Computing

Daniel J. McDonald

9/26/2017

Writing reports

R Markdown

- This is an R Markdown presentation. Markdown is a simple formatting syntax for authoring HTML, PDF (via LaTeX). For more details on using R Markdown see http://rmarkdown.rstudio.com.
- You can edit `.Rmd` files in Emacs, Rstudio, or others. I use Rstudio.
- When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document.

Slide with Bullets

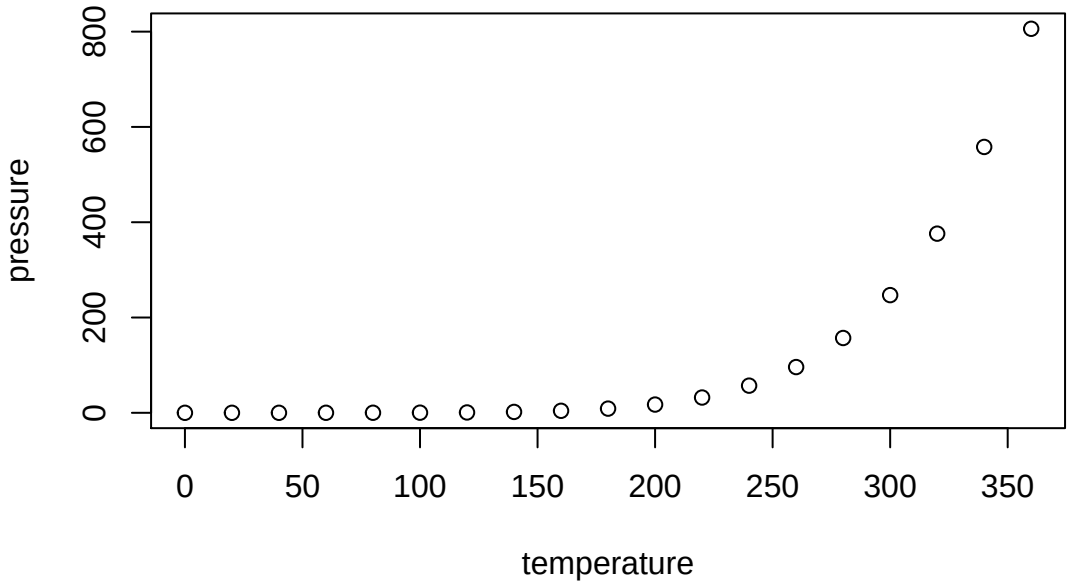
- Bullet 1
- Bullet 2
- Bullet 3

Slide with R Output

```
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00
```

Slide with Plot



When to use it

- I now use Rmd for almost everything.
- For class presentations, I knit to html (as here).
- For assignments/solutions/notes, I knit to pdf.

```
---
title: "HW 1"
author: "STAT - S782"
date: "Due 26 September 2017"
output:
  pdf_document:
    includes:
      in_header: ../Support/782macros.tex
    number_sections: yes
    template: ../Support/dm-docs.tex
    toc: no
---
```

Using (latex) macros

- The `in_header` line includes a text document with macros like `\newcommand{\Expect}[1]{\mathbb{E}\left[ #1 \right]}`
- You can also write these directly in your `.Rmd`, though I find this to be buggy.
- The `template` line is a template I made which includes packages, etc. And it makes the title the way I like.

Fancy talks

- I think the html presentation looks ugly, but it's fine for class.
- I made the seminar talk last week in .Rmd .

```
---
title: "Approximation regularization for analysis of large data sets"
author: |
  | Daniel J. McDonald
  | Indiana University, Bloomington
  | \alo{\footnotesize mypage.iu.edu/~dajmcdon}
date: |
  | 4 November 2016
output:
  beamer_presentation:
    keep_tex: yes
    fig_caption: no
    includes:
      in_header: ApproxRegMacros.tex
---
```

Benefits

- Easy to display/not your code and output
- Easy to regenerate when making minor adjustments.
- I hate the following workflow:
  1. Write a presentation/paper in LaTeX.
  2. Include a pdf figure.
  3. Figure labels are too small.
  4. Regenerate figure in R and save (fig and code).
  5. Compile .tex again.
  6. Now the lines don't show up.
  7. Etc.
- I can do everything in one document now.

