

Research Notes

Writing R Packages

July 13, 2008

1 Getting Started: A Simple Example

First, go to a directory that you want to put your package. Open a R session, and you can use `getwd()` to check the directory. Here is the code to make a simplest package called `mypkg`

```
require(stats)
## two functions and two "data sets" :
f <- function(x,y) x+y
g <- function(x,y) x-y
d <- data.frame(a=1, b=2)
e <- rnorm(1000)

package.skeleton(list=c("f","g","d","e"), name="mypkg")
```

Notice that a subdirectory `mypkg` will be created under your current location, and you may see some subsubdirectories and files there. Skipping the details, and let's first check this package that contains two data set and two simple functions. In the command line of your initial directory, type

R CMD check mypkg

For some reason (explained in the next section), I cannot get `mypkg` pass the checking point, because it generates a example R file which contains symbols `~` in the main area:

```
> ~~ simple examples of the most important functions ~~
Error: unexpected symbol in "~~ simple examples"
```

Nevertheless, we don't think this is a big problem. We can install this package to a local directory `.Rlibs` under the home directory through

R CMD INSTALL mypkg -l \$HOME/.Rlibs

Finally, load the package within R, and we can use the functions and data sets in this package.

```
library("mypkg", lib.loc = "/home/grad/zo2/.Rlibs")
```

```
data(e)
hist(e)
f(1, 4)
g(3, 2)
```

In order to distribute this package, we need to make a tar ball. This can be done by following command, which generates `mypkg_1.0.tar.gz`.

R CMD build mypkg

2 Mask Your Functions: Namespace

Continuing the previous example, by default, when we load `mypkg`, functions f and g are both in the current environment. Suppose we want to use a function called h , which returns the result from functions f and g , but we do not want to keep functions f and g in the current environment. Instead, put f and g in the packaged local namespace, and only export function h to the main field.

To do this, first hack into directory `//mypkg/R/`, and create a new file `h.R` with

```
'h' <-
function(x,y) c(f(x, y), g(x, y))
```

Go back to directory `//mypkg/`, create a file `NAMESPACE` with

```
export(h)
```

Re-install the package

```
R CMD INSTALL mypkg -l $HOME/.Rlibs
```

We can double check that f and g are masked.

```
> library("mypkg")
> h
function (x, y)
c(f(x, y), g(x, y))
<environment: namespace:mypkg>
> h(1, 2)
[1] 3 -1
> f
Error: object "f" not found
> ?h
No documentation for 'h' in specified packages and libraries:
you could try 'help.search("h")'
> ?f
f                package:mypkg                R Documentation
~~function to do ... ~~
Description:
  ~~ A concise (1-5 lines) description of what the function does. ~~
```

We noticed that help file for function h is missing, but we do have the help file for function f . To understand how the help files works, go to directory `//mypkg/man/`. Now we suddenly realize the previous package checking error is due to a automatically generated document file `mypkg-package.Rd`.

After fixing the examples in `mypkg-package.Rd`, running the checking code reveal additional problems with non-edited R document files. Fixing those problem (for example, deleting all irrelevant documents) will allow us to pass the checking, but with some warning messages

```
R CMD check mypkg
```

```
...
WARNING: There were 2 warnings, see
  //package/simple/mypkg.Rcheck/00check.log
for details
```

3 Linking with C Source Codes

Now, let's build a R package that uses C scripts to evaluate kernel functions from the scratch. Make a subdirectory called `myk`. Create the folder structure as follows:

```
[zo2@ocean myk]$ ls
DESCRIPTION  man/  NAMESPACE  R/  src/
```

The minimal requirement for a package does not need the subdirectory `//myk/data/`, but manual, and R subdirectory is required. We also want to put the C source files in `//myk/src/`.

```
//kernelCalculation.cpp
// regression kernel calculations using c++
#include <iostream>
#include <math.h>
using namespace std;
extern "C"{
    // getDesignCpp: To calculate the design matrix in cpp code.
    // Diagonal kernels with maximum 1.
    int getDesignCpp(double *x, // n*d, data matrix vector
        double *c, // p*d, center matrix vector
        double *l, // d*1, kernel vector
        int *isi, // p*1, indicator of intercept
        int *n, // n, number of observations
        int *p, // p, number of kernels (model dimension)
        int *d, // d, observation dimension
        double *z // n*p, design matrix in a vector
    ){
        int i, j, k;
        double prodl, eitem, esum;

        //Calculate the exponent multiplied by the constant
        for(i=0; i<*n; i++){
            for(j=0; j<*p; j++){
                if(isi[j] == 1){
                    z[i+j*(*n)] = 1;
                }else{
                    esum = 0.0;
                    for(k=0; k<*d; k++){
                        eitem = x[i+(*n)*k] - c[j+(*p)*k];
                        esum -= l[k] * eitem * eitem;
                    }
                    z[i+j*(*n)] = pow(M_E, esum);
                }
            }
        }
        return 1;
    }
}
```

