# Tree-based Methods

## The Data Set

```
library(tree)
library(randomForest)
```

```
## randomForest 4.7-1.1
```

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

```
library(gbm)
```

```
## Loaded gbm 2.1.8.1
```

```
biodeg = read.csv("biodeg.csv", stringsAsFactors=TRUE)
head(biodeg)
```

```
##   SpMaxL   JDze nHM F01NN F04CN NssssC nCb    C nCp nO F03CN  SdssC HyWiBm
## 1  3.919 2.6909   0     0     0      0   0 31.4   2  0     0  0.000  3.106
## 2  4.170 2.1144   0     0     0      0   0 30.8   1  1     0  0.000  2.461
## 3  3.932 3.2512   0     0     0      0   0 26.7   2  4     0  0.000  3.279
## 4  3.000 2.7098   0     0     0      0   0 20.0   0  2     0  0.000  2.100
## 5  4.236 3.3944   0     0     0      0   0 29.4   2  4     0 -0.271  3.449
## 6  4.236 3.4286   0     0     0      0   0 28.6   2  4     0 -0.275  3.313
##     LOC  SM6L F03CO    Me    Mi nNN nArNO2 nCRX3 SpPosABp nCIR B01CBr B03CCl
## 1 2.550 9.002     0 0.960 1.142   0      0     0    1.201    0      0      0
## 2 1.393 8.723     1 0.989 1.144   0      0     0    1.104    1      0      0
## 3 2.585 9.110     0 1.009 1.152   0      0     0    1.092    0      0      0
## 4 0.918 6.594     0 1.108 1.167   0      0     0    1.024    0      0      0
## 5 2.753 9.528     2 1.004 1.147   0      0     0    1.137    0      0      0
## 6 2.522 9.383     1 1.014 1.149   0      0     0    1.119    0      0      0
##   N073 SpMaxA Psii1d B04CBr    SdO  TI2L nCrt C026 F02CN nHDon SpMaxBm PsiiA nN
## 1    0  1.932  0.011      0  0.000 4.489    0    0     0     0   2.949 1.591  0
## 2    0  2.214 -0.204      0  0.000 1.542    0    0     0     0   3.315 1.967  0
## 3    0  1.942 -0.008      0  0.000 4.891    0    0     0     1   3.076 2.417  0
## 4    0  1.414  1.073      0  8.361 1.333    0    0     0     1   3.046 5.000  0
## 5    0  1.985 -0.002      0 10.348 5.588    0    0     0     0   3.351 2.405  0
## 6    0  1.980 -0.008      0 10.276 4.746    0    0     0     0   3.351 2.556  0
##   SM6Bm nArCOOR nX class
## 1 7.253       0  0    RB
## 2 7.257       0  0    RB
## 3 7.601       0  0    RB
## 4 6.690       0  0    RB
## 5 8.003       0  0    RB
## 6 7.904       0  0    RB
```

# Tasks

# Training and Test Data

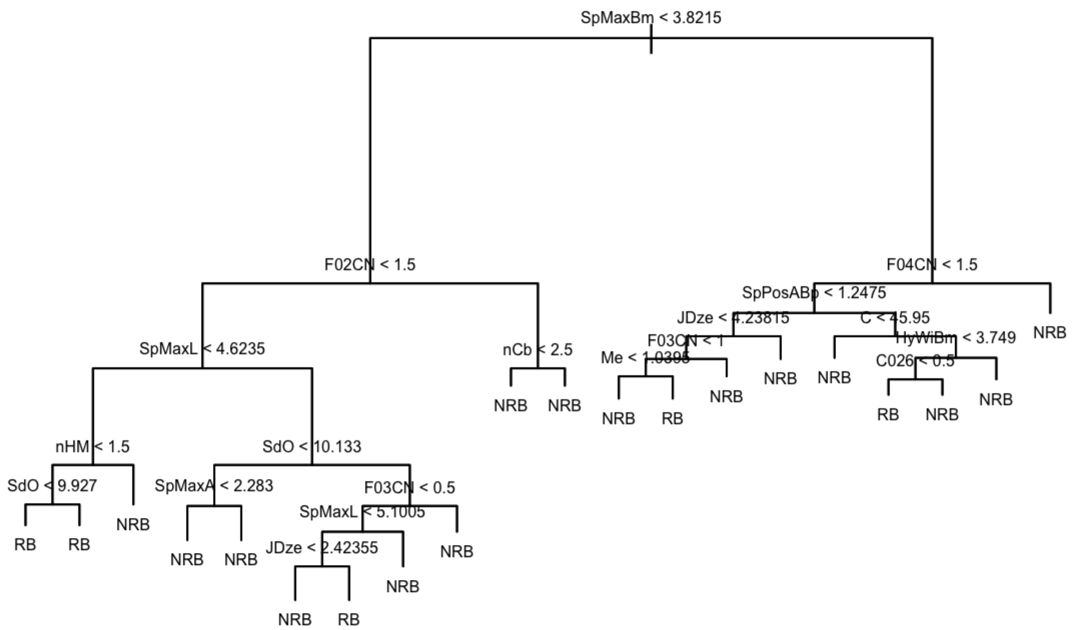
1. Randomly divide the data set into two halves, and save them in two data frames named train and test.

```
set.seed(769)
index <- sample(rep(1:2, length.out=nrow(biodeg)))
train <- biodeg[index ==1, ]
test <- biodeg[index ==2, ]
```

