Computational Skills for Biostatistics I: Lecture 10

Amy Willis, Biostatistics, UW

05 June, 2019

Advantages of R

R is great for many things

- Data analysis
- Statistical inference
- Statistics and statistics packages

R is not a universal language

Different fields use different languages

- Physics: Matlab/Python
- Geology: Python
- Applied math: Matlab/Java
- Computational biology: Python/R

R is not a universal language

Many statisticians don't use R

- Julia: great for large mixed models
- Python: image analysis, spatiotemporal modeling, anything with maps
- C++: Large sampling algorithms (common in Bayesian statistics)
- Java: non-Euclidean metric spaces
  - My PhD dissertation code was in Java

Other languages

Why use another language?

- Inertia
- Speed
- Field standard
- Portability
- Ease of use

Today's focus

Learning objectives

- Understand when you may want to write in C++
- See the infrastructure for interfacing R and C++
- Grasp the basic syntax of Python

It is not possible to learn a new language in a single lecture; it is possible to get a feel for a new language

Programming speed: R and C++

- There is a relatively easy way to use C++ "under the hood" of R
  - Package `Rcpp`
- This allows users to interface with your R package as usual...
- ... but with the potential for significant speed ups

Rcpp: example

```
Rcpp::sourceCpp("t_test_cpp.cpp")
set.seed(1)
x1 <- rnorm(30); x2 <- rnorm(50)
t_test_cpp(x1, x2)
```

```
## [1] -0.1796436
```

Rcpp: example

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
double t_test_cpp(NumericVector x1, NumericVector x2) {
    int n1 = x1.size();
    int n2 = x2.size();

    // Generate numerator and denominator
    double nume = mean(x1) - mean(x2);
    double denom = sqrt(var(x1)/n1 + var(x2)/n2);
    return nume/denom;
}
```

Rcpp: steps

1. Write code in C++
   - Preceed your function with `// [[Rcpp::export]]`
2. Compile your C++ code using `Rcpp::sourceCpp()`
3. Refer to your C++ function in `R`

C++

- C++ is a compiled language: `Rcpp::sourceCpp("t_test_cpp.cpp")` turns the program into machine-readable code
  - This results in highly efficient execution
- C++ is strongly-typed: the type of variable needs to be declared before its first use to allow for memory allocation

```
double t_test_cpp(NumericVector x1, NumericVector x2) {
    int n1 = x1.size();
    int n2 = x2.size();
    double nume = mean(x1) - mean(x2);
    double denom = sqrt(var(x1)/n1 + var(x2)/n2)
}
```

Rcpp

- More recent versions of Rcpp include "syntactic sugar"
- Functionality for easy-to-use R functions
- i.e. C++ is not vectorized, but certain Rcpp functions are
  - e.g., implementations of `sapply`, `pnorm`, `ifelse`, vectorised addition/pointwise multiplication
  - dirk.eddelbuettel.com/code/rcpp/Rcpp-sugar.pdf

Rcpp: when not to bother

- Many operations in R are already backed by C/C++/Fortran, so are already fast

```
total <- 0
x <- rnorm(100)
y <- rnorm(100)
for (i in 1:10) {
  total <- total + x[i]*y[i]
}
x %*% y
```

```
##           [,1]
## [1,] 1.699336
```

Rcpp: when to bother

- Tasks that cannot be vectorised in R can typically be sped up in C++
  - Loops that you can't get rid of
- Iterative procedures where the next step depends on the current step
  - Gradient descent
  - Expectation maximisation
  - Markov chains

Rcpp: considerations

- Always prototype in R, then translate
  - When starting out, assume nothing about syntax; check everything!
- Expect more difficulty debugging
- It is also possible to do with C instead of C++
  - It's much harder and benefits are limited

Rcpp: considerations

- You will need
  - R (>= 3.1)
  - `Rcpp`
  - A C++ compiler
    - Windows: Install `Rtools`
    - Mac: Install `XCode` (Warning: takes a lot of memory)

How to learn a programming language

Good advice for *any* programming language

- Get something minimal working first
- Adapt examples
- Use Google liberally
    - "c++ mean of vector"

Moving beyond R to Python

In your career as a statistician/data scientist, you will probably need *some* familiarity with Python

- Collaborating on projects, especially interdisciplinary
- Working with spatial data/image analysis
- Significantly easier to read/learn if you know R

Moving beyond R to Python

Python and R are arguably the two most dominant languages for data scientists. R is often preferred by statisticians, and Python is often preferred by computer scientists.

Python

- Often faster than R
  - Typically in the same settings that C++ are preferable to R, Python will beat R too
  - Often faster for IO and loops
- Large online community
- Modules and libraries extend base functionality
  - `numpy`, `scipy`, `pandas`, `scikit-learn`

Python

- Great for
  - convex optimisation
  - ML libraries (though TensorFlow/keras now runs from R)
- There is never any reason to be a snob about this stuff...
  - ... or anything else
- Both Python and R are C under the hood, so it's not always worth switching for speed considerations
- May lose statistical audience

Language comparison

Gibbs sampler in various languages (revisited) by Darren Wilkinson (Newcastle)

- R: 1.0
- Python: 1.86
  - PyPy (alternative implementation of Python with a JIT compiler): 14
- Java: 38
- C: 54

Language comparison

The Python code ran [~2x] faster than R. To me, that is ***not really worth worrying too much about****. Differences in coding style and speed-up tricks can make much bigger differences than this... C was the fastest, and this is the reason that most of my MCMC code development has been traditionally done in C. It was around 60 times faster than R.* ***This difference is worth worrying about****, and can make the difference between an MCMC code being practical to run or not.*

Python: modules, assignment, loops

Learning some Python syntax

