lab 5

Awet Tsegay

Lab 5: Ensemble Methods

The topics that are covered in this lab are:

- Object Oriented Programming: Class and Objects in R
- Sampling without Replacement
- Random Forest
- Boosting
- The size of bootstrap samples

1. Object Oriented Programming

To retrieve the class of an object, use the class function.

```
a <- 2021
b <- FALSE
c <- "Data Analysis"
class(a)
```

```
## [1] "numeric"
```

```
class(b)
```

```
## [1] "logical"
```

```
class(c)
```

```
## [1] "character"
```

We can declare our own class using **setClass** function. You need to specify two arguments:

- the **name** of the class, and
- **slots**, which is a list of fields.

For each field, you need to specify the class it belongs to. Then you can create an object under the class using the new function.

```
setClass( "person", slots=list( name="character", age="numeric", gender="character", height="numeric"))
person1 <- new("person", name="Sheldon",age=28,gender="M",height=188)
```

To retrieve a field of an object, use @.

```
person1@age
```

```
## [1] 28
```

We can write functions that take objects under our own class as arguments.

```
is.adult <- function(x){
  if (x@age >= 18){
    print(paste(x@name,"is an adult."))
    } else {
      print(paste(x@name,"is not an adult."))
    }
}
is.adult(person1)
```

```
## [1] "Sheldon is an adult."
```

```
compare.age <- function(x1,x2){
  if (x1@age == x2@age){
    print( paste(x1@name,"and",x2@name,"are of the same age."))
  } else if (x1@age > x2@age){
    print( paste(x1@name," is older than ",x2@name,".",sep="" ))
  } else {
    print( paste(x2@name," is older than",x1@name,".", sep=""))
  }
}

person2 <- new("person",name="Amy",age=30,gender="F",height=173)
compare.age(person1,person2)
```

```
## [1] "Amy is older thanSheldon."
```

```
compare.age(person2,person2)
```

```
## [1] "Amy and Amy are of the same age."
```

1.1. Decision Tree

We focus on regression decision trees;

- the implementation of classification decision trees is similar.

Ready to declare the node class.

```
setClass("node", slots=list(is.leaf="logical", prediction="numeric",attr="character", split="numeric", left.node=
"node", right.node="node" ))
```

```
## Warning: undefined slot classes in definition of "node": left.node(class
## "node"), right.node(class "node")
```

There is a warning message due to recursive use of the node class. **To avoid any trouble, re-run the above code once more.**

To construct the decision tree,

- we first construct the leaves,
- followed by the two internal nodes in the middle of the tree,
- and lastly the root.

```
#leaf00 <- new("node", is.leaf=TRUE, prediction=5)
#leaf00
```

```
#leaf01 <- new("node",is.leaf=TRUE,prediction=15)
#leaf01
```

```
#leaf10 <- new("node",is.leaf=TRUE,prediction=28)
#leaf10
```

```
#leaf11 <- new("node",is.leaf=TRUE,prediction=40)
#leaf11
```

```
#in.left <- new("node",is.leaf=FALSE,attr="tax",split=200,left.node=leaf00,right.node=leaf01)
#in.left
```

```
#in.right <- new("node",is.leaf=FALSE,attr="black",split=100, left.node=leaf10,right.node=leaf11)
#in.right
```

```
#dt.root <- new("node",is.leaf=FALSE,attr="age",split=40, left.node=in.left,right.node=in.right)
#dt.root
```

```
#dt.root@left.node@right.node@prediction
```

Next, we use this decision tree to make predictions with the Boston dataset.

For the convenience to make prediction for any observation in the dataset, - we implement the dt.walk function.

The idea of dt.walk is very simple. - If the current node is a leaf, - then retrieve the prediction value of the node. - Otherwise, decide whether to move to the child node on the left or on the right.

```
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 4.0.4
```

```
names(Boston)
```

```
##  [1] "crim"    "zn"      "indus"   "chas"    "nox"     "rm"      "age"
##  [8] "dis"     "rad"     "tax"     "ptratio" "black"   "lstat"   "medv"
```

```
dt.walk <- function(node.in.dt,observation){
  if (node.in.dt@is.leaf){
    return(node.in.dt@prediction)
  } else if (observation[,node.in.dt@attr] <= node.in.dt@split){
    return(dt.walk(node.in.dt@left.node,observation))
  } else {
    return(dt.walk(node.in.dt@right.node,observation))
  }
}
```

1.2 Sampling Attributes in Random Forests

Recall that for a regression problem - with p attributes, - we sample p/3 attributes at each node (if p/3 is not an integer, round up).

