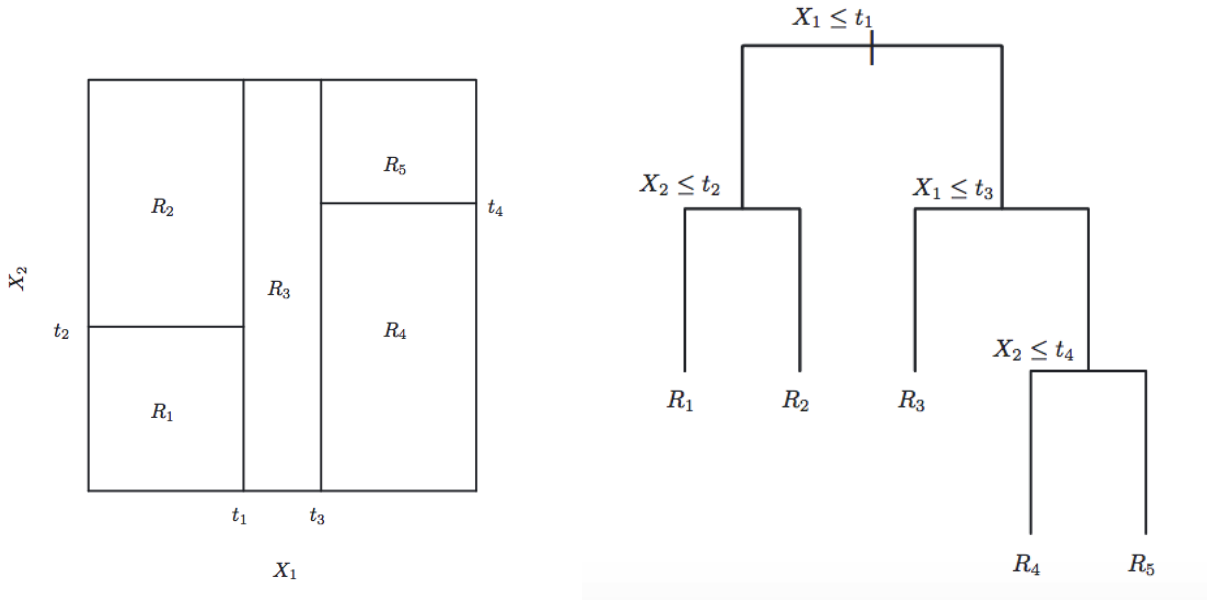
415 HW9 solution

April Cho

Problem 1

The example shows a partition of a two-dimensional feature space with 5 regions. The correct figure should be on (0,1) range on X1 and X2 axis.



Problem 2

a)

```
library(MASS)
```

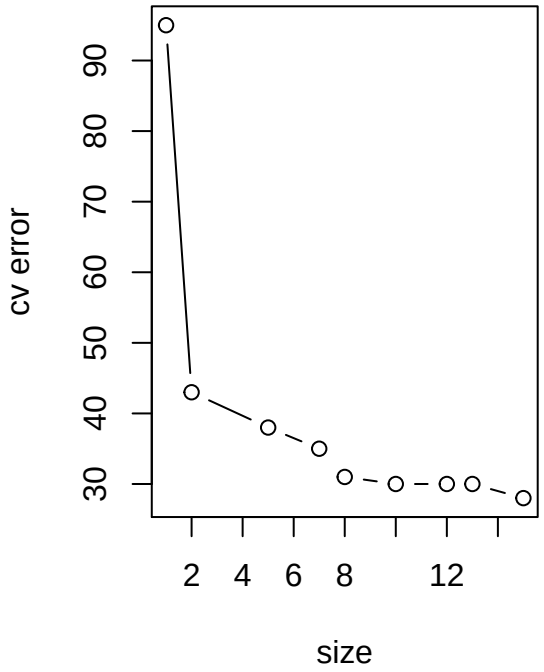
```
## Warning: package 'MASS' was built under R version 3.2.5
```