Expert workflow

- This is the ideal. It's fine for talks and most things, but not for complicated research or papers.
- For papers:
  1. Complicated code in many files.
  2. Paper needs a special .tex template.
  3. Often, R results are saved on a computing cluster.
- In this case, you should use *minimal make* (see K. Broman's tutorial).
  1. You write a *Makefile* in your directory which indicates dependencies.
  2. You keep your directory organized
     - project/
     - project/code/
     - project/manuscript/
     - project/data/
     - project/figures/
     - project/Makefile
  3. The Makefile determines what needs to be (re)run to create the manuscript and runs scripts in succession.

(less-than) Ideal

- I'm not good about doing that (I wish I were).
- I do keep things organized though.
- Code should consist of functions, perhaps wrapped in a package, a script containing parameters to be set, a script to run the functions to produce output, another script to produce graphics from the output.
- Don't put everything in one long script.
- **Don't** type anything in the console.
- Avoid creating a series of different versions: `code1.R`, `code1modified.R`, `newestCode1.R`, etc.
- Use version control (built into Rstudio) to handle such things or move old versions to a `Deprecated` folder.
- **Still make interim reports for me or collaborators in `.Rmd`**
- You can load code or output into your `.Rmd`.
- So if you like `python`, you should learn Jupyter or save output and load into `.Rmd`

Cluster computing

IU cluster

1. Create an account on Karst.
2. You can now use http://rstudio.iu.edu/ if you want to run programs on another machine.
3. This is not in parallel.

Parallel on a cluster

- For IU computing, parallel (or any job submission not via Karst desktop), occurs via a TORQUE script.
- These are a pain to write. Basically you need:
    1. Some code which does things (R, C++, python, etc.)
    2. A script which tells the scheduler how many resources you need and how to run your code (`R CMD BATCH myscript.R`)
    3. Then you ssh into Karst or Big Red and run `qsub myTorqueScript`
    4. Wait until it is done.
    5. If you are doing complicated things, you have to do this.

Simple parallelization

- Most of my major computing needs are "embarassingly parallel"
- I want to run a few algorithms on a bunch of different simulated datasets under different parameter configurations.
- Perhaps run the algos on some real data too.
- R has packages which are good for parallelization (`snow`, `snowfall`, `Rmpi`, etc.)
- It also has a package for parallel experiments **which works on the cluster** `batchtools`.

Intro to `batchtools`

- Batchtools does a few things:

    1. It automates writing/submitting TORQUE scripts.
    2. It automatically stores output, and makes it easy to collect.
    3. It generates lots of jobs.

4. All this from R directly.

- In addition to rstudio.iu.edu, I find it useful to be able to ssh into Karst (or Big Red)

```
ssh dajmcdon@karst.uits.iu.edu
```

- And you should download an sftp client (Cyberduck or Fetch are on IUware for Mac)

Setup

1. Create a directory where all your jobs will live (in subdirectories). Mine is `newBatch`
2. In that directory, you need a template file. `myBatchJobs.tmpl`
3. A configuration file which lives in your home directory. You must name it `.batchtools.conf.R`.

My template

```
#PBS -N <%= job.name %>
## merge standard error and output
#PBS -j oe
#PBS -q condo
#PBS -W group_list=stats
## direct streams to our logfile
#PBS -o <%= log.file %>
#PBS -l walltime=<%= resources$walltime %>,nodes=<%= resources$nodes
  %>:ppn=<%= resources$ppn %>,vmem=<%= resources$memory %>


## Export value of DEBUGME environemnt var to slave
export DEBUGME=<%= Sys.getenv("DEBUGME") %>

## Run R:
## we merge R output with stdout from PBS, which gets then logged via -o option
module load r
cd ~/newBatch
Rscript -e 'batchtools::doJobCollection("<%= uri %>")'
```

`batchtools.conf.R`

```
cluster.functions = makeClusterFunctionsTORQUE('~/newBatch/myBatchJobs.tmpl')
```

`.modules`

- I always load `R` because that's what I use 99% of the time.
- To do this automatically, add to your home directory a file called `.modules`.
- It should contain only the line `module load r`.
- You don't get to use the software unless you first issue a `module load` command when you log on.

