MK476 Lab 3: Computational advertising, part 3

Before you begin

Make sure you have all required libraries installed. For this lab, you will need to install the following new packages:

```
install.packages(c("rpart.plot", "rpart"))
```

The problem

We will continue working with the Cogo Labs dataset that we have used in Labs 1 and 2. This lab will explore tree-based methods for prediction.

Preparing for the lab

Before we get started, let's load some useful libraries.

```
library(data.table)
library(ggplot2)
library(ggthemes)
library(scales)
library(rpart)
library(rpart.plot)

theme_set(theme_bw())
```

The data

Training dataset

The training dataset we will use comes from Lab 1; refer to the Lab 1 description for details. Let load the training data.

IMPORTANT: you will need to update the filename below to match the location of the dataset on your computer.

```
dd <- fread("../../data/cogo-train.tsv", stringsAsFactors = T)
```

Train and test datasets

Start by splitting your data into a train and a test set. Here's how to do it:

```
dd[, test:=0]
dd[sample(nrow(dd), 100000), test:=1] # take 100K random rows and stick them in the test set
# now split
```

```
dd.test <- dd[test==1]
dd.train <- dd[test==0]
```

The training data contains 188298 observations. This will slow down training. Let's take a random subsample, and then when we are happy with the tuning of our algorithms, we can increase the size of the training set further.

```
dd.train.sample.size <- 5000
dd.train.sample <- dd.train[sample(nrow(dd.train), dd.train.sample.size)]
```

Data preparation

Some ML algoritms (like linear regression) take a formula interface. Others take a matrix of responses x and a matrix of predictors y. Let's prepare these matrices now, for both the train and test datasets. We will suffix training-related data with `.train`, and testing-related data with `.test` to easily distinguish them.

```
# here's one simple formula -- it's up to your to add more predictors as you see fit
f1 <- as.formula(p_open ~ browser1 + browser2 + browser3)
```

Now, let's translate the data represented by the formula to a matrix

```
# the [, -1] means take all columns of the matrix except the first column,
# which is an intercept added by default
x1.train.sample <- model.matrix(f1, dd.train.sample)[, -1]

# and this the response
y.train <- dd.train$p_open
y.train.sample <- dd.train.sample$p_open
```

Notice that I have names the matrix `x1.train` to remember it's associated formula `f1`. Later, you may want to experiment with multiple formulas. Instead of overwriting `f1`, you may prefer to create `f2` and `x2.train`, etc.

We will now do the same for the test data, recalling the test data has no response variable.

```
dd.test[, p_open:=1] # hack so that the following line works
x1.test <- model.matrix(f1, dd.test)[, -1]
y.test <- dd.test$p_open
```

Let's predict

Regression tree

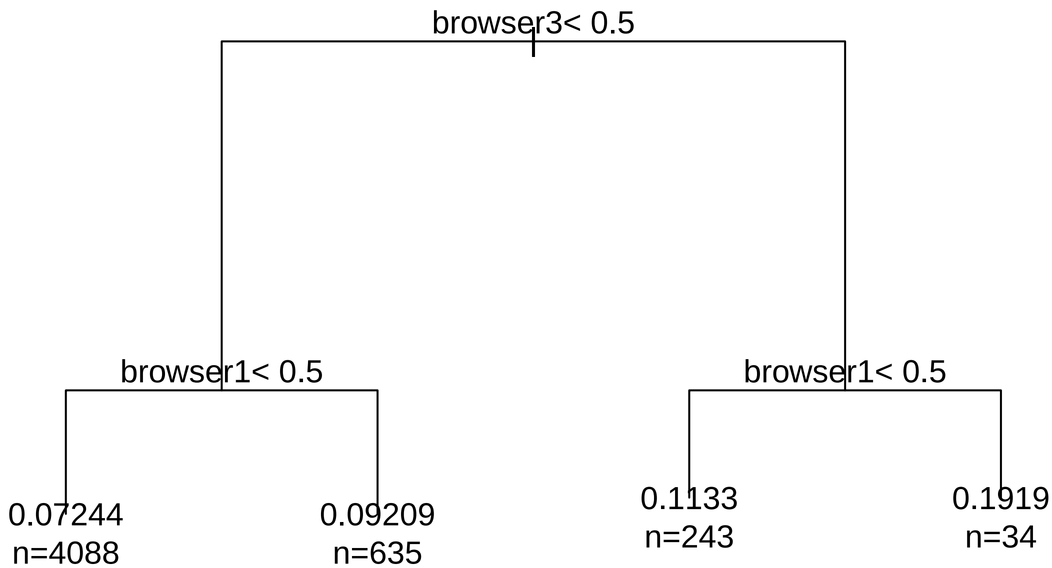
We will start with a straightforward regression tree.

```
fit.tree <- rpart(f1,
    dd.train.sample,
    control = rpart.control(cp = 0.001))
```

You can control the complexity of the tree using the `cp` parameter. Smaller values will give you more complex trees.

