Classification trees with `rpart`

Kevin Stadler

November 24, 2014

```r
# install.packages("rpart")
library(rpart)
```

`rpart()` fits a regression or classification tree, depending on the type of the response variable – when it is a string or factor, it attempts to find the simplest model that predicts the observed categorical outcomes based on the predictor variables. Let's try to predict some string suffixes (i.e. a categorical outcome):

```r
d <- read.csv("rewrite.csv", stringsAsFactors=FALSE)

rpart(q_suffix ~ number + noun + verb, d, control=rpart.control(minsplit=2))
```

```
## n= 18
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 18 12 a (0.3333 0.3333 0.1667 0.1667)
##   2) number=sing 9  3 a (0.6667 0.0000 0.3333 0.0000)
##     4) noun=bird,duck 6  0 a (1.0000 0.0000 0.0000 0.0000) *
##     5) noun=croc 3  0 u (0.0000 0.0000 1.0000 0.0000) *
##   3) number=pl 9  3 ak (0.0000 0.6667 0.0000 0.3333)
##     6) noun=bird,duck 6  0 ak (0.0000 1.0000 0.0000 0.0000) *
##     7) noun=croc 3  0 uk (0.0000 0.0000 0.0000 1.0000) *
```

```r
rpart(v_suffix ~ number + noun + verb, d, control=rpart.control(minsplit=2))
```

```
## n= 18
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 18 12 an (0.33 0.22 0.17 0.11 0.17)
##    2) number=sing 9  3 an (0.67 0 0.33 0 0)
##      4) noun=bird,duck 6  0 an (1 0 0 0 0) *
##      5) noun=croc 3  0 en (0 0 1 0 0) *
##    3) number=pl 9  5 asp (0 0.44 0 0.22 0.33)
##      6) verb=V1,V2 6  2 asp (0 0.67 0 0.33 0)
##       12) noun=bird,duck 4  0 asp (0 1 0 0 0) *
##       13) noun=croc 2  0 esp (0 0 0 1 0) *
##      7) verb=V3 3  0 onk (0 0 0 0 1) *
```

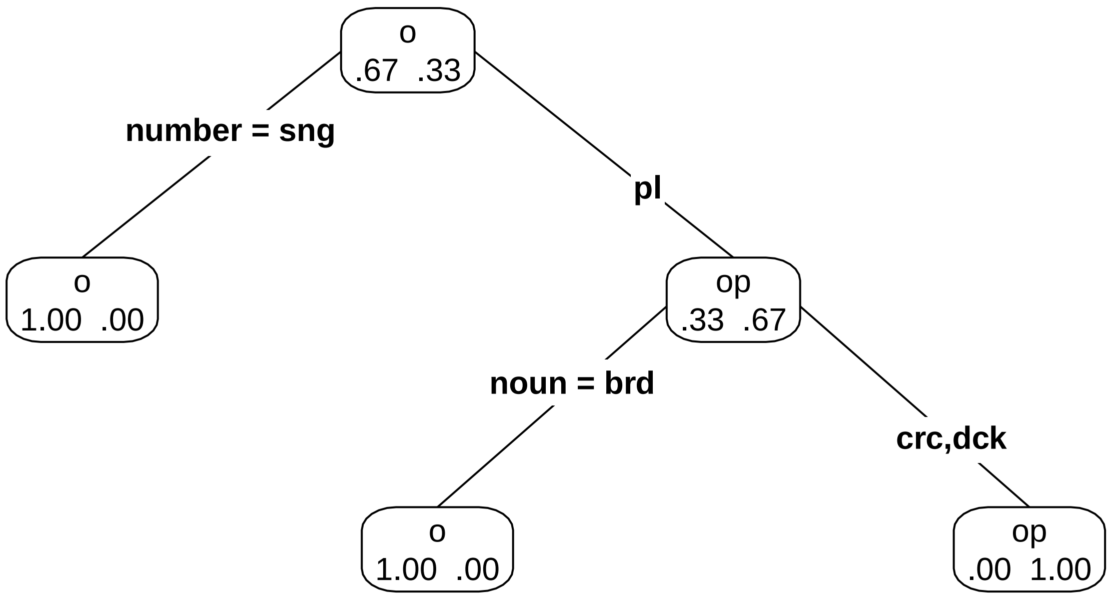
```r
rpart(n_suffix ~ number + noun + verb, d, control=rpart.control(minsplit=2))
```

```
## n= 18
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 18 6 o (0.6667 0.3333)
##   2) number=sing 9 0 o (1.0000 0.0000) *
##   3) number=pl 9 3 op (0.3333 0.6667)
##     6) noun=bird 3 0 o (1.0000 0.0000) *
##     7) noun=croc,duck 6 0 op (0.0000 1.0000) *
```