Using `batchtools`

Comments

- Unfortunately, `batchtools` is a recent upgrade of `BatchExperiments`
- I used that one a lot. Some things have changed.
- Start with nicely organized code.

Workflow

See the vignette: `vignette("batchtools")`

1. Create a Registry with `makeRegistry()` (or `makeExperimentRegistry()`) or load an existing from the file system with `loadRegistry()`.
2. Define computational jobs with `batchMap()` or `batchReduce()` if you used `makeRegistry()` or define with `addAlgorithm()`, `addProblem()` and `addExperiments()` if you started with `makeExperimentRegistry()`. It is advised to test some jobs with `testJob()` in the interactive session and with `testJob(external = TRUE)` in a separate R process. Note that you can add additional jobs if you are using an `ExperimentRegistry`.
3. If required, query the data base for job ids depending on their status, parameters or tags (see `findJobs()`). The returned tables can easily be combined in a set-like fashion with data base verbs: union (`ojoin()` for outer join), intersect (`ijoin()` for inner join), difference (`ajoin()` for anti join).
4. Submit jobs with `submitJobs()`. You can specify job resources here. If you have thousands of fast terminating jobs, you want to `chunk()` them first. If some jobs already terminated, you can estimate the runtimes with `estimateRuntimes()` and chunk jobs into heterogeneous groups with `lpt()` and `binpack()`.
5. Monitor jobs. `getStatus()` gives a summarizing overview. Use `showLog()` and `grepLogs()` to investigate log file. Run jobs in the currently running session with `testJob()` to get a `traceback()`.
6. Collect (partial) results. `loadResult()` retrieves a single result from the file system. `reduceResults()` mimics `Reduce()` and allows to apply a function to many files in an iterative fashion. `reduceResultsList()` and `reduceResultsDataTable()` collect results into a `list` or `data.table`, respectively.