# Classification trees

2. Fit an unpruned classification tree to the training data. Plot it (as pretty as you can). Identify three most important variables from this classification tree.

```
r = tree(class ~ ., data=train)
plot(r)
text(r, pretty=1, cex=0.5)
```




For the heights of splits are proportional to deviance reductions, so we can see from the tree that the 3 most important variables are F02CN, F04CN and nCb.

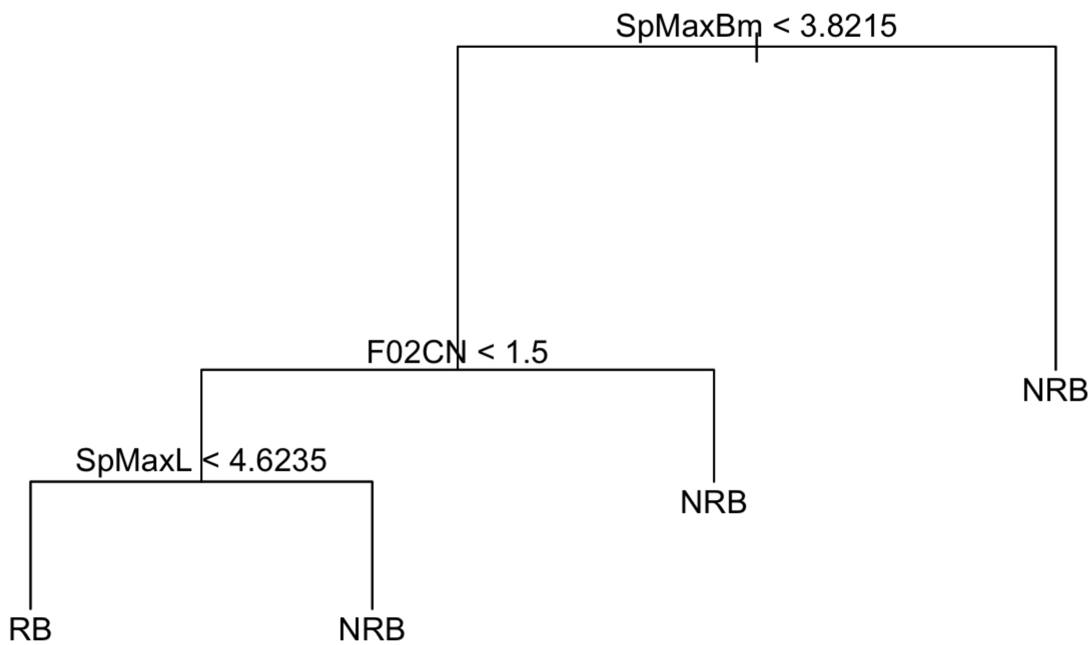
3. Compute the training and test errors. Write a function errors(fit, fhat.tree, train, test) where fit is the output of tree() and fhat.tree is a function that uses fit and computes the class labels for a data set (an argument of fhat.tree).

```
fhat.tree <- function(fit, dataset){
  p = predict(fit, dataset)
  yhat = levels(dataset$class)[apply(p, 1, which.max)]
  yhat
}

errors <- function(fit, fhat.tree, train, test){
  train.error = mean(fhat.tree(fit, train) != train$class)
  test.error = mean(fhat.tree(fit, test) != test$class)
  c(train.error, test.error)
}

(errors(r, fhat.tree, train, test))
```

```
## [1] 0.1098485 0.1726755
```

