lab09

Yixuan Li, UPI:yil845

10 October 2021

The Data: The data source for this lab is from UCI Machine Learning Repository - QSAR biodegradation data set.

Import data

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(189)
ind<-sample(1:nrow(biodeg))
train<-biodeg[ind%%2==0,]
test<-biodeg[ind%%2!=0,]
### teacher's code
n = nrow(biodeg)
i = sample(n, round(n/2))
train = biodeg[i,]
test = biodeg[-i,]
dim(train)
```

```
## [1] 528  42
```

Classification trees

2, Fit an unpruned classification tree to the data. Plot it(as pretty as you can).

The height of tree decides the deviation difference of each variables. The most important three variables are “SpMaxBm” “F02CN” “SpMaxL”. ### Because it is dendrogram, we can use the size of the jump by each split to measure the importance of a split (and thus the variable). Therefore, the three most important variables are.

```
r.tree<-tree(class~., data=train)
r.tree
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##   1) root 528 663.500 NRB ( 0.67803 0.32197 )
##     2) SpMaxBm < 3.6775 222 299.000 RB ( 0.40090 0.59910 )
##       4) F02CN < 2.5 189 237.700 RB ( 0.32275 0.67725 )
##         8) SpMaxL < 4.984 163 181.700 RB ( 0.24540 0.75460 )
##          16) PsiiA < 2.0935 44  60.630 NRB ( 0.54545 0.45455 )
##            32) LOC < 1.6905 22  20.860 NRB ( 0.81818 0.18182 ) *
##            33) LOC > 1.6905 22  25.780 RB ( 0.27273 0.72727 )
##              66) Me < 0.9695 9   0.000 RB ( 0.00000 1.00000 ) *
##              67) Me > 0.9695 13  17.940 RB ( 0.46154 0.53846 )
##               134) JDze < 3.0615 8   8.997 NRB ( 0.75000 0.25000 ) *
##               135) JDze > 3.0615 5   0.000 RB ( 0.00000 1.00000 ) *
##          17) PsiiA > 2.0935 119  93.950 RB ( 0.13445 0.86555 )
##            34) nN < 0.5 91  38.730 RB ( 0.05495 0.94505 )
##              68) PsiiA < 2.58 34  28.390 RB ( 0.14706 0.85294 )
##               136) C < 31.2 13  17.320 RB ( 0.38462 0.61538 ) *
##               137) C > 31.2 21   0.000 RB ( 0.00000 1.00000 ) *
##              69) PsiiA > 2.58 57   0.000 RB ( 0.00000 1.00000 ) *
##            35) nN > 0.5 28  37.520 RB ( 0.39286 0.60714 )
##              70) C < 32.3 16  21.170 NRB ( 0.62500 0.37500 ) *
##              71) C > 32.3 12   6.884 RB ( 0.08333 0.91667 ) *
##         9) SpMaxL > 4.984 26  25.460 NRB ( 0.80769 0.19231 ) *
##       5) F02CN > 2.5 33  28.070 NRB ( 0.84848 0.15152 )
##        10) HyWiBm < 3.1785 20   0.000 NRB ( 1.00000 0.00000 ) *
##        11) HyWiBm > 3.1785 13  17.320 NRB ( 0.61538 0.38462 ) *
##     3) SpMaxBm > 3.6775 306 225.700 NRB ( 0.87908 0.12092 )
##       6) SdssC < -0.595 51  69.740 NRB ( 0.56863 0.43137 )
##        12) SpMaxL < 5.1135 30  36.650 RB ( 0.30000 0.70000 )
##          24) C026 < 1.5 25  21.980 RB ( 0.16000 0.84000 )
##            48) SM6Bm < 8.552 10  13.460 RB ( 0.40000 0.60000 ) *
##            49) SM6Bm > 8.552 15   0.000 RB ( 0.00000 1.00000 ) *
##          25) C026 > 1.5 5   0.000 NRB ( 1.00000 0.00000 ) *
##        13) SpMaxL > 5.1135 21   8.041 NRB ( 0.95238 0.04762 ) *
##       7) SdssC > -0.595 255 114.100 NRB ( 0.94118 0.05882 )
##        14) SpMaxA < 2.3585 161  99.760 NRB ( 0.90683 0.09317 )
##          28) nO < 2.5 130  54.520 NRB ( 0.94615 0.05385 ) *
##          29) nO > 2.5 31  35.400 NRB ( 0.74194 0.25806 )
##            58) nArNO2 < 0.5 19  25.860 NRB ( 0.57895 0.42105 )
##             116) JDze < 4.18945 12  15.280 RB ( 0.33333 0.66667 )
##               232) JDze < 3.2482 6   7.638 NRB ( 0.66667 0.33333 ) *
##               233) JDze > 3.2482 6   0.000 RB ( 0.00000 1.00000 ) *
##             117) JDze > 4.18945 7   0.000 NRB ( 1.00000 0.00000 ) *
##            59) nArNO2 > 0.5 12   0.000 NRB ( 1.00000 0.00000 ) *
##        15) SpMaxA > 2.3585 94   0.000 NRB ( 1.00000 0.00000 ) *
```

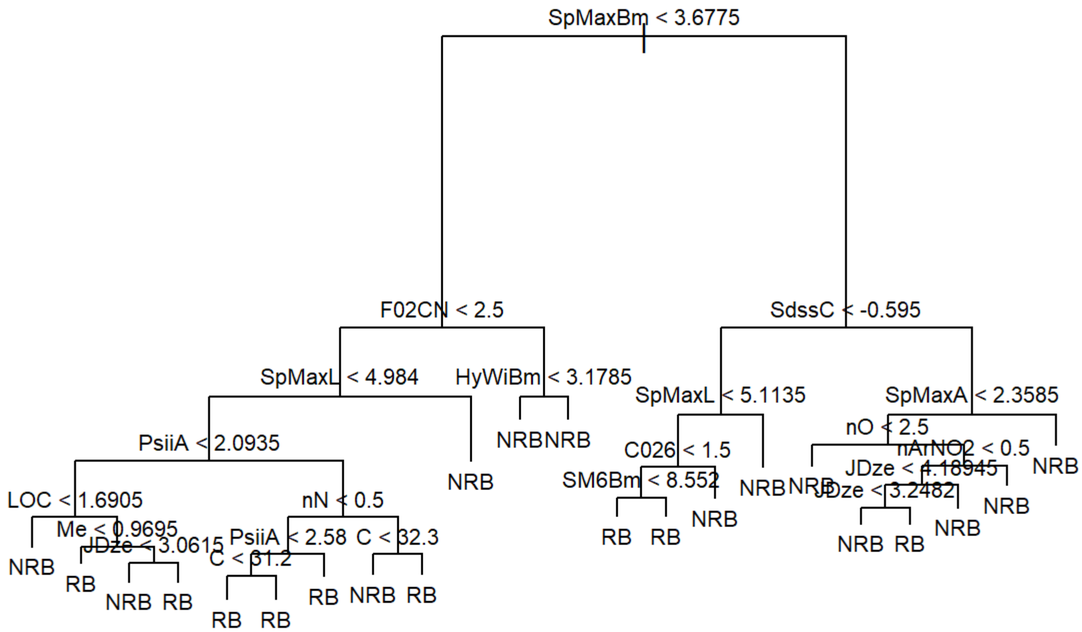
```
summary(r.tree)
```

```
##
## Classification tree:
## tree(formula = class ~ ., data = train)
## Variables actually used in tree construction:
##  [1] "SpMaxBm" "F02CN"   "SpMaxL"  "PsiiA"   "LOC"     "Me"      "JDze"
##  [8] "nN"      "C"       "HyWiBm"  "SdssC"   "C026"    "SM6Bm"   "SpMaxA"
## [15] "nO"      "nArNO2"
## Number of terminal nodes:  22
## Residual mean deviance:  0.3986 = 201.7 / 506
## Misclassification error rate: 0.07955 = 42 / 528
```

