Inference, probability and simulation

March 07, 2016

Data analysis (cont.)

Let's take a more careful look at the model we fit before:

```
affairs <- read.csv("http://koaning.io/theme/data/affairs.csv")
sample_model <-  lm(nbaffairs ~ I(age - 18)*child + factor(religious), data=affairs)
```

We took a look at some values of interest, like the estimated coefficients or the confidence intervals around them. It may also be interesting to take a look at predictions on the original dataset that we used (remember that `sample_model` carries the data used to fit the model).

```
yhat <- predict(sample_model)
head(yhat)
```

```
##     1     2     3     4     5     6
## 2.612 0.177 2.875 1.380 0.638 1.802
```

The `predict` method takes a number of uselful arguments, like `newdata`, which applies the estimated coefficients to a new dataset.

```
my_predictions <- predict(sample_model, newdata=data.frame("age"=54, "child"="yes", religious=1))
my_predictions
```

```
##    1
## 3.23
```

Usually, we want to see predictions with their uncertainty. Let's take a look at the documentation to see how to get confidence intervals:

```
?predict
```

Not very useful, right? The reason is that `predict` is a *generic function* that operates on different kinds of objects/models. Think about predictions for a linear model or for a logistic regression. They are still predictions but they are calculated differently and they should be offering different options. But they user should not need to remember the class of the model that was fit: and the end of the day, we have been insisting on the fact that objects in R carry a lot of information around. If we look at the bottom of the help file, we will see the method for `lm` models, which is what we want:

```
?predict.lm
```

After this small detour, we finally see how to get the confidence intervals:

```
my_predictions <- predict(sample_model,
              newdata=data.frame("age"=54, "child"="yes", religious=1),
              interval="confidence")
my_predictions
```

```
##    fit  lwr upr
## 1 3.23 2.06 4.4
```

A bit more on modeling

We can think about running some other kinds of models on our dataset. For instance, we could think about running a logistic regression.

```
logit_model <-  glm(I(nbaffairs > 0) ~ I(age - 18)*child + factor(religious),
                    data=affairs,
                    family=binomial(link="logit")) # link="logit" is the default
```

Nothing in the previous call should be odd. But we can now use the example to better appreciate the value of the object-oriented style in R:

```
fake_data <- expand.grid(age = c(18, 36, 54, 72),
                         child = c("no", "yes"),
                         religious = 1)
fake_data$prediction <- predict(logit_model, newdata=fake_data, type="response")
fake_data
```

```
##   age child religious prediction
## 1  18    no         1      0.198
## 2  36    no         1      0.398
## 3  54    no         1      0.638
## 4  72    no         1      0.825
## 5  18   yes         1      0.484
## 6  36   yes         1      0.489
## 7  54   yes         1      0.495
## 8  72   yes         1      0.500
```

We did two things here. First, we created a fake dataset by expanding on all the combinations of the values that were passed to `expand.grid`. Then, we applied our predicted model to this new dataset and got the predicted probabilities for each case. Notice I put those predictions back on the fake dataset to be able to see to what combination each prediction corresponds.

We now may want to, for instance, get confidence intervals for each predictions, or test the effect of a discrete change in a variable. We can obviously do it using the statistical theory that we know, but sometimes the easiest approach is to simulate.

Simulation and probability

We saw in the previous script the somewhat mysterious function `qt`. It actually belongs to a very rich family of functions that are associated with probability distributions with a common name structure.

The first letter indicates the operation, while the rest of the name indicates the distribution. In particular, `r` indicate random generation, `d` density function, `q` quantile function, and `p` distribution function. Therefore, `rnorm` will make an extration out of a normal distribution, `pnorm` will give the probability associated with a quantile in a normal distribution and so on. R provides many distributions, Student's t, Snedecor's F, Gamma, Beta, ... and they all follow the same convention.

Thus, `qt` in the previous calculates the probability associated with a quantile in a t distribution with given dof. These functions are very useful. Let's see them in action:

```
rnorm(1)
qnorm(0.975, mean=1, sd=1)
rnorm(5, mean=2, sd=5)
rbeta(1, 1, 5)
```

For simulations it is probably a good idea to set up a seed for the RNG.

```
set.seed(20150211)
rnorm(1, 2, 1)
set.seed(20150211)
rnorm(1, 2, 1)
```

A very useful function for simulation is `replicate` that let's us repeat expressions a number of times. We will see later on another more general approach to this idea of *repeating* things through loops. Consider the sampling distribution of the statistic, for instance:

```
n <- 10
sd(replicate(999, mean(rnorm(n, 3, 2))))
```

```
## [1] 0.647
```

which is approximately $\sigma/\sqrt{n}$, as expected.

We can also do bootstrap sampling using the same approach. The only thing that we need is something that a function that produces a sample with replacement from a vector:

```
x <- rnorm(25, 3.2, 1.7)
sd(replicate(999, mean(sample(x, length(x), replace=TRUE))))
```

```
## [1] 0.375
```

and that matches:

```
sqrt(vcov(lm(x ~ 1)))
```

```
##             (Intercept)
## (Intercept)       0.394
```

With these elements we can now think about, for instance, making extractions of the posterior distribution of the estimated coefficients in the section above to simulate confidence intervals. Or simulate the distribution of transformations of variables. But both tasks are probably easier with tools that we see when we talk about programming.