Extending R

Managing your statistical workflow to save human and computational time.

Ryan Del Bel   rdelbel@gmail.com
github.com/rdelbel/ExtendingR

Introduction

- Everyone knows that R can fit esoteric models and create beautiful graphics.
- This is only a small part of an (applied) statistical workflow.
- We also need to write reports, run expensive code and use other software
- We can tightly integrate R into this broader workflow to save computational and human time.

Overview

- I will quickly survey packages/techniques to show what is available to help the workflow of an applied biostatistician
- References to more comprehensive documentation, tutorial, and books will be provided.

Rstudio

- Rstudio is an integrated development environment (IDE) for R.
- Easy to use and works out of the box
- Integrates with many of the packages we will use today, streamlines package creation and integrates with git(hub).
- The GUI and text editor are somewhat lacking, but more control over panes and integration with vim are promised.

Rstudio

- ‘Projects’ help organize…projects
- Tab completion
- Many shortcuts to make working with R easier.
- Ctrl+up/down: go to previous/next function
- Alt+0: fold all code
- Ctrl+d: Delete Line
- Ctrl+Alt+i: Create Chunk (for knitr)

http://www.rstudio.com/ide/docs/using/keyboard_shortcuts

knitr

- Written by Yihui Xie
- Knitr can ‘knit’ several markup languages and several programming languages.
- This enables reproducible research and automated reporting
- We will discuss knitting LaTeX and R.

What about Sweave?

- “knitr ~ Sweave + cacheSweave + pgfSweave + weaver + animation::saveLatex + R2HTML::RweaveHTML + highlight::HighlightWeaveLatex + 0.2 * brew + 0.1 * SweaveListingUtils + more”

Knitr References

- Setup/todays example

https://github.com/rdelbel/reportRx

- Website has surprisingly good documentation

http://yihui.name/knitr/

- Official book

http://www.crcpress.com/product/isbn/9781482203530

reportRx

- When working on an applied clinical project you might write the results of the analysis to many .csv and image files.
- You then open word (because that is what clinicians use…), copy and paste everything in, spend a long time format tables and hope nothing went wrong.
- After finishing the report, the clinician may change the sample and then you have to do everything again.

reportRx

- This wastes a lot of time, is unpleasant, and prone to error.
- If we set up everything in a clever way then we can use knitr to solve all of these problems.
- reportRx is a set of functions that take in a minimal input and generate LaTeX output commonly included in clinical reports.

reportRx

- These functions are tools we can use with knitr to help create a .pdf report
- Results, tables and figures are automatically injected into the report
- With the help of other free command line software can convert it to a .docx report
- Word macro can ‘clean up’ the conversion
- If the sample changes, we just change the underlying excel file and recompile the report with 1 button
- Created by Ryan Del Bel & Wei Xu.

reportRx references

- High level documentation/examples

https://github.com/rdelbel/reportRx

- Functions are also documented within R

Speeding up R

- There are many ways one can speed up R.
- Use faster functions (eg fastLm vs lm)
- Use distributed computing (eg hadoop)
- Use a supercomputer
- Use pqR (Radford Neal)
- Offload work to other languages (python pandas for wrangling data, c++ for numerical computation, etc)

Speeding up R

- We will talk about the following 3 ways
- Package parallel/multicore to use all our cores
- Package Rcpp to seamlessly integrate R and C++
- Accessing existing C functions directly

parallel/multicore

- Usually R only runs on 1 core. Most of our computers have 4-8.
- On unix based systems, multicore::mclapply and on windows parallel::parLapply can parallelize an lapply over all our cores.
- On unix this is trivial, and on windows it just requires a few more lines of code.

mclapply example

- Lets write a mclapply to run a simulation sim.
- I want to fix beta and modify n and then fix n and modify beta.
- we can coerce this problem into one that lapply can solve as follows (not the only way)

```
df=data.frame(t(rbind(
cbind(rev(seq(200,1000,100)),log(3)),
cbind(500,sapply(seq(2,4,.25),log)))))
```

mclapply example

- require(multicore)
- mclapply(df,function(x) sim(n=x[1],beta=x[2]), mc.set.seed = F, mc.preschedule=F)
- mc.set.seed=F uses the same seed on each iteration
- Setting mc.preschedule=FALSE does not presechedule jobs to cores. Is useful when you have high variance between in runtimes between iterations of your lapply.

mclapply example

- By default mclapply uses all your cores. You can specify how many you want to use by specifying the parameter mc.cores.
- See the documentation for more details