4. Consider pruning the tree using cross-validation with deviance. Produce a pruned tree based by selecting a cost-complexity parameter value, and plot it. Compute the training and test errors for this pruned tree. Do you think the pruning helps?

```
cv.r = cv.tree(r)
j.min = which.min(cv.r$dev)
k = cv.r$k[j.min]
r2 = prune.tree(r, k=k)
plot(r2)
text(r2, pretty=0)
```



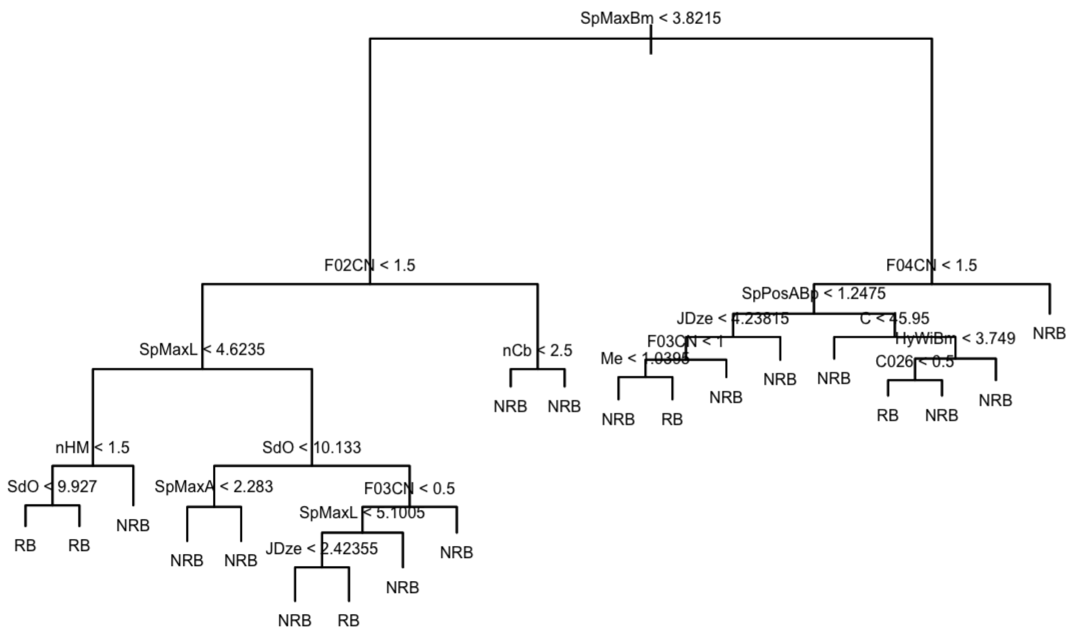
```
(errors(r2, fhat.tree, train, test))
```

```
## [1] 0.1856061 0.2314991
```

The tree is considerably smaller, which is computational efferent, so I think pruning helps in the trade-off of accuracy.

5. Consider pruning the tree using cross-validation with misclassification rates. Produce a pruned tree by selecting a tree size, and plot it. Compute the training and test errors for this pruned tree. Do you think the pruning helps?

```
cv.r = cv.tree(r, FUN = prune.misclass)
j.min = which.min(cv.r$dev)
size = cv.r$size[j.min]
r3 = prune.tree(r, best=size)
plot(r3)
text(r3, pretty=1, cex=0.5)
```




```
(errors(r3, fhat.tree, train, test))
```

```
## [1] 0.1098485 0.1726755
```

It produced the exactly the same tree as the unpruned tree, so I don't think pruning helps this time.

