

Random Forest based on the lecture of Edward Malthouse

Motivation

The random forest is an algorithm that consists on predicting the results using many random small trees to calculate the mean of their prediction for a regression problem, or the maximum count for a classification problem. It was first introduced in Breiman (2001).

Trees are unstable

Trees are easy to interpret but very unstable and depend highly on the training sample. Let us prove this statement with an example.

We will use two different random samples from our *mpg_new.csv* file for training and fit a tree for each sample. Are they very different? If the answer is yes, although the samples come from the same database, then we have proven their instability.

First we need to learn how to pick a random sample from a database in R. One option is to use function *sample*. For example, the code below chooses a sample of size 3 from the sequence 1, 2, 3, ..., 10 (without replacement).

```
set.seed(42)
sample(1:10, 3)
```

```
## [1] 1 5 10
```

We can use that feature to randomize dataset too. For example, the next chunk of code returns three random values of variable *Sepal.Length* from dataset *iris* because it is a vector.

```
set.seed(42)
sample(iris$Sepal.Length, 3)
```

```
## [1] 5.3 5.6 6.1
```

However, if we want to get 3 random rows of the whole dataset (not only of one variable), the following command does not work.

```
sample(iris, 3)
```

Instead, we have to get a random sample of the number of rows and use those. See the code below:

```
len <- NROW(iris)
iris[sample(1:len,3), ]
```

Any questions?

Exercise 1

Following the steps below, write your lines of code in the R-chunk below:

1. Create a vector with 1000 consecutive values with name *mydata*
2. Get three random samples from *mydata*
 - random sample *s1* with 500 values
 - random sample *s2* with 20 values
 - random sample *s3* with 1000 values
3. Obtain the third last values of each random sample and compare them. Are they the same? Why? Write your answer after the R-chunk.

#1

#2

#3

Answer 1:

Exercise 2

1. Upload file *mpg_new.csv*.
2. How many rows and columns does *mpg_news.csv* have? Figure out with R code and write the code in the chunk below.
3. Discover the variable names of your database. Write the code in the chunk below.
4. Select the relevant variables into a new dataset called *mpg2*. These variables are: “cylinders”, “displacement”, “horsepower”, “weight”, “acceleration”, “mpg” (response).
5. Select the first 70% of the values in *mpg2* to create the dataset *train* and the rest for data set *pred*

#1

#2

#3

#4

#5

6. From the *train* data set, create two samples of 100 values (without repetition) using function *sample()* to generate *train1* and *train2*.

#6

Below, we plot the miles per gallon *mpg* for Sample 1 (red dots) and Sample 2 (blue dots) that should be different in general. Note that the code below will not work without the previous chunk.

```
plot(train1$mpg, ylab = "Miles per gallon", col=2, pch=16)
points(train2$mpg, col=4, pch=16)
legend("topright", c("Sample 1", "Sample 2"), col=c(2, 4), pch=16)
```

Exercise 3

Obtain one tree for each of the samples and name them *mpg.rpart1* and *mpg.rpart2*, respectively. Write you code in the chunk below.

#(your code below here)

We plot the two trees that you have estimated and they are very different. Not even the top node is the same. Only because they were created with two samples of the **same** dataset. That is why we say that the tree method is very unstable because a small change in the input data produces a big change in the output data. Note that the code will not work without the previous chunk.

```
par(mfrow = c(1, 2))
rpart.plot(mpg.rpart1)
rpart.plot(mpg.rpart2)
par(mfrow = c(1,1))
```

This difference will affect their prediction too. In the figure below, we compare display graphically the real values of concentration of *mpg* from dataset *pred* (black dots) and their predictions with Sample 1 (red dots) and Sample 2 (blue dots).

```
plot(pred$mpg, ylab = "MPG", xlab="", pch = 16)
points(predict(mpg.rpart1, newdata = pred), col=2, pch=16)
points(predict(mpg.rpart2, newdata = pred), col=4, pch=16)
legend("topleft", c("True mpg", "Sample 1", "Sample2"), col=c(1,2, 4), pch=16, ncol=2)
```

Bagging

One of the key elements of the random forest algorithm is **bagging**. Bagging is a sampling technique, also called bootstrap aggregation, that consists on randomly picking several training samples with replacement and average the estimates of any given algorithm. The idea is that obtaining the average of many imperfect estimators will result in a good estimator.