```
plot(r.tree)
r.tree
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##   1) root 528 663.500 NRB ( 0.67803 0.32197 )
##     2) SpMaxBm < 3.6775 222 299.000 RB ( 0.40090 0.59910 )
##       4) F02CN < 2.5 189 237.700 RB ( 0.32275 0.67725 )
##         8) SpMaxL < 4.984 163 181.700 RB ( 0.24540 0.75460 )
##          16) PsiiA < 2.0935 44  60.630 NRB ( 0.54545 0.45455 )
##            32) LOC < 1.6905 22  20.860 NRB ( 0.81818 0.18182 ) *
##            33) LOC > 1.6905 22  25.780 RB ( 0.27273 0.72727 )
##              66) Me < 0.9695 9   0.000 RB ( 0.00000 1.00000 ) *
##              67) Me > 0.9695 13  17.940 RB ( 0.46154 0.53846 )
##               134) JDze < 3.0615 8   8.997 NRB ( 0.75000 0.25000 ) *
##               135) JDze > 3.0615 5   0.000 RB ( 0.00000 1.00000 ) *
##          17) PsiiA > 2.0935 119  93.950 RB ( 0.13445 0.86555 )
##            34) nN < 0.5 91  38.730 RB ( 0.05495 0.94505 )
##              68) PsiiA < 2.58 34  28.390 RB ( 0.14706 0.85294 )
##               136) C < 31.2 13  17.320 RB ( 0.38462 0.61538 ) *
##               137) C > 31.2 21   0.000 RB ( 0.00000 1.00000 ) *
##              69) PsiiA > 2.58 57   0.000 RB ( 0.00000 1.00000 ) *
##            35) nN > 0.5 28  37.520 RB ( 0.39286 0.60714 )
##              70) C < 32.3 16  21.170 NRB ( 0.62500 0.37500 ) *
##              71) C > 32.3 12   6.884 RB ( 0.08333 0.91667 ) *
##         9) SpMaxL > 4.984 26  25.460 NRB ( 0.80769 0.19231 ) *
##       5) F02CN > 2.5 33  28.070 NRB ( 0.84848 0.15152 )
##        10) HyWiBm < 3.1785 20   0.000 NRB ( 1.00000 0.00000 ) *
##        11) HyWiBm > 3.1785 13  17.320 NRB ( 0.61538 0.38462 ) *
##     3) SpMaxBm > 3.6775 306 225.700 NRB ( 0.87908 0.12092 )
##       6) SdssC < -0.595 51  69.740 NRB ( 0.56863 0.43137 )
##        12) SpMaxL < 5.1135 30  36.650 RB ( 0.30000 0.70000 )
##          24) C026 < 1.5 25  21.980 RB ( 0.16000 0.84000 )
##            48) SM6Bm < 8.552 10  13.460 RB ( 0.40000 0.60000 ) *
##            49) SM6Bm > 8.552 15   0.000 RB ( 0.00000 1.00000 ) *
##          25) C026 > 1.5 5   0.000 NRB ( 1.00000 0.00000 ) *
##        13) SpMaxL > 5.1135 21   8.041 NRB ( 0.95238 0.04762 ) *
##       7) SdssC > -0.595 255 114.100 NRB ( 0.94118 0.05882 )
##        14) SpMaxA < 2.3585 161  99.760 NRB ( 0.90683 0.09317 )
##          28) nO < 2.5 130  54.520 NRB ( 0.94615 0.05385 ) *
##          29) nO > 2.5 31  35.400 NRB ( 0.74194 0.25806 )
##            58) nArNO2 < 0.5 19  25.860 NRB ( 0.57895 0.42105 )
##             116) JDze < 4.18945 12  15.280 RB ( 0.33333 0.66667 )
##               232) JDze < 3.2482 6   7.638 NRB ( 0.66667 0.33333 ) *
##               233) JDze > 3.2482 6   0.000 RB ( 0.00000 1.00000 ) *
##             117) JDze > 4.18945 7   0.000 NRB ( 1.00000 0.00000 ) *
##            59) nArNO2 > 0.5 12   0.000 NRB ( 1.00000 0.00000 ) *
##        15) SpMaxA > 2.3585 94   0.000 NRB ( 1.00000 0.00000 ) *
```

```
text(r.tree,pretty = 0,cex=0.7)
```



3, Compute the training and test errors.

This function fhat.tree computes yhat according to fit and data, errors() function computes classification errors.

```
errors<- function(fit,f,train,test) {
  train_class = f(fit, train)
  test_class = f(fit,test)
  c(training.error=mean(train$class != train_class), test.error=mean(test$class != test_class))
}
fhat.tree <- function(fit,data){
  yhat<-predict(fit,data)
  yhat.class<-levels(data$class)[apply(yhat, 1, which.max)]
}
```

training and test error for unpruned tree

```
yhat.class<-fhat.tree(r.tree,train) ## converting probability to class index
head(yhat.class)
```

```
## [1] "RB"  "NRB" "NRB" "NRB" "NRB" "NRB"
```

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

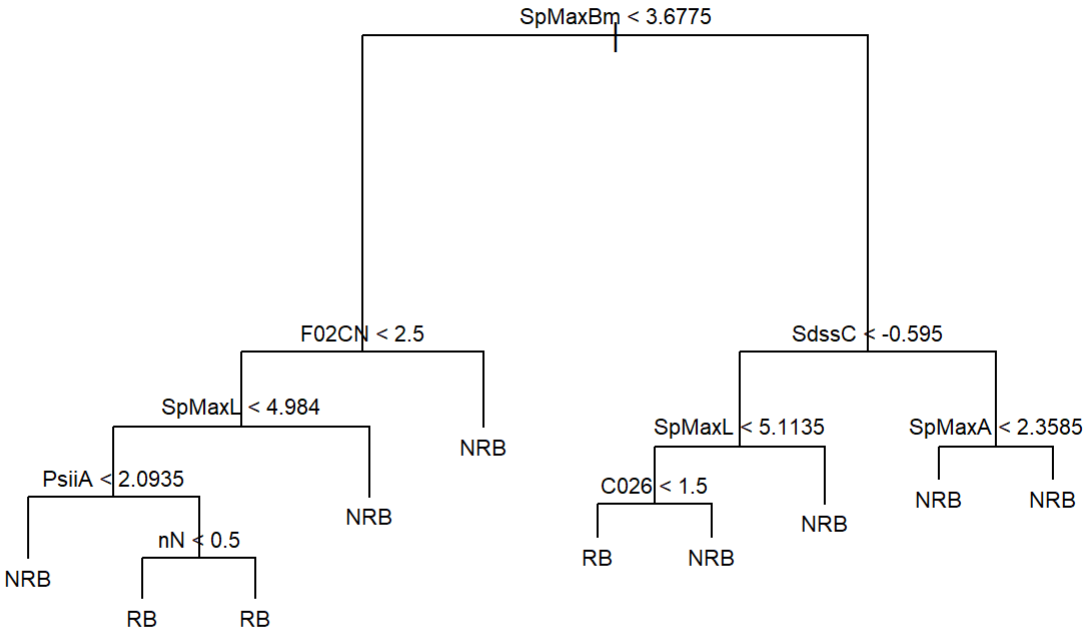
```
## training.error     test.error
##     0.07954545     0.20683112
```

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.

The results shows training & test errors do not improve with pruning. Prunning is a data-compression method of reducing size of decision tree to prevent over-fitting,(T) This is the same as returned by function errors(). The test error is much higher than the training error.

(T)While it is understandable that the training error will increase, the test error has also increased a little, from 0.197 to 0.207. The pruning does not help much in terms of reducing the misclassification rate, but it does produce a smaller tree. The pruning should likely have helped in terms of reducing the (test set) deviance.