```
import numpy as np
x = np.zeros(shape = (11, 2))
x[:, 1] = np.linspace(start = 0, stop = 10, num=11)
x
x[:, 0] = x[:, 1] / 3 + 1
x
for i in range(11):
    x[i, 1] = x[i, 1] + 1
x
```

Python: Syntax

- Indentation/spacing is *very important*
  - Use either spaces or tabs
  - Spacing determines the grouping of operations, e.g., levels of loops
- Indexing starts at 0, not 1
  - First element in an array is number 0
- You will usually need the `numpy` library for working with arrays and matrices

Python: matrix algebra

```
import numpy as np
A = np.array([[1, 2, 3], [4, 5, 6]])
A.T # transpose of A
np.transpose(A) # transpose of A
b1 = np.array([0.5, 1, 2])
b1
b2 = np.reshape(b1, (3, 1))
b2
A.dot(b2)
np.matmul(A, b2)
```

Python: data analysis

```python
import pandas as pd
dt = pd.read_csv('lecture2/colon_cancer.csv')
dt.columns
dt['Tissue']
dt.iloc[1] # get second row
dt['Polyp type'].describe()
dt.describe(include='all')
```

Python scripts

- You can execute a Python script on the command line by running `/usr/local/bin/python2.7 myscript.py`
  - or the equivalent for your installation location
  - or `python myscript.py` if python is in your PATH
- You can execute a Python script via a shell script

Python on the cluster

With the script you want to run in `my_script.py` and with header `#!/usr/local/bin/python2.7`, you would have `call_script.sh` as

```
#!/bin/sh
/usr/local/bin/python2.7 my_script.py
```

and `submit_sim.sh` as

```
#!/bin/sh
qsub -t 1-22
  -cwd -e Trashfiles/ -o Trashfiles/
  -q s-normal.q
  -M name@uw.edu -m e call_script.sh
```

Python: scripting

- Alternative to RStudio: Spyder
- Alternative to RMarkdown: Jupyter Notebooks
    - Easy install: `conda install ipython jupyter`
        - Maybe `conda install pyzmq` too
    - Awesome tutorials! (linked)

Python: Versions

- Python 2.7 and Python 3.3 are the most common
- Python 2.x is the legacy version
  - Better libraries, documentation
  - Default distribution for Unix
- Python 3.x is the *present and future* (from the Wiki)
  - Moving forward, new features will be in Python 3 but not added to Python 2
  - Personal recommendation: learn Python 3

Wrap up

Wrap up of the class

- Learning programming languages is like learning any language: *immersion* and *practice* are critical
  - Continue to practice
  - Every minute you spend coding improves your skills as a programmer

Wrap up of the class

- This syllabus is very modern, emphasising best practices, cutting-edge software, and self-teaching
- Well done to everyone for keeping up, *especially those who started with less R background*

You can add the following your CV/Resume

Computing skills

- Proficient: R
  - Data analysis: tidyverse (dplyr, tibble, stringr, readr, tidyr), magrittr, [others from other classes, e.g. survival, lme4, inla...]
  - Reporting: ggplot2, RMarkdown, knitr
  - Performance: Rcpp, benchmarking, debugging, vectorisation
  - Development: R package creation and maintenance, simulator
- Familiar: git, shell scripting, distributed/cluster systems, optimisation algorithms, Python, C++

Final wrap-up

- Homework 10: final homework due next *Wednesday*
    - A final exercise with tidyverse
    - Basic use of Rcpp
    - Basic use of Python
- Final 561 office hours: *Tuesday 12:30-1pm*
- **Best of luck with your continuing education as a programmer and data analyst!**