# Bagging

6. Produce a Bagging model for the training data with 500 trees construnted. What are the three most important variables, in terms of decreasing the Gini index, according to Bagging?

```
p = ncol(biodeg) - 1
(r = randomForest(class ~ ., data=train, mtry=p, importance=TRUE, ntree=500))
```

```
##
## Call:
##  randomForest(formula = class ~ ., data = train, mtry = p, importance = TRUE,
ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 41
##
##         OOB estimate of  error rate: 15.72%
## Confusion matrix:
##     NRB  RB class.error
## NRB 325  33  0.09217877
## RB   50 120  0.29411765
```

```
round(importance(r), 2)
```

```
##             NRB    RB MeanDecreaseAccuracy MeanDecreaseGini
## SpMaxL    5.02 24.49                24.04            20.27
## JDze      7.24  6.21                 9.68             5.79
## nHM       0.56  9.50                 9.99             1.39
## F01NN     2.38  2.65                 2.95             0.26
## F04CN     1.78  9.96                 9.10             2.72
## NssssC    4.64  7.37                 7.21             1.38
## nCb       3.63  2.69                 5.14             0.80
## C         1.52  2.96                 3.52             5.35
## nCp       1.68 11.36                 9.62             2.91
## nO        5.69  5.03                 8.09             3.37
## F03CN    -1.46 10.69                 9.43             2.46
## SdssC    15.45 17.22                21.81            12.51
## HyWiBm   13.18  6.66                15.44             7.23
## LOC       6.52  8.14                10.77             5.72
## SM6L      9.05  4.81                11.23             4.52
## F03CO     8.40  8.55                11.61             4.87
## Me        8.90  2.90                 9.93             5.56
## Mi        9.03  8.44                12.24             7.68
## nNN       2.46  1.42                 2.31             0.19
## nArNO2    2.73  2.24                 3.16             0.15
## nCRX3     0.00  0.00                 0.00             0.01
## SpPosABp 13.03  9.53                16.79            11.43
## nCIR      0.94  3.24                 3.41             0.48
## B01CBr    1.00  0.00                 1.00             0.03
## B03CCl    0.38  4.54                 3.66             0.36
## N073      1.42 -2.46                 0.49             0.28
## SpMaxA   10.01 12.34                16.80            12.35
## Psii1d   12.38 -3.99                10.91             3.72
## B04CBr   -1.42  0.00                -1.42             0.05
## SdO       6.12  4.66                 7.63             3.70
## TI2L      8.67  4.58                 9.87             5.44
## nCrt      4.96  8.94                 9.85             1.60
## C026      3.73  4.17                 5.52             1.09
## F02CN     8.06 22.24                21.84            13.05
## nHDon     2.31  4.44                 5.33             2.36
## SpMaxBm  21.34 20.33                31.57            42.09
## PsiiA    12.14  6.85                14.31             8.81
## nN        6.48 14.32                13.68             6.51
## SM6Bm    20.58 10.95                24.09            17.33
## nArCOOR   2.00  8.99                 8.15             3.08
## nX       -1.75 11.42                10.72             1.60
```

SpMaxBm, SM6Bm and SpMaxL are the three most important variables according to bagging.

7. Compute both the training and test errors of this Bagging predictor. Is your test error similar to the OOB estimate? Do you think Bagging helps prediction here?

```
yhat = predict(r, train)
(mean(train$class != yhat))
```

```
## [1] 0
```

```
yhat = predict(r, test)
(mean(test$class != yhat))
```

```
## [1] 0.1442125
```

The test set error is pretty close to OOB error, and the error rate slightly decreased. So I think bagging helps prediction.

# Random Forests

8. Produce a Random Forest model with 500 trees constructed. What are the three most important variables, in terms of accuracy, according to Random Forest?

```
(r = randomForest(class ~ ., data=train, mtry=10, importance=TRUE, ntree=500))
```

```
##
## Call:
##  randomForest(formula = class ~ ., data = train, mtry = 10, importance = TRUE,
ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 10
##
##         OOB estimate of  error rate: 15.53%
## Confusion matrix:
##     NRB  RB class.error
## NRB 324  34  0.09497207
## RB   48 122  0.28235294
```

```
round(importance(r), 2)
```

```
##              NRB    RB MeanDecreaseAccuracy MeanDecreaseGini
## SpMaxL   10.59 18.62                19.60            18.49
## JDze      6.86  5.47                 9.22             6.84
## nHM       5.02  9.36                10.63             3.23
## F01NN     1.59  4.04                 4.34             0.36
## F04CN     5.08  9.67                 9.98             3.03
## NssssC    4.75  6.92                 7.36             2.21
## nCb       5.50  4.25                 7.10             2.75
## C         4.36  1.66                 4.79             6.31
## nCp       5.14  8.59                 9.59             3.34
## nO        6.61 10.21                12.47             5.23
## F03CN     5.00 10.36                11.27             4.12
## SdssC    11.44 12.80                15.95             9.31
## HyWiBm   12.73  8.56                14.79             9.89
## LOC       2.46  8.59                 7.80             5.86
## SM6L      8.83  9.82                13.78             8.26
## F03CO     8.05  8.98                12.81             5.10
## Me        6.94  3.10                 8.21             5.82
## Mi        7.65  6.63                10.64             7.01
## nNN       1.15  1.00                 1.41             0.25
## nArNO2    2.57  2.13                 2.91             0.15
## nCRX3     0.00  1.00                 1.00             0.03
## SpPosABp 13.36 11.66                18.13            14.47
## nCIR      2.24  3.43                 4.36             1.22
## B01CBr   -1.61  1.00                -1.13             0.04
## B03CCl   -0.54  3.07                 1.97             0.26
## N073      1.56 -1.74                 0.98             0.20
## SpMaxA   10.23 12.85                15.96            14.42
## Psii1d    6.91 -2.28                 5.88             4.26
## B04CBr   -2.55  0.00                -2.56             0.07
## SdO       7.39 10.34                12.57             5.41
## TI2L      6.73  7.31                10.06             6.22
## nCrt      3.84  4.28                 5.97             1.27
## C026      5.57  5.49                 6.72             2.03
## F02CN     9.14 15.27                16.97             8.99
## nHDon     3.50  4.06                 5.77             2.76
## SpMaxBm  14.15 15.78                20.84            22.69
## PsiiA     8.52  5.95                10.98             7.65
## nN        9.62 16.12                16.81             8.65
## SM6Bm    15.15 13.59                21.59            17.56
## nArCOOR   2.49  6.73                 6.70             2.28
## nX        3.90  8.80                 9.35             1.73
```