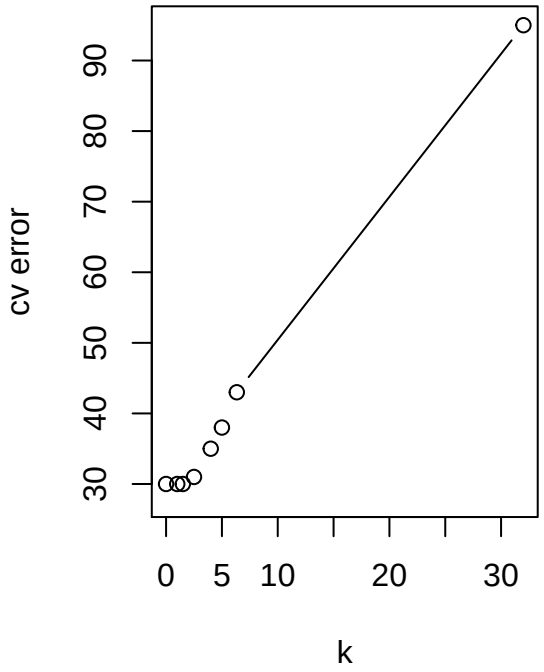
```
attach(crabs)
set.seed(45678)
blueMale = which(sp == "B" & sex == "M")
orangeMale = which(sp == "O" & sex == "M")
blueFemale = which(sp == "B" & sex == "F")
orangeFemale = which(sp == "O" & sex == "F")
train_id = c(sample(blueMale, size = trunc(0.80 * length(blueMale))),
          sample(orangeMale, size = trunc(0.80 * length(orangeMale))),
          sample(blueFemale, size = trunc(0.80 * length(blueFemale))),
          sample(orangeFemale, size = trunc(0.80 * length(orangeFemale))))
crabs_train = crabs[train_id, ]
crabs_test = crabs[-train_id,]
```

b)

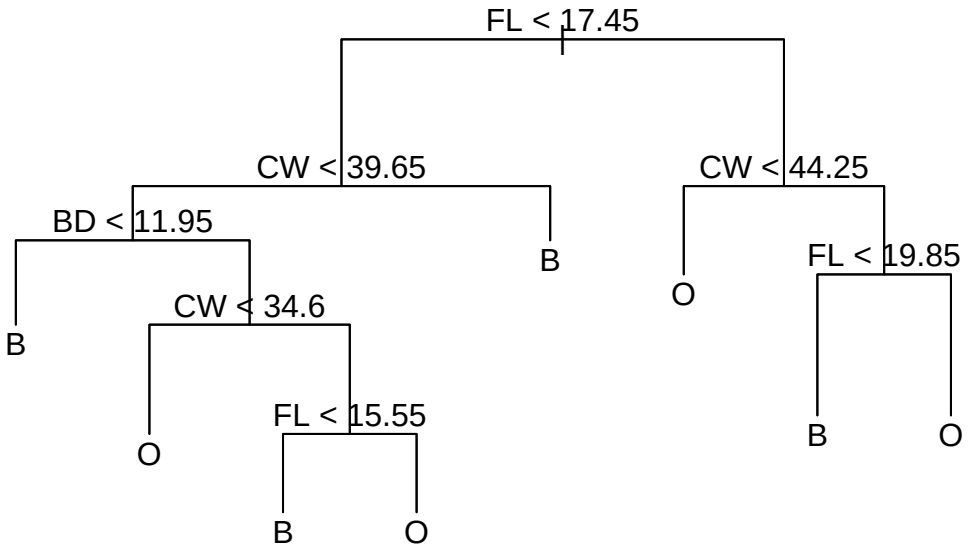
We exclude 'index' variable from the model since it is not one of the five numerical measurements or sex. Based on cross validation result, we choose 8 as the tree size since we are asked to choose a tree with no more than 8 splits. The variables used by tree is FL, CW, and BD.

```
library(tree)
tree.crabs=tree(sp~.-index,crabs,subset=train_id)
cv.crabs=cv.tree(tree.crabs,FUN=prune.misclass) #size 8 chosen by CV
par(mfrow=c(1,2))
plot(cv.crabs$size,cv.crabs$dev,ylab="cv error", xlab="size",type="b")
plot(cv.crabs$k,cv.crabs$dev,ylab="cv error", xlab="k",type="b")
```






```
par(mfrow=c(1,1))
prune.crabs=prune.misclass(tree.crabs,best=8)
plot(prune.crabs)
text(prune.crabs,pretty=0)
```




```
summary(prune.crabs)
```

```
##
## Classification tree:
## snip.tree(tree = tree.crabs, nodes = c(15L, 18L, 39L, 8L))
## Variables actually used in tree construction:
## [1] "FL" "CW" "BD"
## Number of terminal nodes:  8
## Residual mean deviance:  0.5125 = 77.9 / 152
## Misclassification error rate: 0.09375 = 15 / 160
```

```
#training and test error
test.pred=predict(prune.crabs,crabs_test,type="class")
tb_test = table(test.pred,crabs_test[,"sp"])
testerr_singletree = 1 - sum(diag(tb_test))/sum(tb_test) #test errors

train.pred=predict(prune.crabs,crabs_train,type="class")
tb_train = table(train.pred,crabs_train[,"sp"])
trainerr_singletree = 1 - sum(diag(tb_train))/sum(tb_train) #train errors

testerr_singletree
```

```
## [1] 0.1
```

```
trainerr_singletree
```

```
## [1] 0.09375
```

c)