Mclapply example 2

- We can also use a single lapply to run multiple functions

```
ns=rev(seq(200,1000,100))
df=data.frame(t(rbind(cbind(ns,“sim1"),
cbind(ns, “sim2"))),stringsAsFactors=F)
mclapply(df,function(x){z=get(x[2])
z(n=as.numeric(x[1]))})
```

parallel::parLapply example

- We first need to make a cluster
- Then we need to load all of the packages and functions to our cluster
- Then we can run parLapply

parallel::parLapply example

```
require(parallel)
cl <- makePSOCKcluster(8) #number of cores
setDefaultCluster(cl)
clusterEvalQ(NULL, require(survival))
clusterExport(NULL, c(‘foo1’,’foo2’,…,’foon’))
parLapply(NULL, 1:8, function(x) foon(x))
stopCluster
```

http://stackoverflow.com/a/17201586/2529376

Rcpp

- We will combine the ease of use of R with the speed of C++
- Rcpp makes it simple to combine C++ and R
- Rcpp makes doing R-like things simple in C++
- Rstudio integrates with C++/Rcpp to make things even more seamless
- Created by Dirk Eddelbuettel and Romain Francois

4 usage cases

- There are 4 main ways to use Rcpp
- Use C++ code in R
- Use C code in R
- Use armadillo (C++ linear algebra library based on LAPACK) based code in R
- Do R like things with R like code at C++ like speed

Rcpp Sugar

- Rcpp provides its own classes to mimic those in R. (eg DataFrame, List, NumericVector)
- We can use these to write R-like code to do R-like things at C++ speed.
- d/p/q/r functions, vectorized arithmetic, apply, etc are all usable

Rcpp References

- Beginners (R)cpp Tutorial (very good, read this first)

http://adv-r.had.co.nz/Rcpp.html

- Todays Example

https://raw.github.com/rdelbel/ExtendingR/master/Rcpp/Rcppexample.R

https://raw.github.com/rdelbel/ExtendingR/master/Rcpp/Rcppexample.cpp

- Rcpp Sugar

http://cran.r-project.org/web/packages/Rcpp/vignettes/Rcpp-sugar.pdf

Google will have answers to many (R)cpp problems you encounter.

Extracting Existing Functions

- Often times the main part of our code is a predefined function
- Can carefully go through the source and cut out what we don’t need
- If we are lucky we might find that the core of the function is already written in C or fortran
- We can then write a wrapper in R around this function

Coxph example

- We will speed up coxph as a case study
- coxph is an extremely robust function written by Terry Therneau. It has a huge amount of functionality that generalizes the standard coxph model, checks all of its inputs and produces beautiful output.
- When we do a GWAS with underlying coxph models we do not care about the advanced statistical features or beautiful output.
- We just want the HR, CI, and p-value of a ‘standard’ coxph model.

coxfit6

- Easiest way to view the coxph source code is to download the survival package source off the survival CRAN page. The R source code will be in the R folder, and the compiled (c/c++/fortran) code will be in the src folder.
- If we trace a call to coxph using 'standard' parameters we find the following structure:

coxph->coxph.fit->coxfit6-> more

- coxfit6 is a C function that does the actual fitting
- If we learn the input and output of coxfit6 we can call it directly to fit coxph models faster.

coxfit6

- Learn by reading source code, modifying coxph.fit to print input/output to coxfit6 and testing the behaviour of coxfit6.
- See how it deals with exceptions such as models that do not converge, parameters that do not converge, singular matrices, etc.
- Once we learn how to work with coxfit6 we can write a small wrapper function to interface with it directly, rather than calling coxph

Fast coxph

- The function on the next page can fit a ‘standard’ non-null coxph model with numeric covariates, up to one strata, and no missing data.
- Could easily handle missing data with complete.cases and ‘factors’ with model.matrix if we wanted to.
- It does not check for exceptions. (see reference)
- In practice we might want to move the sorting out of the function to reduce unnecessary computation

```
#time/status = vector, x=vector/matrix, strata=vector
fcoxph<-function(time,status,x,strata=NULL){
  n=length(time)
  if(class(x)!="matrix") x=matrix(x)
  if(!is.null(strata)){
    sorted = order(strata, time)
    newstrata=as.integer(c(1 * (diff(as.numeric(strata[sorted])) !=0), 1))
  }else{
    sorted=order(time)
    newstrata=rep(0L,n)}
.Call("coxfit6", 20L, time[sorted],
as.integer(status[sorted]),
x[sorted,,drop=F],rep(0,n),rep(1,n),newstrata,
1L, 1e-09,  1.818989e-12,rep(0,ncol(x)), 1L)}
```

Fast coxph reference

- Todays Example

https://raw.github.com/rdelbel/ExtendingR/master/fast-coxph/coxphexample.R

GENmatic

- Goal is to provide simple R interface to automate common GWAS and candidate gene study tasks
- Will integrate R, knitr, plink, haploview, phase, Rserve, unix commands and more
- Save human time by automating QC, analysis and reporting
- Sometimes also saves computational time
- Created by Ryan Del Bel & Wei Xu

GENmatic References

- Still in alpha. High level documentation and examples will eventually be here

https://github.com/rdelbel/GENmatic

- Current functions are documented
- More functions to come
- Can install package with the following code

```
install.packages('devtools')
devtools::install_github('GENmatic','rdelbel')
```