Let us see first what we mean with random sample with replacement in the chunk below.

```
set.seed(42)
sample(1:10, replace = TRUE)
```

```
## [1] 1 5 1 9 10 4 2 10 1 8
```

In this case, the element in position 1 would have been chosen 3 times for my training sample, the element in position 10 is chosen 2 times and for example element 3 is not chosen. We could use that sample to find a tree and keep the results. Let us do another random pick of training sample with replacement:

```
sample(1:10, replace = TRUE)
```

```
## [1] 7 4 9 5 4 10 2 3 9 9
```

```
# Observations 1, 6, 8 omitted can go into testing sample
```

The bagging algorithm exploits the variation in the estimation of the model using different samples to produce a better estimate. For example, we could use bagging and estimating trees with the following algorithm.

1. Repeat the following B times:
 - Draw a sample of size n with replacement from the available data
 - Estimate a tree
2. Average the estimates from the B trees for the final estimate in a regression case, or take the largest count in the classification case

The main advantage of bagging is that because it picks many different samples of the training set, then the final model does not overfit the data and it is better for prediction/forecast of other values.

Simulation of bagged tree estimate

Let us use an example to motivate the functioning of the random forest. We generate a dataset with variables x and y .

- x is a random variable with a Uniform[0,1] distribution
- y is a random variable calculated like $y = \sin(x\pi/2) + \epsilon$ where ϵ is a standard normal random variable.

```
n = 100
# Generate a random Uniform [0,3] variable
x=runif(n)*3
# Create an error signal from a wave
y <- sin(x*pi/2) + rnorm(n)/4
mydata <- data.frame(x=x, y=y)
# The true signal is
grid <- seq(0, 3, by = 0.05)
f <- sin(grid*pi/2)
newdata <- data.frame(x= grid, y = f)
plot(x, y, pch=16, xlab = "x", ylab = "y")
lines(grid, f, lwd = 4)
```

Let us run a bagging + tree algorithm using 20 different samples:

```

plot(x, y, pch=16, xlab="x", ylab="y")
lines(grid, f, lwd = 4)
B <- 20
result <- numeric(length(grid))
for (i in 1:B)
{
  train <- sample(1:n, n, TRUE)
  fit <- rpart(y~x, data = mydata[train,])
  if(i <= 6)
  {# this plots only the first 6 trees prediction
    lines(grid, predict(fit, newdata = newdata), col= 4)
  }
  result <- result + predict(fit, newdata = newdata)
}
pred.20 <- result/B
lines(grid, pred.20, lwd=4, col=2)
legend("topright", c("f", "Individual tree", "Average tree"),
      col = c(1, 4, 2), lty = 1, lwd = 3)

```

The figure above shows that the red line kind of lies in the middle of the blue lines. When there are disagreement between individual trees, when this is averaged (red line) we get the right answer. When we have a unstable estimator, their average is right. The problem is when there is no instability among the individual trees. That is why having a bunch of estimates that are wrong, getting the average is a good thing provided some are lower and other upper the right thing.

Bagged tree estimate in the MPG case

Let us try bagging with our *train* data. We will take 100 samples of size 120 and obtain the final predicted values:

```
B = 100
mpg.pred <- numeric(NROW(pred))
for (i in 1:B)
{
  train2 <- train[sample(1:273, 100, replace = TRUE), ]
  mpg.rpart.bag <- rpart(mpg~., data=train2)
  #We add to the new prediction to previous values of prediction
  #so later to we can make the average to create the bagging prediction
  mpg.pred <- mpg.pred + predict(mpg.rpart.bag, newdata = pred)
}
#Divide by the number of values
mpg.bagging <- mpg.pred/B
```

Exercise 4

Compare the three prediction with MAE and RMSE, that is, the two trees prediction and the bagging prediction.

****Answer 4**:**

Random Forest

- The problem with simple bagging is that trees are sometime too similar, without variation, the concept of bagging is to create a new dataset for each tree.
- Random forest goes a little bit further. It introduces randomness in the process of building a tree. It does this by:

- + \sqrt{p} number of predictors for a classification problem and
- + $p/3$ number of predictors for a regression case.

```
library(randomForest)
mpg.forest <- randomForest(mpg ~ ., data = train)
mpg.forest
```

Fill the gaps in this sentence: This output shows that the problem is a regression problem, that we have used 500 small trees, each tree has ... predictors, the MSE is ... and the percentage of the variability in miles per gallon explained by the model is ...

Interpretation

The random forest is difficult to interpret because it has many trees, but we can figure out which predictors (variables, features) are the most important in the prediction. This is explained with what we call *importance* and can be obtained adding parameter *importance= TRUE* in your call to the *randomForest* function. The

higher the value the more importance, meaning that if we remove that predictor from our model then the purity of our trees decrease.

```
mpg.forest <- randomForest(mpg ~ ., data = train, importance= TRUE)
importance(mpg.forest)
```

The importance is shown visually:

```
varImpPlot(mpg.forest, main = "Variables Importance")
```

Homework

Watch the lecture in this [YouTube video](#)