_index, ]
```

Simple EDA:

```
ggplot(data, aes(x = drug1.dose, y = drug2.dose, color = effectiveness)) +
geom_point()
```



Logistic regression

```
fit.logit <- glm(effectiveness~., data = train, family = "binomial")
summary(fit.logit)
```

```
##
## Call:
## glm(formula = effectiveness ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.9643  -0.9079  -0.8842   1.4243   1.5935
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.85972    0.41364  -2.078   0.0377 *
## drug1.dose  -0.09043    0.13030  -0.694   0.4877
## drug2.dose   0.15679    0.13294   1.179   0.2382
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1018.9  on 799  degrees of freedom
## Residual deviance: 1017.3  on 797  degrees of freedom
## AIC: 1023.3
##
## Number of Fisher Scoring iterations: 4
```

```
# Compute training error
pred.train <- factor(as.numeric(predict(fit.logit, newdata = train) > 0.5))
levels(pred.train) <- c("1", "0")
train.error <- sum(pred.train != train$effectiveness)/length(train$effectiveness)

# Compute testing error
pred.test <- factor(as.numeric(predict(fit.logit, newdata = test) > 0.5))
levels(pred.test) <- c("1", "0")
test.error <- sum(pred.test != test$effectiveness)/length(test$effectiveness)

data.frame(train.error, test.error)
```

```
##   train.error test.error
## 1     0.66625      0.635
```

SVM with Gaussian Kernel

```
fit.svm <- svm(effectiveness ~ ., data = train)
summary(fit.svm)
```

```
##
## Call:
## svm(formula = effectiveness ~ ., data = train)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  201
##
##  ( 101 100 )
##
##
## Number of Classes:  2
##
## Levels:
```

```
##  0 1
```

```
# Compute training error
pred.train <- predict(fit.svm, newdata = train, type = "class")
train.error <- sum(pred.train != train$effectiveness)/length(train$effectiveness)

# Compute testing error
pred.test <- predict(fit.svm, newdata = test, type = "class")
test.error <- sum(pred.test != test$effectiveness)/length(test$effectiveness)

data.frame(train.error, test.error)
```

```
##   train.error test.error
## 1       0.085      0.095
```

Classification Tree

```
fit.tree <- rpart(effectiveness ~ ., method="class", data = train,
             control = rpart.control(minsplit =1,
                                     minbucket=1,
                                     xval = 10,
                                     cp=0,
                                     maxdepth = 3))

# Compute training error
pred.train <- predict(fit.tree, newdata = train, type = "class")
train.error <- sum(pred.train != train$effectiveness)/length(train$effectiveness)

# Compute testing error
pred.test <- predict(fit.tree, newdata = test, type = "class")
test.error <- sum(pred.test != test$effectiveness)/length(test$effectiveness)

data.frame(train.error, test.error)
```

```
##   train.error test.error
## 1     0.29625      0.355
```

Random Forest

```
fit.rf <- randomForest(
  effectiveness ~ .,
  data = train,
  ntree = 1000,
  mtry = 2,
  importance = T
  )
print(fit.rf)
```

```
##
## Call:
##  randomForest(formula = effectiveness ~ ., data = train, ntree = 1000,      mtry = 2, importance = T
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 9.38%
## Confusion matrix:
```

```
##     0   1 class.error
## 0 498  35  0.06566604
## 1  40 227  0.14981273
```

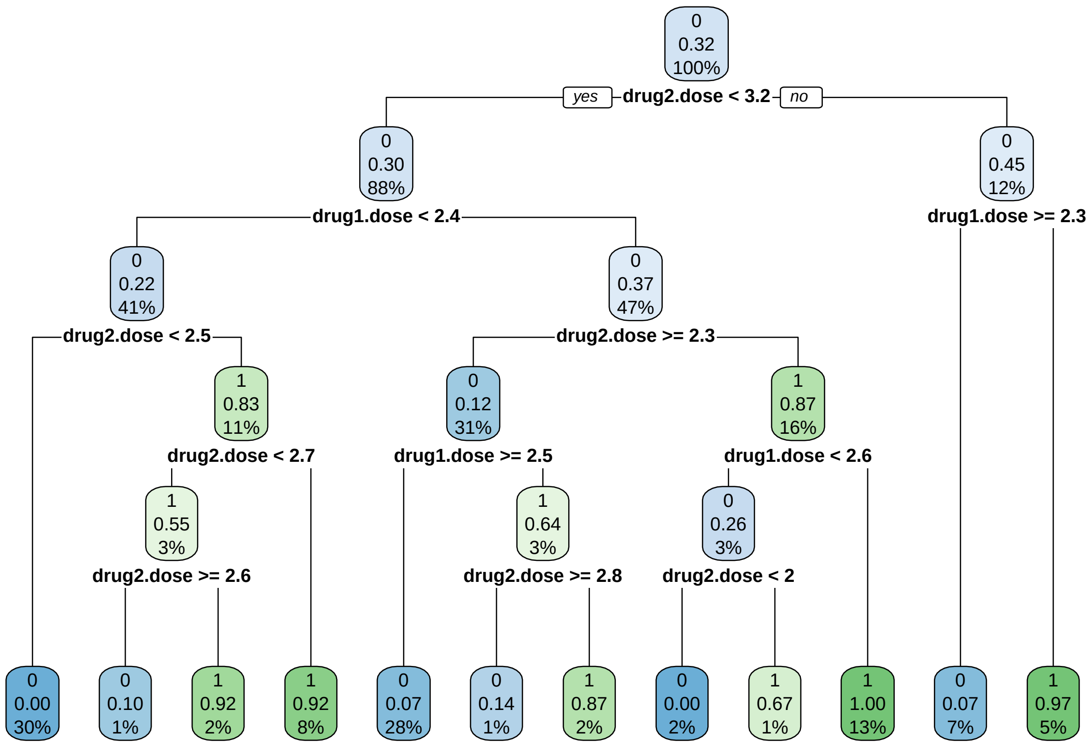
```
# Compute training error
pred.train <- predict(fit.rf, newdata = train, type = "class")
train.error <- sum(pred.train != train$effectiveness)/length(train$effectiveness)

# Compute testing error
pred.test <- predict(fit.rf, newdata = test, type = "class")
test.error <- sum(pred.test != test$effectiveness)/length(test$effectiveness)

data.frame(train.error, test.error)
```

```
##   train.error test.error
## 1           0      0.095
```

Adaboost

```
fit.boost <- boosting(effectiveness ~ ., data=train, mfinal = 30)
rpart.plot(fit.boost$trees[[1]])
```

```
## Warning: Cannot retrieve the data used to build the model (so cannot determine roundint and is.binar
## To silence this warning:
##     Call rpart.plot with roundint=FALSE,
##     or rebuild the rpart model with model=TRUE.
```



```
# Compute training error
pred.train <- predict(fit.boost, newdata = train, type = "class")
train.error <- sum(pred.train$class != train$effectiveness)/length(train$effectiveness)

# Compute testing error
pred.test <- predict(fit.boost, newdata = test, type = "class")
```

```r
test.error <- sum(pred.test$class != test$effectiveness)/length(test$effectiveness)

data.frame(train.error, test.error)
```

```
##   train.error test.error
## 1           0        0.1
```

For this classification problem, logistic regression is based on a misspecified model, so we don't expect a high prediction accuracy. The other algorithms(svm with gaussian kernel and tree type methods) fit with the structure of the data and demonstrate high prediction accuracy.

We are not grading based on accuracy though so take it easy!