```
cv.r = cv.tree(r.tree)
j.min<-which.min(cv.r$dev)
k = cv.r$k[j.min]
r2 = prune.tree(r.tree, k=k)              # tree pruning, based on cost-complexity
plot(r2)
text(r2, pretty=0,cex=0.7)
```




```
#### training & test errors
errors(r2,fhat.tree,train,test)
```

```
## training.error     test.error
##      0.1250000      0.2106262
```

5, Consider pruning the tree using cross-validation with misclassification rates. Produce a pruned tree by selecting a tree size, and plot it.

The results shows training & test errors increase. Pruning can help with simplify decision tree by removing less-important nodes, sometimes it may increase classification accuracy, but not with this sample.

```
(cv.r = cv.tree(r.tree))              # Cross-validation
```

```
## $size
##  [1] 22 21 19 18 17 14 12 11 10  9  8  7  6  5  4  3  2  1
##
## $dev
##  [1] 736.6644 671.4829 651.9426 654.2912 632.7191 629.4700 602.8071 602.8071
##  [9] 573.7172 574.4742 555.2797 512.8777 521.3702 524.8104 541.3779 555.9796
## [17] 552.2203 664.7430
##
## $k
##  [1]       -Inf   7.638170   8.392276   8.523261   9.466416   9.987454
##  [7]  10.705538  10.748428  13.988768  14.338653  14.668364  17.699562
## [13]  25.044679  27.067664  30.622457  33.172936  41.838027 138.882538
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
(j.min = which.min(cv.r$dev))          # smallest CV deviance
```

```
## [1] 12
```

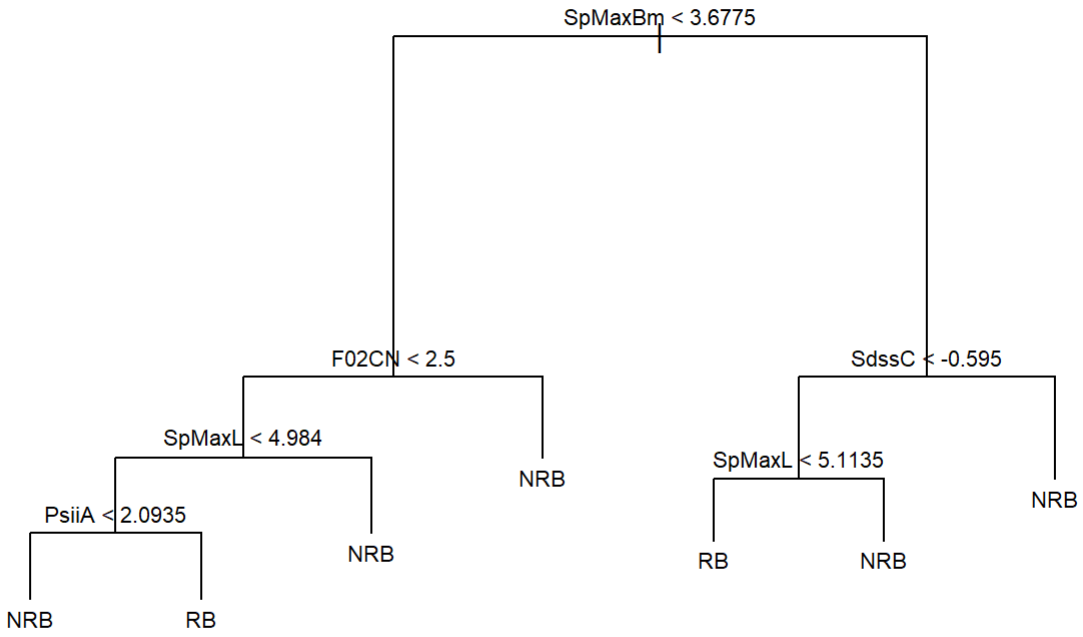
```
(size = cv.r$size[j.min])
```

```
## [1] 7
```

```
(r3 = prune.tree(r.tree, best=size))        # tree pruning, based on the best size
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 528 663.500 NRB ( 0.67803 0.32197 )
##    2) SpMaxBm < 3.6775 222 299.000 RB ( 0.40090 0.59910 )
##      4) F02CN < 2.5 189 237.700 RB ( 0.32275 0.67725 )
##        8) SpMaxL < 4.984 163 181.700 RB ( 0.24540 0.75460 )
##         16) PsiiA < 2.0935 44  60.630 NRB ( 0.54545 0.45455 ) *
##         17) PsiiA > 2.0935 119  93.950 RB ( 0.13445 0.86555 ) *
##        9) SpMaxL > 4.984 26  25.460 NRB ( 0.80769 0.19231 ) *
##      5) F02CN > 2.5 33  28.070 NRB ( 0.84848 0.15152 ) *
##    3) SpMaxBm > 3.6775 306 225.700 NRB ( 0.87908 0.12092 )
##      6) SdssC < -0.595 51  69.740 NRB ( 0.56863 0.43137 )
##       12) SpMaxL < 5.1135 30  36.650 RB ( 0.30000 0.70000 ) *
##       13) SpMaxL > 5.1135 21   8.041 NRB ( 0.95238 0.04762 ) *
##      7) SdssC > -0.595 255 114.100 NRB ( 0.94118 0.05882 ) *
```

```
plot(r3)
text(r3, pretty=0,cex=0.7)
```




```
##### training & test classification errors
errors(r3,fhat.tree,train,test)
```

```
## training.error     test.error
##      0.1344697      0.2201139
```

Bagging

6, Produce a Bagging model with 500 trees construnted.

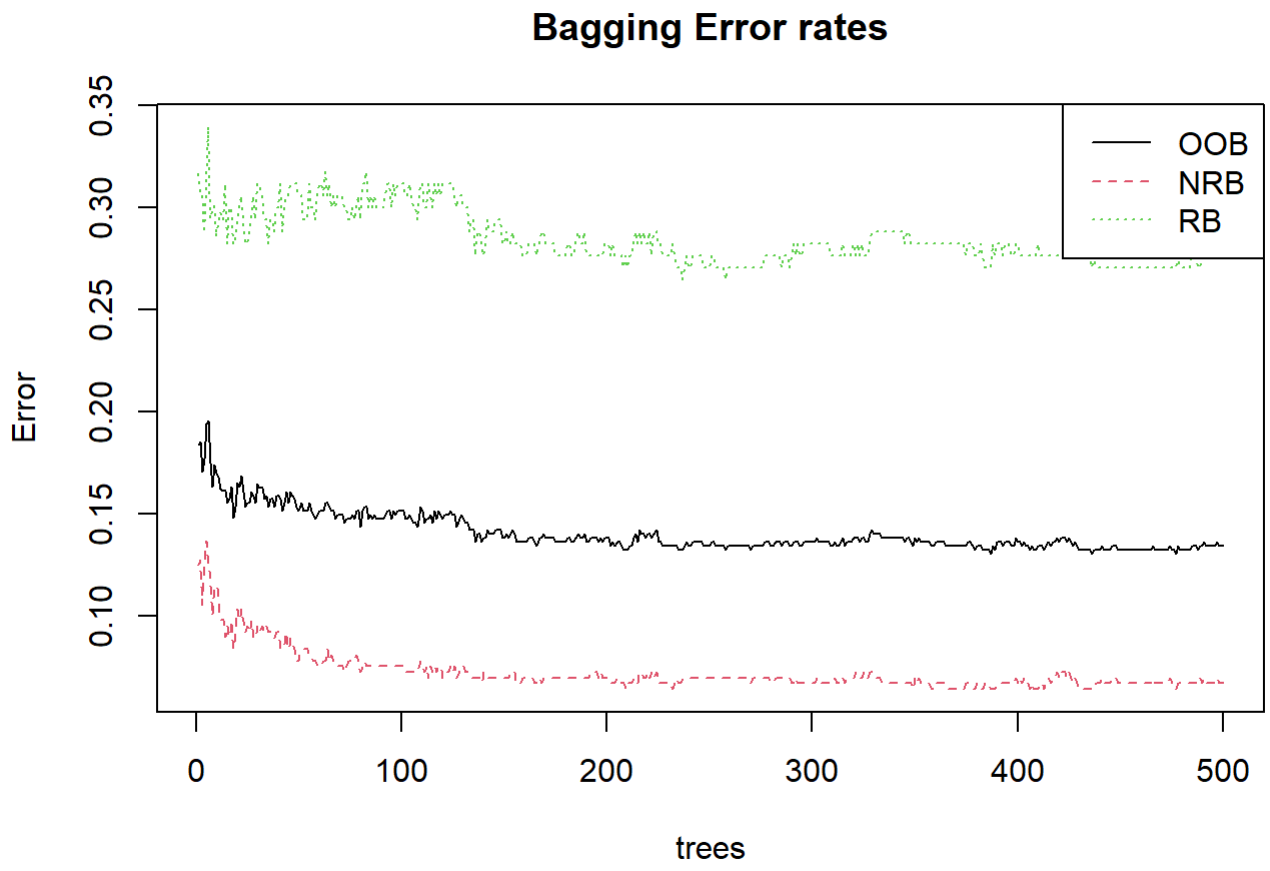
For bagging, mtry is number of variables. According to “Mean decrease Gini”,the three most important variables are SpMaxBm,SpPosABp,SpMaxL.

(T)The test error is 0.148, which is very similar to the OOB estimate of the test error: 0.140. Bagging helps prediction here. It reduces the test error from 0.203 of a pruned tree to 0.148.The test error is 0.146, which is also very similar to the OOB estimate of the test error: 0.140.RandomForest helps prediction here, if compared with the pruned tree (test error 0.203), but not much if compared with Bagging (0.148).