Example (slightly dated)

```
library(BatchExperiments)
library(precondls)
library(splines)

reg = makeExperimentRegistry(id='AllCompressionAnalyses', packages = 'precondls')

## Parameter settings (make changes here) ------------

## Cluster resources
n.chunks = 200 # for submitting multple jobs on a cluster
nodes = 1
ppn = 1
walltime = '24:00:00'
memory = '10gb'

## Data generation parameters
covType = 'corrUniform'
```

```
noisesd = 50 # note that this value propagates to the ridge algorithm with
                      # optimal lambda, see line 72, and change there as well
rhos = c(.2,.8)
ntrain = 5000
ntest = 5000
p = c(50,100,250,500)
nrepls.sim = 50

## Compression algorithm parameters (simulation)
compTypes = c('qxy','qxqy','linComb','convexComb')
lam.min.sim = 0.05
lam.max.sim.randn = 150
lam.max.sim.constant = 50000
q.sim = c(500, 1000, 1500)

## Compression algorithm parameters (genetics)
compTypes.genes = c('xy','qxy','qxqy','linComb','convexComb')
trainFrac = .75
lam.max.genes = .3
q.genes = c(10000,20000)

## Compression algorithm parameters (galaxies)
compTypes.galaxies = c('xy','qxy','qxqy','linComb','convexComb')
lam.max.galaxies = .3
q.galaxies = c(1000, 2000)
p.galaxies = 250


## Code to run the simulation --------------------
addProblem(reg, id='ridgePrior', dynamic=createData, seed=12345)
dataParamsRidge = list(ntrain=ntrain, ntest=ntest,
    covType=covType, rho=rhos, p=p,
    bmethod='randN',  SNR=NA, noisesd=noisesd, sd=sqrt(pi/2))
dataRidge.design = makeDesign('ridgePrior', exhaustive=dataParamsRidge)

addProblem(reg, id='constantBeta', dynamic=createData, seed=12345)
dataParamsConst = list(ntrain=ntrain, ntest=ntest,
    covType=covType, rho=rhos, p=p,
    bmethod=c('constant','ones'),  SNR=NA, noisesd=noisesd)
dataConst.design = makeDesign('constantBeta', exhaustive=dataParamsConst)

compressSimAlgo <- function(job, static, dynamic, ...){
    out = compressedRidge(dynamic$Xtrain, dynamic$Ytrain,...)
    res = list(
        train = out$train,
        GCV = out$GCV,
        lam = out$lam,
        df = out$df,
        test = colMeans( (c(dynamic$Ytest) - predict(out, dynamic$Xtest))^2 ),
        esterr = colMeans( (dynamic$bstar - out$bhat)^2 ),
        oraclePred = mean( (dynamic$Ytest - dynamic$Xtest %*% dynamic$bstar)^2 )
        )
    return(res)
```

```
}

ridgeSimOptimAlgo <- function(job, static, dynamic, ...){
    ## for use when the betas are gaussian
    lamstar = (50 / sqrt(pi/2))^2 / nrow(dynamic$Xtrain) # 50 is the noisesd, change if you change it a
    out = compressedRidge(dynamic$Xtrain, dynamic$Ytrain, 'xy', lam=lamstar)
    res = list(
        train = out$train,
        GCV = out$GCV,
        lam = out$lam,
        df = out$df,
        test = mean( (dynamic$Ytest - predict(out, dynamic$Xtest))^2 ),
        esterr = mean( (dynamic$bstar - c(out$bhat))^2 ),
        oraclePred = mean( (dynamic$Ytest - dynamic$Xtest %*% dynamic$bstar)^2 )
        )
    return(res)
}

ridgeSimAlgo <- function(job, static, dynamic, ...){
    ## for use otherwise
    out = compressedRidge(dynamic$Xtrain, dynamic$Ytrain, 'xy', ...)
    best = which.min(out$GCV)
    res = list(
        train = out$train[best],
        GCV = out$GCV[best],
        lam = out$lam[best],
        df = out$df[best],
        test = mean( (dynamic$Ytest - predict(out, dynamic$Xtest)[,best])^2 ),
        esterr = mean( (dynamic$bstar - c(out$bhat[,best]))^2 ),
        oraclePred = mean( (dynamic$Ytest - dynamic$Xtest %*% dynamic$bstar)^2 )
        )
    return(res)
}

olsSimAlgo <- function(job, static, dynamic, ...){
    out = lm(dynamic$Ytrain~dynamic$Xtrain)
    res = list(
        train = mean(residuals(out)^2),
        df = length(dynamic$bstar)+1,
        test = mean( (dynamic$Ytest - cbind(1, dynamic$Xtest) %*% coef(out))^2 ),
        esterr = mean( (dynamic$bstar - coef(out)[-1])^2 ),
        oraclePred = mean( (dynamic$Ytest - dynamic$Xtest %*% dynamic$bstar)^2 )
        )
    return(res)
}

addAlgorithm(reg, id='compressSim', fun=compressSimAlgo)
compressSimRidge.design = makeDesign('compressSim',
    exhaustive=list(compression=compTypes, q=q.sim, lam.min = lam.min.sim,
        lam.max=lam.max.sim.randn))
compressSimConst.design = makeDesign('compressSim',
    exhaustive=list(compression=compTypes, q=q.sim, lam.min = lam.min.sim,
        lam.max=lam.max.sim.constant))
```

```
addAlgorithm(reg, id='olsSim', fun=olsSimAlgo)
ols.design = makeDesign('olsSim')

addAlgorithm(reg, id='ridgeSimOptim', fun=ridgeSimOptimAlgo)
ridgeoptim.design = makeDesign('ridgeSimOptim')

addAlgorithm(reg, id='ridgeSimConst', fun=ridgeSimAlgo)
ridgeconst.design = makeDesign('ridgeSimConst', exhaustive=list(lam.max=lam.max.sim.constant))

## Ridge prior
addExperiments(reg, repls=nrepls.sim, prob.designs=dataRidge.design,
               algo.designs = list(compressSimRidge.design, ols.design, ridgeoptim.design))

## Constant beta(s)
addExperiments(reg, repls=nrepls.sim, prob.designs=dataConst.design,
               algo.designs = list(compressSimConst.design, ols.design, ridgeconst.design))
ids = getJobIds(reg)
chunked = chunk(ids, n.chunks=n.chunks, shuffle=TRUE)

submitJobs(reg, chunked, resources=list(nodes=nodes, ppn=ppn, walltime=walltime, memory=memory))
```