We can even draw the trees, simply by calling `plot()` on the resulting test objects. The default plots are not very beautiful, but the `rpart.plot` package allows for rather beautiful plots using `prp()` (check http://www.milbo.org/rpart-plot/prp.pdf for more plotting options).

```
t <- rpart(n_suffix ~ number + noun + verb, d, control=rpart.control(minsplit=2))

# install.packages("rpart.plot")
library(rpart.plot)
prp(t, type=4, extra=4)
```



Inconsistent data

Pool data together and look at the noun suffix tree (because it's the simplest): twice the amount of the same data results in the same tree as above:

```
d2 <- rbind(d,d)
rpart(n_suffix ~ number + noun + verb, d2, control=rpart.control(minsplit=2))
```

```
## n= 36
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 36 12 o (0.6667 0.3333)
##   2) number=sing 18  0 o (1.0000 0.0000) *
##   3) number=pl 18  6 op (0.3333 0.6667)
##     6) noun=bird 6  0 o (1.0000 0.0000) *
##     7) noun=croc,duck 12  0 op (0.0000 1.0000) *
```

Let's pretend data pooling creates heterogenous input, i.e. one participant marks one of the singulars differently in one of its items:

```
d2[1, "n_suffix"] <- "foo"

rpart(n_suffix ~ number + noun + verb, d2, control=rpart.control(minsplit=2))
```

```
## n= 36
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 36 13 o (0.02778 0.63889 0.33333)
##   2) number=sing 18  1 o (0.05556 0.94444 0.00000) *
##   3) number=pl 18  6 op (0.00000 0.33333 0.66667)
##     6) noun=bird 6  0 o (0.00000 1.00000 0.00000) *
##     7) noun=croc,duck 12  0 op (0.00000 0.00000 1.00000) *
```

It's still the same tree, but because the data is heterogeneous it's not possible to make a deterministic tree, i.e. the terminal node is probabilistic:

```
##    2) number=sing 18  1 o (0.05556 0.94444444 0.00000000) *
```

aka the default suffix is "o" but there is 1 exception to it (~5.6% of the 18 datapoints) that cannot be accounted for based on the predictors that are available.

System complexity vs. pooling complexity

What's kind of nice about these classification trees is that they selectively allow you to tease apart systematic complexity (from consistent but tedious exceptions of one slot exhibited by all speakers) and complexity arising from noise/pooling (i.e. inconsistent use of suffixes within one slot). If you're interested to know how much of the variability is actually from pooling you could compare the tree created from pooled data with one created with the same model but *including speaker identity as a predictor*:

```
# ain't got no data for this
rpart(n_suffix ~ number + noun + verb + SPEAKER, d, control=rpart.control(minsplit=2))
```