```
set.seed(289)
(r.bag = randomForest(class ~ ., data=train, mtry=41, importance=TRUE))
```

```
##
## Call:
##  randomForest(formula = class ~ ., data = train, mtry = 41, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 41
##
##         OOB estimate of  error rate: 13.45%
## Confusion matrix:
##     NRB  RB class.error
## NRB 334  24  0.06703911
## RB   47 123  0.27647059
```

```
plot(r.bag, main="Bagging Error rates")      # error rates
legend("topright", leg=colnames(r.bag$err.rate), lty=1:3, col=1:3)
```



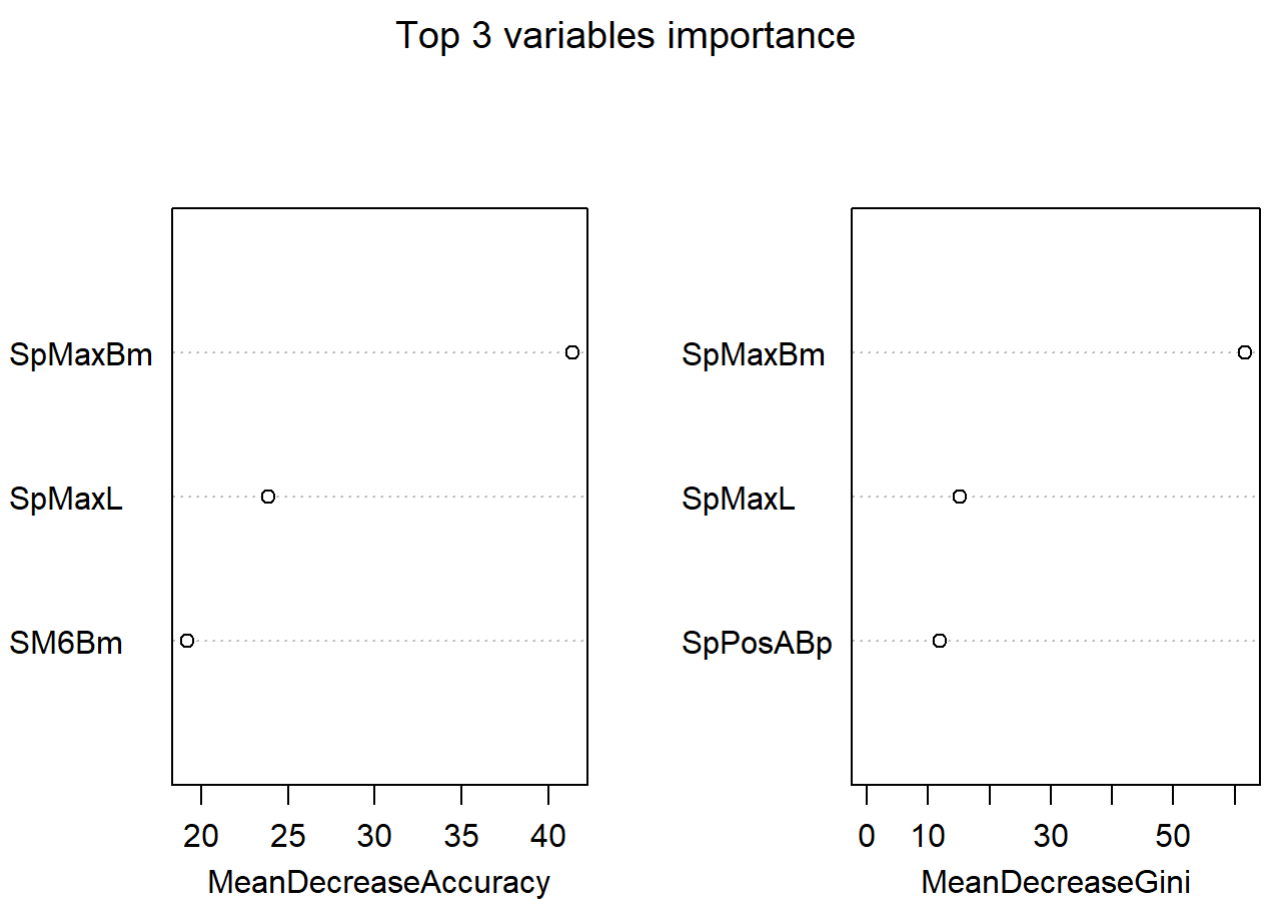

```
### Mean decrease Gini
round(importance(r.bag),2) # show importance of variables - higher values mean more important
```

```
##            NRB    RB MeanDecreaseAccuracy MeanDecreaseGini
## SpMaxL    8.35 23.33                23.86            15.15
## JDze      8.77  1.65                 8.66             5.97
## nHM       0.39  9.07                 9.79             1.05
## F01NN     0.54  1.60                 1.88             0.09
## F04CN    -1.96  8.64                 8.13             1.46
## NssssC    7.68  9.31                10.53             3.38
## nCb       4.08  3.06                 5.42             0.85
## C         7.05 -0.67                 6.07             5.23
## nCp       3.33 11.82                10.96             3.28
## nO        4.40  5.49                 7.77             3.33
## F03CN     3.89 10.44                10.36             3.00
## SdssC     9.86 11.09                14.52             8.39
## HyWiBm    8.61  4.51                11.07             4.32
## LOC       4.44 10.18                11.64             6.65
## SM6L      8.05  7.91                12.60             4.77
## F03CO    11.58  9.36                13.18             5.30
## Me        9.88  6.77                13.06             8.00
## Mi        9.70  0.31                 9.26             5.94
## nNN       0.00  0.00                 0.00             0.04
## nArNO2    2.58  2.55                 3.09             0.14
## nCRX3     1.00  1.00                 1.00             0.01
## SpPosABp 16.16  5.41                17.62            11.96
## nCIR      0.86  5.36                 5.58             0.61
## B01CBr    2.47  0.00                 2.50             0.08
## B03CCl   -0.05  2.05                 1.23             0.24
## N073     -1.00  0.00                -1.00             0.01
## SpMaxA    7.66  8.94                13.34             6.39
## Psii1d   11.21 -2.99                 9.14             4.89
## B04CBr   -3.75 -0.87                -3.96             0.12
## SdO       5.09  6.79                 9.05             3.40
## TI2L      6.81  5.11                 8.50             4.74
## nCrt      5.24  8.48                 8.86             1.62
## C026      3.52  2.47                 4.18             1.01
## F02CN     4.31 14.62                13.66             6.94
## nHDon     1.58  0.92                 1.75             1.40
## SpMaxBm  30.48 25.77                41.38            61.68
## PsiiA    13.48  8.76                16.06            11.40
## nN       11.16 14.07                15.61             7.55
## SM6Bm    17.52  6.94                19.22            10.43
## nArCOOR  11.53 18.15                19.03             8.45
## nX       -0.25  9.62                 9.76             1.19
```