We set 'mtry' = 3 and 'ntree' = 1000 to use three randomly selected predictors at each split, and 1000 trees total. The variable importance plot shows FL, BD, and CW as the top three important variables. This result agrees with what we observed from a single tree.

```
library(randomForest)
```

```
## randomForest 4.6-12
```

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

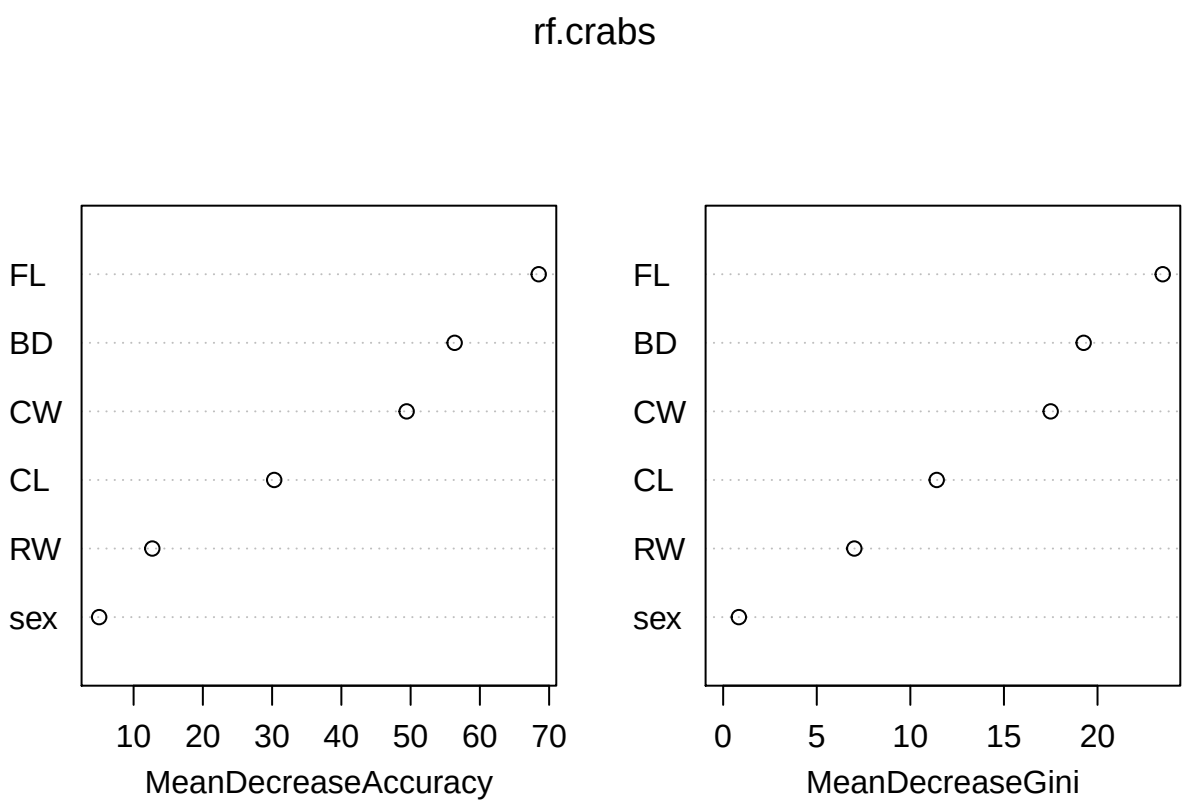
```
rf.crabs=randomForest(sp~.-index,data=crabs_train,mtry=3,ntree=1000,importance=TRUE)
rf.crabs
```

```
##
## Call:
##  randomForest(formula = sp ~ . - index, data = crabs_train, mtry = 3,      ntree = 1000, importance =
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 13.12%
## Confusion matrix:
##    B  O class.error
## B 68 12      0.1500
## O  9 71      0.1125
```

```
importance(rf.crabs)
```

```
##             B         O MeanDecreaseAccuracy MeanDecreaseGini
## sex  3.963740  2.924591             5.028778        0.8363914
## FL  41.225418 58.819339            68.492650       23.4831055
## RW  -1.205359 15.720313            12.702611        7.0067926
## CL  17.196297 18.917046            30.303853       11.4088366
## CW  35.852626 40.876920            49.425451       17.4965365
## BD  55.327066 27.521350            56.354521       19.2584665
```

```
varImpPlot(rf.crabs)
```




```
#training error calculated using predicted classes
mean(rf.crabs$predicted != crabs_train[,"sp"])
```

```
## [1] 0.13125
```

```
#training error calculated using confusion matrix
rf.crabs$confusion
```

```
##    B  O class.error
## B 68 12      0.1500
## O  9 71      0.1125
```

```
21/160
```

```
## [1] 0.13125
```

```
#test error
pred.rf = predict(rf.crabs,newdata=crabs_test)
table(pred.rf, crabs_test[,"sp"])
```

```
##
## pred.rf  B  O
##       B 19  1
##       O  1 19
```

```
2/40
```

```
## [1] 0.05
```