The code below presents an example on how to sample attributes.

```
names(Boston)
```

```
##  [1] "crim"    "zn"      "indus"   "chas"    "nox"     "rm"      "age"
##  [8] "dis"     "rad"     "tax"     "ptratio" "black"   "lstat"   "medv"
```

```
p <- ncol(Boston) - 1
attributes <- names(Boston)[1:p]
attributes
```

```
##  [1] "crim"    "zn"      "indus"   "chas"    "nox"     "rm"      "age"
##  [8] "dis"     "rad"     "tax"     "ptratio" "black"   "lstat"
```

```
mtry <- ceiling(p/3)
sample(attributes,mtry,replace=FALSE)
```

```
## [1] "ptratio" "dis"     "black"   "indus"   "chas"
```

2. Bagging and Random Forests

We apply random forest method to the Boston dataset using the randomForest package in R.

In randomForest function, there are two crucial parameters. - mtry is the number of attributes sampled at each node. - ntree is the number of decision trees to be generated

First, we set mtry to be the number of attributes in the Boston dataset. As all attributes are used, there is no sampling of attributes going on.

In this case, the result is same as that of using the bagging method.

```
library(randomForest)
```

```
## randomForest 4.6-14
```

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

```
library(MASS)
set.seed(12)
p <- ncol(Boston)-1
train <- sample(nrow(Boston),2*nrow(Boston)/3,replace=FALSE)
rf.bag <- randomForest(medv~.,data=Boston,subset=train,mtry=p,ntree=10)
rf.bag
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = Boston, mtry = p, ntree = 10,      subset = train)
##                Type of random forest: regression
##                      Number of trees: 10
## No. of variables tried at each split: 13
##
##           Mean of squared residuals: 17.58238
##                     % Var explained: 79.04
```

We compute the test MSE as below.

```
yhat <- predict(rf.bag,newdata=Boston[-train,])
boston.test <- Boston[-train,"medv"]
mean((yhat-boston.test)^2)
```

```
## [1] 17.85341
```

Next, we use the real random forest, which should use p/3 random attributes (if p/3 is not an integer, round up) when building a random forest of regression trees.

When building a random forest of classification trees, it should use √p attributes instead.

```
rf.tree <- randomForest(medv~.,data=Boston,subset=train,
mtry=ceiling(p/3),ntree=10)
yhat.tree <- predict(rf.tree,newdata=Boston[-train,])
mean( (yhat.tree - boston.test)^2 )
```

```
## [1] 14.56833
```

We want to see how the number of trees affect the test MSE.