```
sort(importance(r.bag)[,4],decreasing =TRUE) # Gini index ranking, max -> min important variable
s
```

```
##      SpMaxBm       SpMaxL     SpPosABp        PsiiA        SM6Bm      nArCOOR
## 61.680985612 15.148565258 11.959512038 11.395871186 10.433418194  8.448168365
##        SdssC           Me           nN        F02CN          LOC       SpMaxA
##  8.390369740  8.000780939  7.545341927  6.943576476  6.647890360  6.394527205
##         JDze           Mi        F03CO            C       Psii1d         SM6L
##  5.967641782  5.937843088  5.295158442  5.228120736  4.890491922  4.766519425
##         TI2L       HyWiBm          SdO       NssssC           nO          nCp
##  4.743215356  4.316582028  3.401054025  3.384234052  3.328450287  3.282389808
##        F03CN         nCrt        F04CN        nHDon           nX          nHM
##  2.995905269  1.615628233  1.462584684  1.402257402  1.190961236  1.045300914
##         C026          nCb         nCIR       B03Cl1       nArNO2       B04CBr
##  1.005638861  0.854898683  0.606116445  0.239236157  0.144502021  0.124249887
##        F01NN       B01CBr          nNN         N073        nCRX3
##  0.087566202  0.083739880  0.035839514  0.011333333  0.007108782
```

```
varImpPlot(r.bag,sort=T,n.var = 3,main = "Top 3 variables importance") # importance plot
```



7, Compute both the training and test errors of this Bagging predictor.

Because model is fitted with training set, training error is reduced to 0, while the test error is similar to the OOB estimate. The Bagging does not help much with prediction.

```
set.seed(289)
r.bag = randomForest(class ~ ., data=train, mtry=41, importance=TRUE)

table(train$class,predict(r.bag,train)) ## confusion table train
```

```
##
##       NRB  RB
##   NRB 358   0
##   RB    0 170
```

```
table(test$class,predict(r.bag,test)) ## confusion table test
```

```
##
##       NRB  RB
##   NRB 316  25
##   RB   48 138
```

```
##### training & test classification errors
errors(r.bag,predict,train,test) ##
```

```
## training.error     test.error
##      0.0000000      0.1366224
```

Random Forests

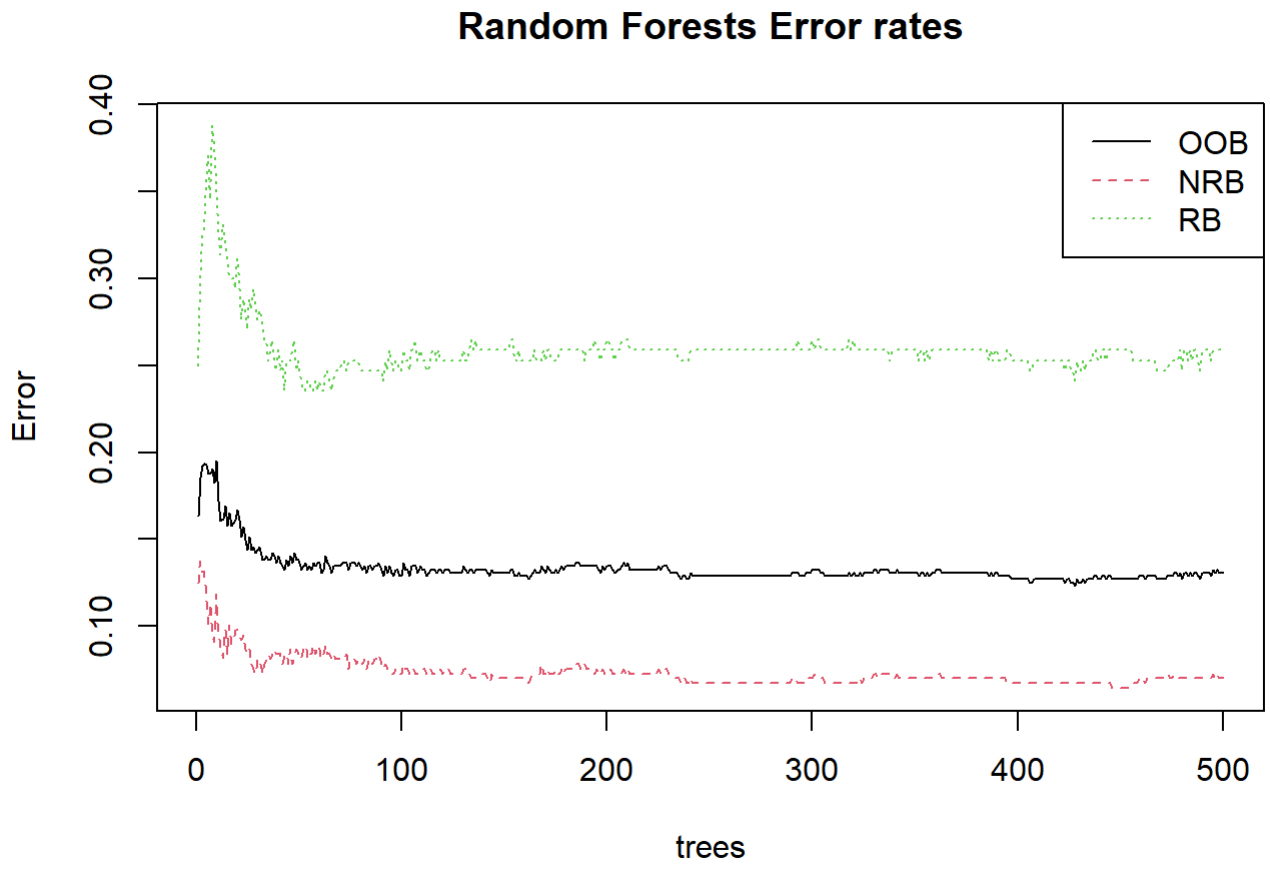
8, Produce a Random Forest model with 500 trees constructed.

For random forest, we take m = sqrt(number of variables), sqrt(41)~6. The three most important variables are SpMaxL,SpMaxBm and SM6Bm in terms of mean decrease accuracy-the deviance. This OOB of Random Forest is slightly better than bagging method.