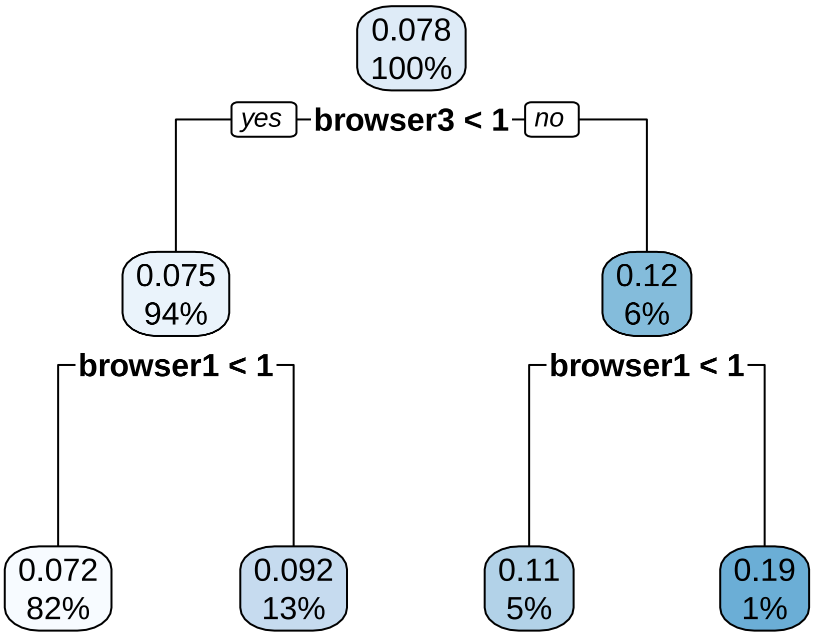
One of the advantages of trees is that they are simple enough to plot.

```r
par(xpd = TRUE)
plot(fit.tree, compress=TRUE)
text(fit.tree, use.n=TRUE)
```



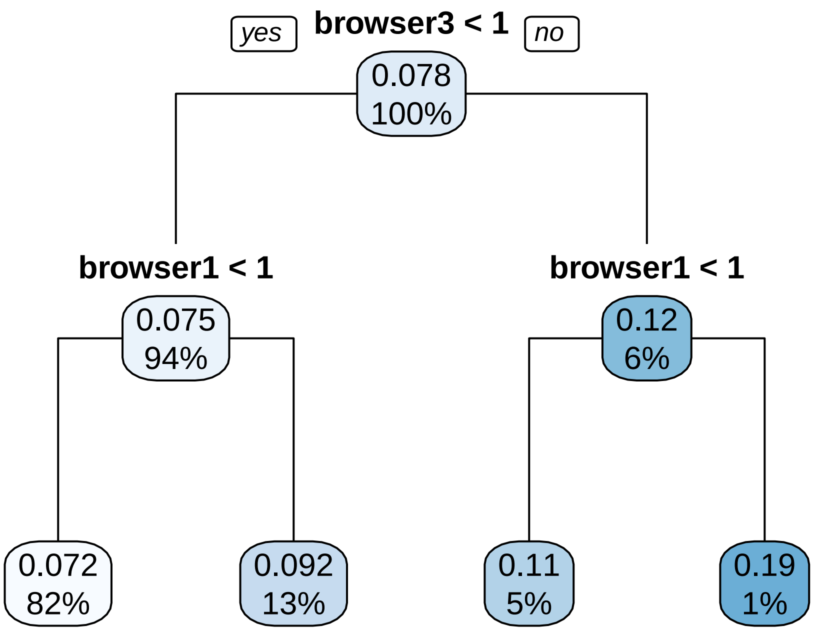
We can produce prettier plots using the `rpart.plot` function.

```r
rpart.plot(fit.tree)
```



The `rpart.plot` function accepts a numeric type argument that creates different styles of plots. For example:

```
rpart.plot(fit.tree, type = 1)
```



You can learn more about `rpart.plot` here: http://www.milbo.org/doc/prp.pdf

This a rather boring tree. Nevertheless, let's make some predictions and compute a train MSE.

```
yhat4.tree <- predict(fit.tree, dd.train.sample)
mse.tree <- mean((yhat4.tree - y.train.sample) ^ 2)
```