Next, we need the R script that calls the C library. Note, the shared library `myk.so` will be generated during the compiling, and we do not need to worry about it now. The R script should be located in `//myk/R/`

```
# getdesign.R
getdesign <- function(dMat, cMat, theta){
  nvec <- theta$nvec;
  cMat1 <- rbind(1, cMat);
  isi <- rep(0, sum(nvec>0));
  if(nvec[1] > 0){
    isi[1] <- 1;
  }
  z <- .C("getDesignCpp",
        as.double(dMat),
        as.double(cMat1[nvec>0,]),
        as.double(theta$L),
        as.integer(isi),
        as.integer(dim(dMat)[1]),
        as.integer(length(isi)),
        as.integer(dim(dMat)[2]),
        z = double(dim(dMat)[1]*length(isi)))$z; # n*p' design matrix
  z <- matrix(z, nc=length(isi));
  return(z);
}

getdesignR <- function(dMat, cMat, theta){
  K <- matrix(0, nr=dim(dMat)[1], nc=dim(cMat)[1]);
  for (i in 1:dim(dMat)[1]){
    for (j in 1:dim(cMat)[1]){
      K[i, j] <- exp(-sum(theta$L*(dMat[i,] - cMat[j,])^2));
    }
  }
  K <- cbind(1, K);
  K <- K[, theta$nvec>0];
  return(K);
}

test <- function(n, p, d, total){
  dMat <- matrix(rnorm(n*d), nr=n);
  cMat <- matrix(rnorm(p*d), nr=p);
  L <- rexp(d);
  nvec <- rmultinom(1, total, rep(1, p+1)/(p+1)); # (p+1)*1
  theta <- list(L=L, nvec=nvec);

  rc <- getdesign(dMat, cMat, theta);
  rr <- getdesignR(dMat, cMat, theta);
  print(max(abs(rc-rr))); #small is good.
}
```

We also need to provide a minimal documentation for this package, called

```
//myk/man/myk-package.Rd
```

with contents

```
\name{myk-package}
\alias{myk-package}
\alias{myk}
\docType{package}
\title{
  Kernel Example
}
\description{
  Illustrate how to use C code in building a R package
}
\details{
\tabular{ll}{
Package: \tab myk\cr
Type: \tab Package\cr
Version: \tab 1.0\cr
Date: \tab 2008-07-13\cr
License: \tab What license is it under?\cr
LazyLoad: \tab yes\cr
}
Overview:
Functions:
getdesign()
test()
}
\author{
  eggstone

Maintainer: Yourself <yourfault@somewhere.net>
}
\references{
  Literature or other references for background information
}
  Optionally other standard keywords, one per line, from file KEYWORDS in
  the R documentation directory
\keyword{ kernel }

\examples{
  test(100, 150, 1, 2)
}
```

The remaining files to specify is the DESCRIPTION and NAMESPACE. The DESCRIPTION is easy, and nothing new here:

```
Package: myk
Type: Package
Title: package of kernel function
Version: 1.0
Date: 2008-07-13
Author: Zhi
Maintainer: Zhi <here@email.com>
Description: Example of package using C code
License: GPL-2
LazyLoad: yes
```

While we do have something new in the `NAMESPACE`, specifying how to load the shared library:

```
useDynLib(myk)
export(getdesign, test)
```

Finally, install this package

```
[zo2@ocean simple]$ R CMD INSTALL myk
* Installing to library '/home/grad/zo2/.Rlibs'
* Installing *source* package 'myk' ...
** libs
g++ -I/usr/lib64/R/include -I/usr/local/include
-fpic -O3 -g -c kernelCalculation.cpp -o kernelCalculation.o
g++ -shared -Bdirect,--hash-stype=both,-Wl,-O1 -o
myk.so kernelCalculation.o -L/usr/lib64/R/lib -lR
** R
** preparing package for lazy loading
** help
>>> Building/Updating help pages for package 'myk'
      Formats: text html latex example
** building package indices ...
* DONE (myk)
```

And we can check how it works in R

```
> library("myk")
> test(100, 150, 1, 2)
[1] 1.110223e-16
```