```
set.seed(289)
(r.RF = randomForest(class ~ ., data=train, mtry=6, importance=TRUE))
```

```
##
## Call:
##  randomForest(formula = class ~ ., data = train, mtry = 6, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 13.07%
## Confusion matrix:
##     NRB  RB class.error
## NRB 333  25   0.0698324
## RB   44 126   0.2588235
```

```
plot(r.RF, main="Random Forests Error rates")      # error rates
legend("topright", leg=colnames(r.RF$err.rate), lty=1:3, col=1:3)
```



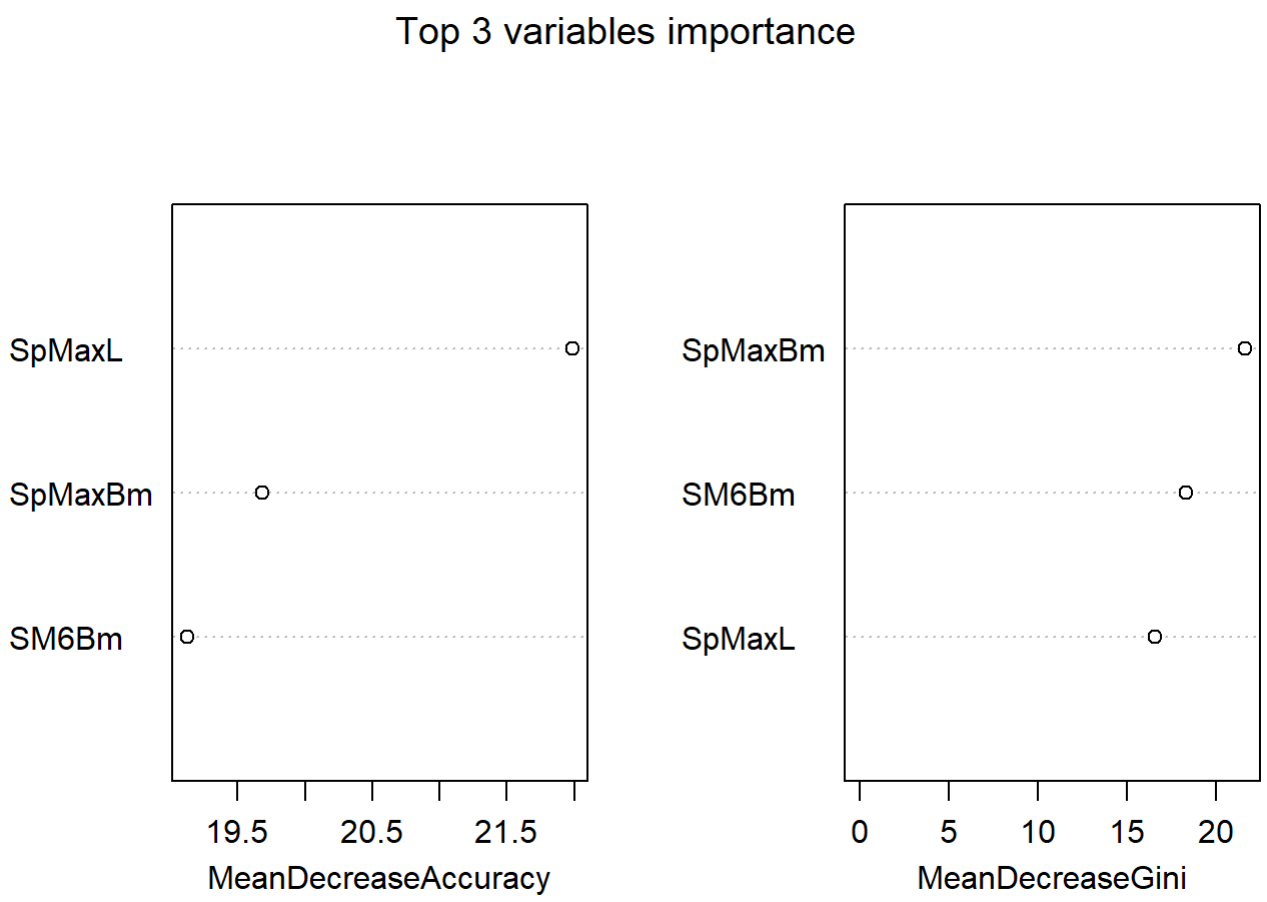

```
### importance of variables matrix
round(importance(r.RF),2)
```

```
##            NRB    RB MeanDecreaseAccuracy MeanDecreaseGini
## SpMaxL   13.03 18.62                21.99            16.55
## JDze      8.36  3.65                 9.06             6.36
## nHM       7.05  8.58                 9.97             3.62
## F01NN    -0.75  1.03                -0.02             0.27
## F04CN     5.55  9.68                10.28             2.92
## NssssC    8.13  9.51                10.64             3.16
## nCb       6.48  6.05                 9.02             3.98
## C         7.26  3.86                 8.75             6.31
## nCp       3.58  7.27                 7.76             3.55
## nO        9.76 12.18                15.07             6.89
## F03CN     6.10 11.93                13.20             4.78
## SdssC     8.22  9.98                12.06             7.56
## HyWiBm    9.99  8.01                13.00             8.48
## LOC       6.36  8.83                10.82             7.05
## SM6L     10.11 10.42                15.81             8.40
## F03CO    10.21 10.00                14.31             6.20
## Me        9.39  7.55                12.81             7.10
## Mi        7.98  3.82                 9.40             7.06
## nNN       1.74  0.00                 1.74             0.06
## nArNO2    3.52  2.64                 3.84             0.30
## nCRX3     0.11  1.00                 0.65             0.03
## SpPosABp 13.72  9.51                16.61            14.42
## nCIR      3.78  5.43                 6.86             2.20
## B01CBr   -0.77 -0.80                -1.00             0.14
## B03CCl    2.89  3.88                 4.34             0.62
## N073     -0.43 -1.41                -1.19             0.06
## SpMaxA   10.80 11.48                15.36            11.76
## Psii1d    7.59  1.87                 7.57             4.68
## B04CBr   -0.45 -0.73                -0.58             0.10
## SdO       8.80  7.35                10.97             5.78
## TI2L      6.69  7.80                10.79             6.69
## nCrt      3.69  3.97                 4.91             1.22
## C026      7.53  3.92                 8.50             3.49
## F02CN     5.84 12.83                13.13             5.92
## nHDon     1.68  2.55                 3.38             2.17
## SpMaxBm  15.84 13.33                19.68            21.61
## PsiiA    10.16  8.85                13.70             8.15
## nN        9.23 12.22                13.79             6.07
## SM6Bm    14.60 13.67                19.13            18.32
## nArCOOR   6.35  9.86                10.56             2.95
## nX        5.32  7.54                 8.60             2.65
```

```
### ranking of mean decrease accuracy (deviance) from max to min
sort(round(importance(r.RF)[,3],2),decreasing = TRUE)
```

```
##   SpMaxL  SpMaxBm    SM6Bm SpPosABp     SM6L   SpMaxA       nO    F03CO
##    21.99    19.68    19.13    16.61    15.81    15.36    15.07    14.31
##       nN    PsiiA    F03CN    F02CN   HyWiBm       Me    SdssC      SdO
##    13.79    13.70    13.20    13.13    13.00    12.81    12.06    10.97
##      LOC     TI2L   NssssC  nArCOOR    F04CN      nHM       Mi     JDze
##    10.82    10.79    10.64    10.56    10.28     9.97     9.40     9.06
##      nCb        C       nX     C026      nCp   Psii1d     nCIR     nCrt
##     9.02     8.75     8.60     8.50     7.76     7.57     6.86     4.91
##   B03CCl   nArNO2    nHDon      nNN    nCRX3    F01NN   B04CBr   B01CBr
##     4.34     3.84     3.38     1.74     0.65    -0.02    -0.58    -1.00
##     N073
##    -1.19
```

```
### variable importance plot
varImpPlot(r.RF,sort=T,n.var = 3,main = "Top 3 variables importance") # importance plot
```



9, Compute both the training and test errors of this Random Forest predictor.

The test error is similar to the OOB estimate 13.4%. The tweak used by Random Forest does not help prediction here.