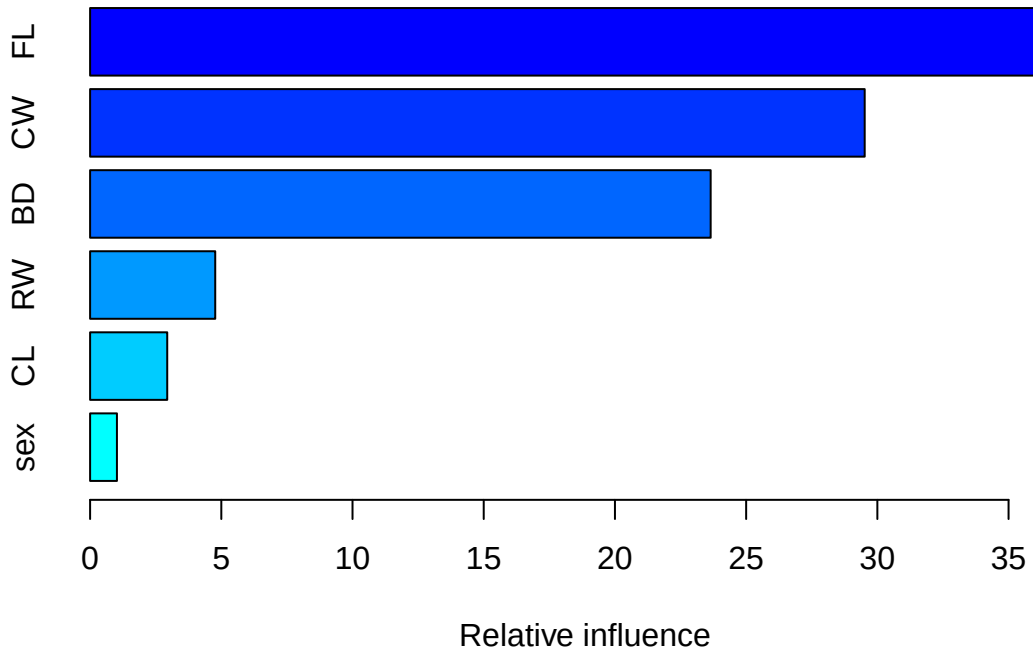
d)

adaboost requires the response variable to be numeric instead of factor. So we recode the response variable with 0, 1 values. To run adaboost, we set 'distribution' = "adaboost". To calculate training and test errors as a function of the number of trees, we get predictions for every tree values from 1 to 1000. This can be done by setting 'n.trees' in predict function equal to a vector of values. type="response" returns a class probability so probability > 0.5 means we classify Y as class 1.

```
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 3.2.5
## Loading required package: survival
## Loading required package: lattice
## Warning: package 'lattice' was built under R version 3.2.5
## Loading required package: splines
## Loading required package: parallel
## Loaded gbm 2.1.3
```

```
boost.crabs=gbm(ifelse(sp=="B",1,0)~.-index,data=crabs_train,distribution="adaboost",
                n.trees=1000,interaction.depth=4)
summary(boost.crabs)
```