SpMaxBm, SpMaxL and SM6Bm are the most three important variables according to random forrest.

9. Compute both the training and test errors of this Random Forest predictor. Is your test error similar to the OOB estimate? Do you think the tweak used by Random Forest helps prediction here?

```
yhat = predict(r, train)
(mean(train$class != yhat))
```

```
## [1] 0
```

```
yhat = predict(r, test)
(mean(test$class != yhat))
```

```
## [1] 0.1347249
```

The test set error is pretty close to OOB error, and the error rate was even better than that with bagging. So I think it helps prediction.

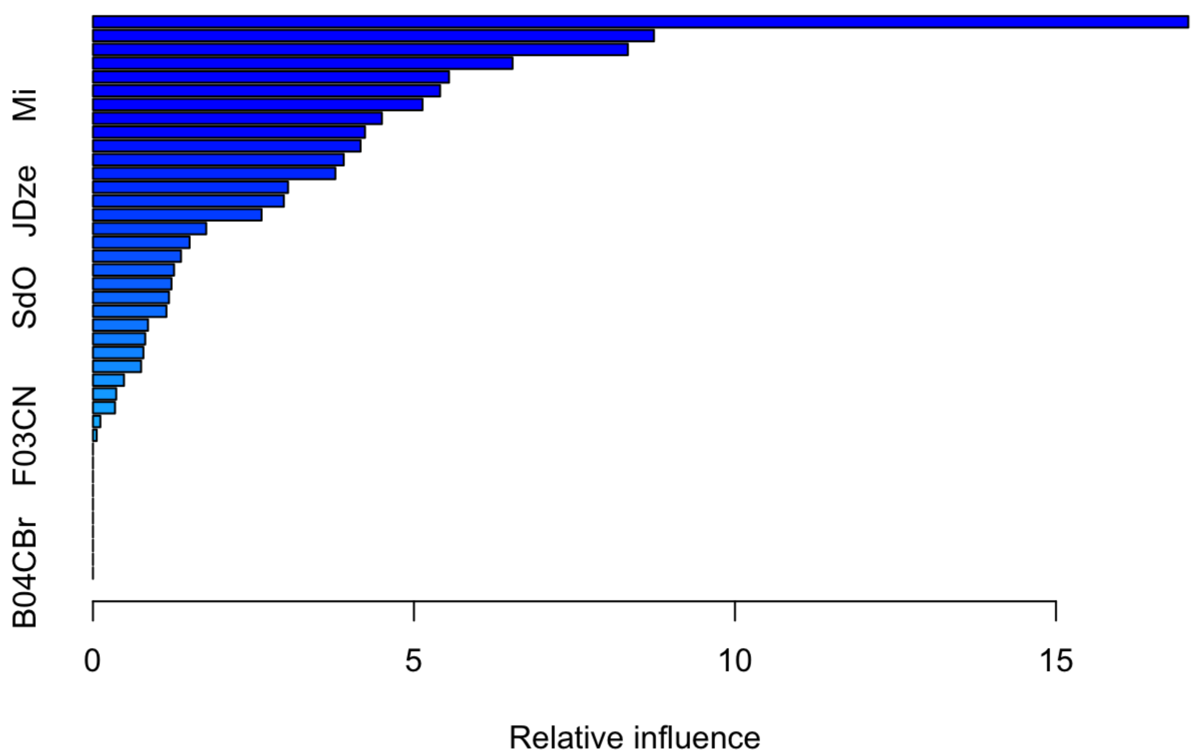
# Boosting

10. Produce a Boosting model, with 500 trees constructed. What are the three most important variables, according to Boosting?

```
train$class = as.integer(train$class)-1
test$class = as.integer(test$class)-1
(r = gbm(class ~ ., data=train, distribution="bernoulli", n.trees=500))
```

```
## gbm(formula = class ~ ., distribution = "bernoulli", data = train,
##     n.trees = 500)
## A gradient boosted model with bernoulli loss function.
## 500 iterations were performed.
## There were 41 predictors of which 31 had non-zero influence.
```

```
summary(r)
```



```
##                  var     rel.inf
## SpMaxBm   SpMaxBm 17.06033867
## SdssC       SdssC  8.73791894
## SpMaxL     SpMaxL  8.33174383
## SpPosABp SpPosABp  6.53530162
## HyWiBm     HyWiBm  5.54586333
## PsiiA       PsiiA  5.40705813
## Mi             Mi  5.13265840
## F02CN       F02CN  4.50074247
## SM6Bm       SM6Bm  4.23748234
## LOC           LOC  4.16786118
## nN             nN  3.90635146
## SpMaxA     SpMaxA  3.77574882
## C               C  3.03901731
## JDze         JDze  2.97354214
## TI2L         TI2L  2.62600574
## Psii1d     Psii1d  1.76390280
## Me             Me  1.50505290
## F03CO       F03CO  1.37128421
## nO             nO  1.26297730
## nHDon       nHDon  1.22289433
## SdO           SdO  1.18392919
## NssssC     NssssC  1.14402089
## nArCOOR   nArCOOR  0.85722592
## C026         C026  0.81399057
## nCrt         nCrt  0.78544697
## SM6L         SM6L  0.74975356
## nX             nX  0.48314202
## nCb           nCb  0.36209309
## nCp           nCp  0.34350445
## nHM           nHM  0.11477904