```
### errors of train and test
errors(r.RF, predict,train,test)
```

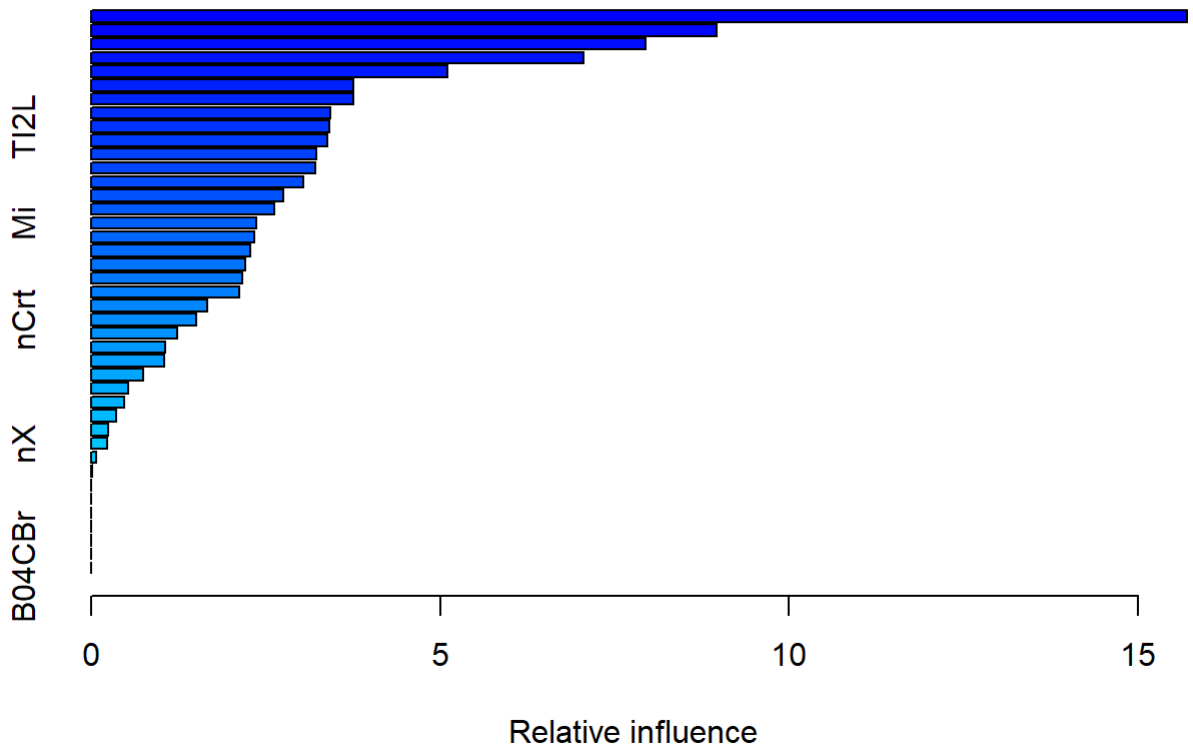
```
## training.error     test.error
##      0.0000000      0.1574953
```

Boosting

10, Produce a Boosting model, with 500 trees constructed.

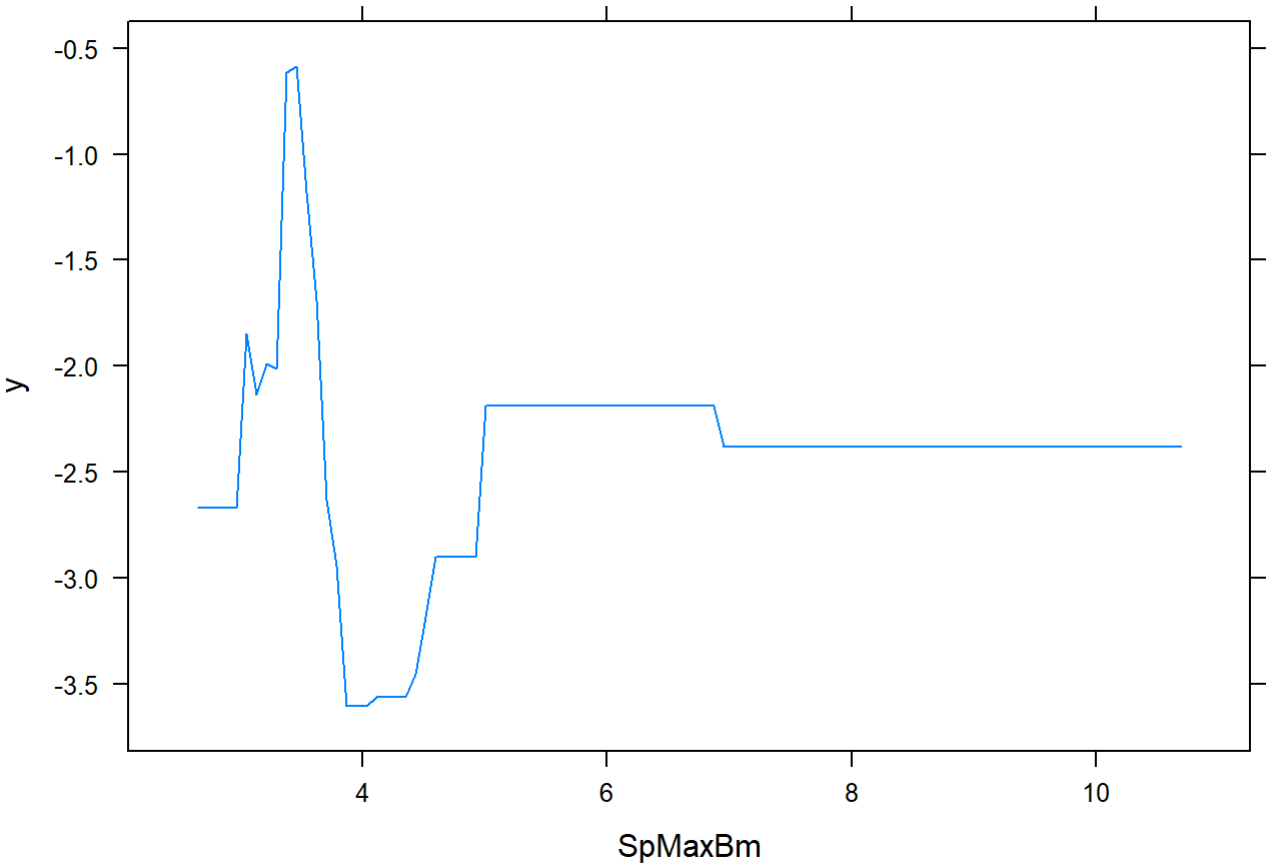
The three most important variables are SpMaxBm,SpPosABp and SM6Bm according to relative influence value (rel.inf) .

```
set.seed(289)
# use distribution="bernoulli" for a two-class classification problem
train2 = cbind(train, class2=as.integer(train$class)-1) # convert class to integer class
test2 = cbind(test, class2=as.integer(test$class)-1)    # convert class to integer class 2
r.boo = gbm(class2 ~ . - class, data=train2, distribution="bernoulli", n.trees=500, interaction.
depth=3)
summary(r.boo)
```



```
##               var      rel.inf
## SpMaxBm   SpMaxBm 15.696686603
## SpPosABp SpPosABp  8.959091051
## SpMaxL     SpMaxL  7.944346203
## SM6Bm       SM6Bm  7.060195664
## PsiiA       PsiiA  5.105365920
## nN             nN  3.758154886
## C               C  3.756462023
## F02CN       F02CN  3.421308727
## TI2L         TI2L  3.406940535
## SdssC       SdssC  3.381051656
## HyWiBm     HyWiBm  3.228784925
## JDze         JDze  3.209986617
## SM6L         SM6L  3.036339030
## Me             Me  2.752224411
## Psii1d     Psii1d  2.623964969
## Mi             Mi  2.364063499
## F03CO       F03CO  2.333888290
## SpMaxA     SpMaxA  2.286668931
## nO             nO  2.211646432
## LOC           LOC  2.165157794
## SdO           SdO  2.118978679
## NssssC     NssssC  1.668535061
## nCrt         nCrt  1.508725206
## nArCOOR   nArCOOR  1.240192801
## F03CN       F03CN  1.055786379
## C026         C026  1.050270582
## nHDon       nHDon  0.753652774
## nCp           nCp  0.531574980
## nCb           nCb  0.470227089
## nCIR         nCIR  0.353771746
## nHM           nHM  0.239405063
## nX             nX  0.224196521
## F04CN       F04CN  0.072838972
## B01CBr     B01CBr  0.009515978
## F01NN       F01NN  0.000000000
## nNN           nNN  0.000000000
## nArNO2     nArNO2  0.000000000
## nCRX3       nCRX3  0.000000000
## B03CCl     B03CCl  0.000000000
## N073         N073  0.000000000
## B04CBr     B04CBr  0.000000000
```

```
plot(r.boo, "SpMaxBm")## how SpMaxBm changes would affect prediction of y values
```



11, Compute both the training and test errors of this Boosting predictor.

The test error 14.9% is higher than the OOB estimate 13.4% with random forest. The boosting does not help with the prediction for this dataset.(T)The test error is 0.142, an improvement from the pruned tree.