we use for loop to compute the test MSE when the numbers of trees are 10, 20, 30, . . . , 400.

```
for (i in 1:40){
  rf.tree <- randomForest(medv~.,data=Boston,subset=train,mtry=ceiling(p/3),ntree=i*10)
  yhat.tree <- predict(rf.tree,newdata=Boston[-train,])
  print(paste("The test MSE when ntree=" ,i*10 , "is", mean((yhat.tree-boston.test)^2),"." , sep = " " ) )
}
```

```
## [1] "The test MSE when ntree= 10 is 16.0179912587114 ."
## [1] "The test MSE when ntree= 20 is 14.7542344931051 ."
## [1] "The test MSE when ntree= 30 is 14.642037759002 ."
## [1] "The test MSE when ntree= 40 is 13.2051809210968 ."
## [1] "The test MSE when ntree= 50 is 13.1370386707152 ."
## [1] "The test MSE when ntree= 60 is 12.9570536108065 ."
## [1] "The test MSE when ntree= 70 is 12.9839023377645 ."
## [1] "The test MSE when ntree= 80 is 13.77853543814 ."
## [1] "The test MSE when ntree= 90 is 12.6963273068626 ."
## [1] "The test MSE when ntree= 100 is 13.2046653978386 ."
## [1] "The test MSE when ntree= 110 is 12.8242560517629 ."
## [1] "The test MSE when ntree= 120 is 14.0970653600715 ."
## [1] "The test MSE when ntree= 130 is 12.7580075951174 ."
## [1] "The test MSE when ntree= 140 is 13.0862823507115 ."
## [1] "The test MSE when ntree= 150 is 12.476561545325 ."
## [1] "The test MSE when ntree= 160 is 13.5429042787023 ."
## [1] "The test MSE when ntree= 170 is 13.6166498024094 ."
## [1] "The test MSE when ntree= 180 is 13.4687267778139 ."
## [1] "The test MSE when ntree= 190 is 13.903696253376 ."
## [1] "The test MSE when ntree= 200 is 13.0866751341519 ."
## [1] "The test MSE when ntree= 210 is 13.7401765081853 ."
## [1] "The test MSE when ntree= 220 is 13.3525145363638 ."
## [1] "The test MSE when ntree= 230 is 12.8387103640454 ."
## [1] "The test MSE when ntree= 240 is 12.903428760566 ."
## [1] "The test MSE when ntree= 250 is 13.3067700715894 ."
## [1] "The test MSE when ntree= 260 is 13.628997195369 ."
## [1] "The test MSE when ntree= 270 is 13.1037107987819 ."
## [1] "The test MSE when ntree= 280 is 13.0383342258046 ."
## [1] "The test MSE when ntree= 290 is 13.5249682631026 ."
## [1] "The test MSE when ntree= 300 is 12.5197354499444 ."
## [1] "The test MSE when ntree= 310 is 12.9732203516432 ."
## [1] "The test MSE when ntree= 320 is 13.3550277399826 ."
## [1] "The test MSE when ntree= 330 is 13.2952537944802 ."
## [1] "The test MSE when ntree= 340 is 13.1309882125597 ."
## [1] "The test MSE when ntree= 350 is 13.0915491244749 ."
## [1] "The test MSE when ntree= 360 is 13.0132042077361 ."
## [1] "The test MSE when ntree= 370 is 13.463895413187 ."
## [1] "The test MSE when ntree= 380 is 13.3166000013659 ."
## [1] "The test MSE when ntree= 390 is 13.4606752699057 ."
## [1] "The test MSE when ntree= 400 is 13.1228150595552 ."
```

Observe that the test MSE stabilizes when ntree is over 100.

3. Boosting

Here we use the gbm package, and within it the gbm() function, to fit boosted **regression trees** to the Boston dataset.

In the code below, we run gbm() with the option distribution="gaussian" since this is a regression problem;

if it were a binary classification problem, we would use distribution="bernoulli".

- The argument shrinkage=0.01 is the learning rate (same as η in the lecture,
- while n.trees=2000 indicates that we want 2000 trees.
- The option interaction.depth=4 limits the depth of each tree.

```
library(gbm)
```

```
## Loaded gbm 2.1.8
```

```
library(MASS)
set.seed(12)
train <- sample(nrow(Boston),2*nrow(Boston)/3,replace=FALSE)
boost.boston <-gbm(medv~.,data=Boston[train,],distribution="gaussian", shrinkage=0.01,n.trees=2000,interaction.de
pth=4)
boost.boston
```

```
## gbm(formula = medv ~ ., distribution = "gaussian", data = Boston[train,
##     ], n.trees = 2000, interaction.depth = 4, shrinkage = 0.01)
## A gradient boosted model with gaussian loss function.
## 2000 iterations were performed.
## There were 13 predictors of which 13 had non-zero influence.
```

We compute the test MSE as below. - (In predict, the argument n.trees=2000 is not necessary; - however, if you do not specify it, - it will prompt a message to notify you the number of trees being used, which can be annoying when you predict for multiple times. - So we specify n.trees to avoid the message.)

```
yhat.boost <- predict(boost.boston,newdata=Boston[-train,],n.trees=2000)
boston.test <- Boston[-train,"medv"]
mean((yhat.boost-boston.test)^2)
```

```
## [1] 12.47269
```

You can try different choices of learning rates to see how the test MSE varies.

- The code below computes the test MSE for η = 0.002, 0.004, . . . , 0.200,
- and plot the test MSE against η.

```
test.MSE = c()
for (i in 1:100){
  eta <- i/500.0
  boost.boston <- gbm(medv~.,data=Boston[train,], distribution="gaussian",shrinkage=eta, n.trees=2000,interactio
n.depth=4)
  yhat.boost <-predict(boost.boston,newdata=Boston[-train,])
  test.MSE <- c(test.MSE,mean((yhat.boost-boston.test)^2))
}
```

```
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
##
## Using 2000 trees...
```

```
eta.seq <- (1:100) / 500
```

```
plot(eta.seq , test.MSE)
```