Writing unit tests for BaTFLED

Nathan Lazar

Aug 11, 2016

This tutorial is meant to walk you through writing unit tests using my package BaTFLED as an example.

Trying out unit testing

Before we start playing with BaTFLED, lets first try writing some unit tests for simple functions. For this we'll need the package `testthat` written by R guru Hadley Wickem.

```
pkgTest <- function(x) { # load packages and install if needed
  if (!require(x,character.only = TRUE)) {
    install.packages(x,dep=TRUE)
    if(!require(x,character.only = TRUE)) stop("Package not found")
}}
pkgTest('testthat')
```

Now define some simple function to play with. (I stole most of this bit from R blogger John Myles White here)

```
factorial <- function(n) {
  if(n==0) {return(1)} else {
    return(n * factorial(n-1))
  }
}
```

Some simple testing can be done with the functions `expect_equal`, `expect_true`, `expect_that`, etc. from the `testthat` package. The `test_that` function allows you to group tests and name them. The first argument is the name of the test which is displayed when a test fails.

```
expect_that(factorial(0), equals(1))
expect_true(factorial(3)==6)

test_that("factorial function works", {
  expect_equal(factorial(4), 24)
  expect_false(factorial(3)==5)
  expect_that(factorial(3)==5, is_false())
  expect_warning(log(-1))
})

test_that("this fails", {
  expect_equal(factorial(4), 12)
})

test_that("vector and matrix tests", {
  expect_equal(c(1,2,3), c(1,2,3))
  expect_equal(matrix(c(1,0,0,1), nrow=2, ncol=2), diag(1,2))
})
```

The real use case for this is to have a set of tests that you run on code as you're developing to make sure that you don't break anything. To do this, create a directory called `tests` in the BaTFLED3D directory that will store all of the test cases. In `tests` create a file `test1.R` that will contain your first set of tests. For example, you might have this:

```
expect_that(2 ^ 2, equals(4))
expect_that(2 + 2 == 4, is_true())
expect_that(2 == 1, is_false())
expect_that(1, is_a('numeric'))
expect_that(print('Hello World!'), prints_text('Hello World!'))
expect_that(log('a'), throws_error())
expect_that(factorial(4), equals(24))
expect_false(factorial(3)==5)
expect_warning(log(-1))
expect_that(factorial(5), equals(110))
```

To run this set of tests, you can use `test_file` to just run this file or `test_dir` to run all the test files in the directory. For `test_dir`, the files containing tests have to be named `test*` and are executed in alpha-numeric order.

```
test_file('tests/test1.R')
test_dir('tests')
```

Also, you can keep watch on a directory and run tests anytime something is changed. First make a new directory called `code` and make a file containing your `factorial` function. Then, try running the code below and changing or rewriting the factorial code to see if you've broken anything.

```
auto_test('code', 'tests')
```

Writing tests for BaTFLED

Now lets try testing some real code in my package BaTFLED. First walk through the Rmarkdown document `BaTFLED_sim_CP.Rmd` to see that you can run the package. Then, try changing parameters for building the toy model or running BaTFLED and add some tests specific to BaTFLED3D to your `test1.R` file. Some examples are below, note that there are random aspects to BaTFLED, so to get a specific result for some functions, you may need to set the seed.

```
# Some unit tests of the input_data object
input.data <- input_data$new(mode1.X = matrix(rnorm(30), nrow=3, ncol=10),
                             mode2.X = matrix(rnorm(36), nrow=4, ncol=9),
                             mode3.X = matrix(rnorm(40), nrow=5, ncol=8),
                             resp = array(rnorm(60), dim=c(3,4,5)))
expect_that(input.data$mode1.X, is_a('matrix'))
expect_equal(nrow(input.data$mode1.X), dim(input.data$resp)[[1]])

# Tests of get_data_params and mk_toy
cp.params <- get_data_params(list('decomp=CP'))
cp.toy <- mk_toy(data.params)
expect_that(cp.params$R > 0, is_true())
expect_that(is.null(cp.params$R1), is_true())
expect_that(cp.toy$core, is_a('array'))
expect_that(identical(cp.toy$resp, array(0,dim=c(cp.params$m1.rows,
```

```
                                                cp.params$m2.rows,
                                                cp.params$m3.rows))),
            is_false())

# High level test to be run after training a model (not really a "unit" test):
# The lower bounds should always be increasing
if(model.params$lower.bnd)
  expect_that(sum((trained$lower.bnd[-1]-trained$lower.bnd[-reps])<0), equals(0))
```

Feel free to dig into the `BaTFLED3D/R/` folder, find a function or object and try writing a test for it. The documentation (`?<function or object>`) should have an explanation and some examples. Please tell me if you find anything difficult to understand, missing or not working. Some of the simpler objects/functions that could use tests are:

Objects:
- `input_data`
- `CP_model`
- `Tucker_model`

Functions:
- `safe_log`
- `safe_prod`
- `exp_var`
- `mult_3d`: I plan on reimplementing this without the `rTensor` package so unit tests would be helpful
- `get_data_params`
- `get_model_params`

Core functionality that has some random aspects:
- `train_Tucker`
- `train_CP`
- `update_mode[123]_CP`
- `update_mode[123]_Tucker`
- `update_core_Tucker`

Some high level things that should be true after training a model:
- RMSEs should always be positive
- explained variation and correlations should be less than 1