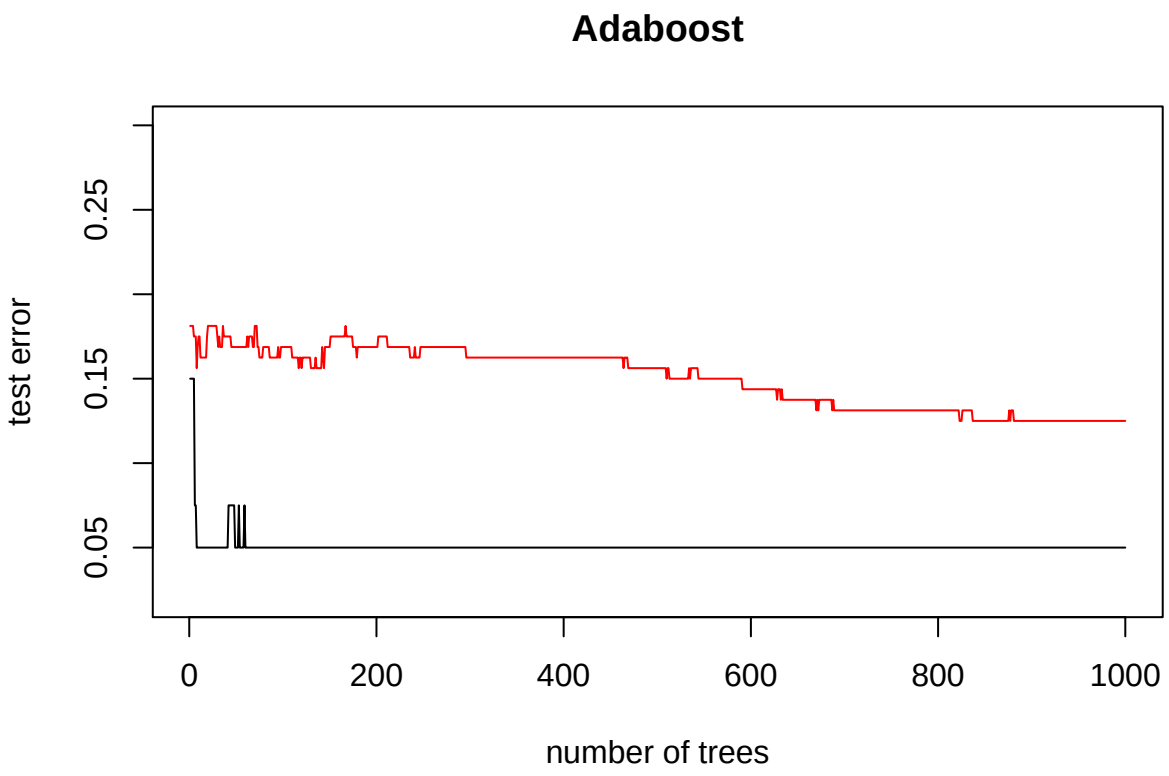



```
##     var   rel.inf
## FL   FL 38.105460
## CW   CW 29.516844
## BD   BD 23.647742
## RW   RW  4.768597
## CL   CL  2.939260
## sex sex  1.022097
```

```
#test error
pred.adaboost.test = predict(boost.crabs,newdata=crabs_test, n.trees=1:1000, type="response")
yhat.adaboost.test = apply(pred.adaboost.test,2,function(x) ifelse(x>0.5,'B','O'))
testerr.adaboost = apply(yhat.adaboost.test,2,function(x) mean(x != crabs_test[,"sp"]))

#training error
pred.adaboost.train = predict(boost.crabs, newdata=crabs_train, n.trees=1:1000, type="response")
yhat.adaboost.train = apply(pred.adaboost.train,2,function(x) ifelse(x>0.5,'B','O'))
trainerr.adaboost = apply(yhat.adaboost.train,2,function(x) mean(x != crabs_train[,"sp"]))

plot(1:1000, testerr.adaboost, type='l', ylim=c(0.02,0.3),xlab="number of trees",
     ylab="test error", main="Adaboost")
lines(1:1000, trainerr.adaboost, col="red")
```



e)

All methods chooses FL, CW, and BD as top three important variables. Based on test error, both random forest and adaboost performs well. Even though it's slightly higher, the single tree has quite low test error as well. Based on training and test error, adaboost looks the best but all methods are quite good and their results are consistent.

	Training error	Test error
Single Tree	0.09375	0.10
Random Forest	0.13125	0.05
Adaboost	0.12500	0.05