If you're interested in quantifying the complexity of the classification trees beyond their number of splits, terminal nodes and their depth, probz use the `summary()` command to find out stuff about the probability of passing through specific nodes, the explanatory 'work' carried out by specific splits in the tree, etc. Presumably some of these measures will be affected by the amount of data your tree is based on, while others won't be. This is the summary of the pure pooled data (contains every datapoint twice):

```
summary(rpart(n_suffix ~ number + noun + verb, rbind(d,d), control=rpart.control(minsplit=2)))
```

```
## Call:
## rpart(formula = n_suffix ~ number + noun + verb, data = rbind(d,
##     d), control = rpart.control(minsplit = 2))
##   n= 36
##
##     CP nsplit rel error xerror   xstd
## 1 0.50      0         1      1 0.2357
## 2 0.01      2         0      0 0.0000
##
## Variable importance
##   noun number
##     50     50
```

```
##
## Node number 1: 36 observations,    complexity param=0.5
##   predicted class=o   expected loss=0.3333  P(node) =1
##     class counts:    24    12
##    probabilities: 0.667 0.333
##   left son=2 (18 obs) right son=3 (18 obs)
##   Primary splits:
##       number splits as  RL,  improve=8, (0 missing)
##       noun   splits as  LRR, improve=4, (0 missing)
##
## Node number 2: 18 observations
##   predicted class=o   expected loss=0  P(node) =0.5
##     class counts:    18     0
##    probabilities: 1.000 0.000
##
## Node number 3: 18 observations,    complexity param=0.5
##   predicted class=op  expected loss=0.3333  P(node) =0.5
##     class counts:     6    12
##    probabilities: 0.333 0.667
##   left son=6 (6 obs) right son=7 (12 obs)
##   Primary splits:
##       noun splits as  LRR, improve=8, (0 missing)
##
## Node number 6: 6 observations
##   predicted class=o   expected loss=0  P(node) =0.1667
##     class counts:     6     0
##    probabilities: 1.000 0.000
##
## Node number 7: 12 observations
##   predicted class=op  expected loss=0  P(node) =0.3333
##     class counts:     0    12
##    probabilities: 0.000 1.000
```

Compare this to the summary of the tree generated for the data with the one fudged singular marker:

```r
summary(rpart(n_suffix ~ number + noun + verb, d2, control=rpart.control(minsplit=2)))
```

```
## Call:
## rpart(formula = n_suffix ~ number + noun + verb, data = d2, control = rpart.control(minsplit = 2)
##   n= 36
##
##       CP nsplit rel error  xerror    xstd
## 1 0.4615      0   1.00000 1.00000 0.22169
## 2 0.0100      2   0.07692 0.07692 0.07585
##
## Variable importance
##   noun number
##     52     48
##
## Node number 1: 36 observations,    complexity param=0.4615
##   predicted class=o   expected loss=0.3611  P(node) =1
##     class counts:     1    23    12
##    probabilities: 0.028 0.639 0.333
##   left son=2 (18 obs) right son=3 (18 obs)
##   Primary splits:
##       number splits as  RL,  improve=7.3890, (0 missing)
##       noun   splits as  LRR, improve=4.3610, (0 missing)
##       verb   splits as  LRR, improve=0.1111, (0 missing)
```

```
##
## Node number 2: 18 observations
##   predicted class=o   expected loss=0.05556  P(node) =0.5
##     class counts:     1    17     0
##    probabilities: 0.056 0.944 0.000
##
## Node number 3: 18 observations,    complexity param=0.4615
##   predicted class=op  expected loss=0.3333  P(node) =0.5
##     class counts:     0     6    12
##    probabilities: 0.000 0.333 0.667
##   left son=6 (6 obs) right son=7 (12 obs)
##   Primary splits:
##       noun splits as  LRR, improve=8, (0 missing)
##
## Node number 6: 6 observations
##   predicted class=o   expected loss=0  P(node) =0.1667
##     class counts:     0     6     0
##    probabilities: 0.000 1.000 0.000
##
## Node number 7: 12 observations
##   predicted class=op  expected loss=0  P(node) =0.3333
##     class counts:     0     0    12
##    probabilities: 0.000 0.000 1.000
```

For more on how to interpret `rpart` objects: http://cran.r-project.org/web/packages/rpart/rpart.pdf#rpart.object