## F03CN       F03CN  0.05836838
## F01NN       F01NN  0.00000000
## F04CN       F04CN  0.00000000
## nNN           nNN  0.00000000
## nArNO2     nArNO2  0.00000000
## nCRX3       nCRX3  0.00000000
## nCIR         nCIR  0.00000000
## B01CBr     B01CBr  0.00000000
## B03CCl     B03CCl  0.00000000
## N073         N073  0.00000000
## B04CBr     B04CBr  0.00000000
```

SpMaxBm, SdssC and SpMaxL are the three most important variables according to boosting.

11. Compute both the training and test errors of this Boosting predictor. Do you think Boosting helps prediction here?

```
yhat = (predict(r, train, type="response") > 0.5)
```

```
## Using 500 trees...
```

```
(mean(train$class != yhat))
```

```
## [1] 0.0530303
```

```
yhat = (predict(r, test, type="response") > 0.5)
```

```
## Using 500 trees...
```

```
(mean(test$class != yhat))
```

```
## [1] 0.1480076
```

It helps in regards to the pruned decision tree, but the test error could be too optimistic compared to OOB error from bagging and random forest.

12. Demonstrate that Boosting can overfit.

```
r = gbm(class ~ ., data=train, distribution="bernoulli", n.trees=200000, n.cores=8 )
yhat = (predict(r, train, type="response") > 0.5)
```

```
## Using 200000 trees...
```

```
(mean(train$class == yhat))
```

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

```
yhat = (predict(r, test, type="response") > 0.5)
```

```
## Using 200000 trees...
```

```
(mean(test$class == yhat))
```

```
## [1] 0.8045541
```

# Summary

In this lab, we played around QSAR biodegradation data set which uses 41 molecular descriptors (SpMax_L, etc.) to predict class labels (ready biodegradable (RB) and not ready biodegradable (NRB)).

We used the tree family to do the classification. We went from the foundation - a single decision tree, to bagging which over-samples the data set with replacement so that different models can be combined to have a unbiased estimate, to random forest with the tweak to use only some of the variables to find the optimal cutoff points, to boosting that gradually makes to a better prediction.

We found that SpMaxBm is the common important variable recognised by all the models, and all the models have a fair prediction of the class labels.