```
# new function to calculate yhat based on probablity and errors based on numeric class2
fhat.boo = function(r,data) {
  p.boo = predict(r,data,type="response")
  yhat.boo = as.integer(p.boo>0.5)
}
errors.boo<- function(r,f,train,test) {
  train_class = f(r, train)
  test_class = f(r,test)
  c(training.error=mean(train$class2 != train_class), test.error=mean(test$class2 != test_clas
s))
}

#### training / test error

errors.boo(r.boo,fhat.boo,train2,test2)
```

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

```
## training.error     test.error
##       0.000000       0.142315
```

12, Demonstrate that Boosting can overfit.

100-2000 number of trees are attempted in test, the resulting train / test errors are printed. The result shows 500-600 trees have smallest test errors. Number >1000 can cause over-fitting.

```
trees = 2^(0:15)
test.error=rep(0,length(trees))
train.error=rep(0,length(trees))
for (i in 1:length(trees)){
  set.seed(189)
  r.over<- gbm(class2 ~ . - class, data=train2, distribution="bernoulli", n.trees=trees[i],inter
action.depth=3)
  ## train2 prediction
  p.boo1 = predict(r.over,train2,type="response")
  yhat.boo1 = as.integer(p.boo1>0.5)
  ## test 2 prediction
  p.boo2 = predict(r.over,test2,type="response")
  yhat.boo2 = as.integer(p.boo2>0.5)
  ## train / test errors
  train.error[i]<-mean(train2$class2 != yhat.boo1)
  test.error[i]<-mean(test2$class2 != yhat.boo2)
}
```

```
## Using 1 trees...
##
## Using 1 trees...
```

```
## Using 2 trees...
##
## Using 2 trees...
```

```
## Using 4 trees...
##
## Using 4 trees...
```

```
## Using 8 trees...
##
## Using 8 trees...
```

```
## Using 16 trees...
##
## Using 16 trees...
```

```
## Using 32 trees...
##
## Using 32 trees...
```

```
## Using 64 trees...
##
## Using 64 trees...
```

```
## Using 128 trees...
##
## Using 128 trees...
```

```
## Using 256 trees...
##
## Using 256 trees...
```

```
## Using 512 trees...
##
## Using 512 trees...
```

```
## Using 1024 trees...
##
## Using 1024 trees...
```

```
## Using 2048 trees...
##
## Using 2048 trees...
```

```
## Using 4096 trees...
##
## Using 4096 trees...
```

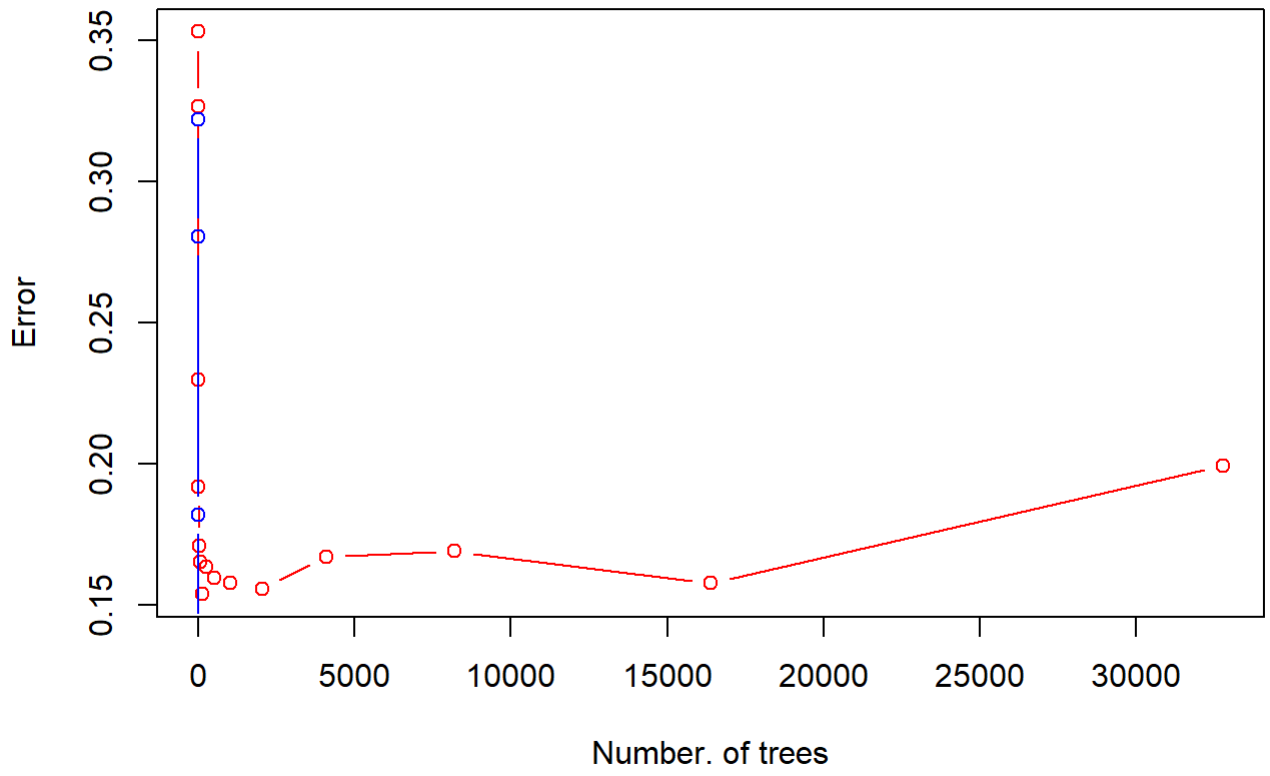
```
## Using 8192 trees...
##
## Using 8192 trees...
```

```
## Using 16384 trees...
##
## Using 16384 trees...
```

```
## Using 32768 trees...
##
## Using 32768 trees...
```

```
### GBM train / test errors of trees 100 - 1000
rgbm<-t(rbind(trees=trees,train.error = train.error,test.error = test.error))

### plot tree & test / train errors
plot(rgbm[,1],rgbm[,3],type = "b",col="red",xlab="Number. of trees",ylab="Error")#,log = "x")
lines(rgbm[,1],rgbm[,2],type = "b", col="blue")
```



Summary

In this class, we learn different tree-based classification/regression methods, including unpruned tree, pruned tree, bagging, random forest and GBM. We use 50% randomly generated orginal data as training data to train the model and test model classfication errors with 50% test dataset. Because the training data is used to generate the model, the test error would reflect more the accuracy of the fitted model. The result shows that the pruned tree models seleted by either Gini index or variance do not reduce the test errors, both ~ 19% test errors. About bagging, the estimated OOB of error rate is 14.04%, the real testing error is 14.4%. For random forest, OOB estimate is 13.5% , the real test error is 14.4%. For GBM, we teseted number of trees from 100- 1000, the lowest test errors have 500-600 trees with 14.7% error rate. Trees over 600 can cause over-fitting of model. Sofar, The bagging / random forest are probably the best methods according to the result of test errors.