```
    else{
      warning(paste("No intercept term",
        "detected. Uncentered VIFs computed."))
      return(Uncentered.VIF = uc.struct)
    }
  }
```

The user of such a function should allow greater critical values for the uncentered than for the centered VIFs.

Bibliography

D. A. Belsley (1991). *Conditioning Diagnostics. Collinearity and Weak Data in Regression*. Wiley, New York. 14

J. Fox (1997). *Applied Regression, Linear Models, and Related Methods*. Sage Publications, Thousand Oaks. 13

J. Fox (2002). *An R and S-Plus Companion to Applied Regression*. Sage Publications, Thousand Oaks. 13

W. N. Venables (2002). *[R] RE: [S] VIF Variance Inflation Factor*. http://www.R-project.org/nocvs/mail/r-help/2002/8566.html. 13

Jürgen Groß
University of Dortmund
Vogelpothsweg 87
D-44221 Dortmund, Germany
gross@statistik.uni-dortmund.de

Building Microsoft Windows Versions of R and R packages under Intel Linux

A Package Developer's Tool

by Jun Yan and A.J. Rossini

It is simple to build R and R packages for Microsoft Windows from an ix86 Linux machine. This is very useful to package developers who are familiar with Unix tools and feel that widespread dissemination of their work is important. The resulting R binaries and binary library packages require only a minimal amount of work to install under Microsoft Windows. While testing is always important for quality assurance and control, we have found that the procedure usually results in reliable packages. This document provides instructions for obtaining, installing, and using the cross-compilation tools.

These steps have been put into a Makefile, which accompanies this document, for automating the process. The Makefile is available from the contributed documentation area on Comprehensive R Archive Network (CRAN). The current version has been tested with R-1.7.0.

For the impatient and trusting, if a current version of Linux R is in the search path, then

```
make CrossCompileBuild
```

will run all Makefile targets described up to the section, *Building R Packages and Bundles*. This assumes: (1) you have the Makefile in a directory `RCrossBuild` (empty except for the makefile), and (2) that `./RCrossBuild` is your current working directory. After this, one should manually set up the packages of interest for building, though the makefile will still help with many important steps. We describe this in detail in the section on *Building R Packages and Bundles*.

Setting up the build area

We first create a separate directory for R cross-building, say `RCrossBuild` and will put the Makefile into this directory. From now on this directory is stored in the environment variable $RCB. Make sure the file name is `Makefile`, unless you are familiar with using the **make** program with other control file names.

Create work area

The following steps should take place in the `RCrossBuild` directory. Within this directory, the Makefile assumes the following subdirectories either exist or can be created, for use as described:

- `downloads` is the location to store all the sources needed for cross-building.
- `cross-tools` is the location to unpack the cross-building tools.
- `WinR` is the location to unpack the R source and cross-build R.
- `LinuxR` is the location to unpack the R source and build a fresh Linux R, which is only needed if your system doesn't have the current version of R.

- `pkgsrc` is the location of package sources (tarred and gzipped from `R CMD build`) that need to cross-build.

- `WinRlibs` is the location to put the resulting Windows binaries of the packages.

These directories can be changed by modifying the `Makefile`. One may find what exactly each step does by looking into the `Makefile`.

Download tools and sources

```
make down
```

This command downloads the i386-mingw32 cross-compilers and R source (starting from R-1.7.0, other sources that used to be necessary for cross building, `pcre` and `bzip2`, are no longer needed). It places all sources into a separate directory called `RCrossBuild/downloads`. The **wget** program is used to get all needed sources. Other downloading tools such as **curl** or **links/elinks** can be used as well. We place these sources into the `RCrossBuild/downloads` directory. The URLs of these sources are available in file `src/gnuwin32/INSTALL` which is located within the R source tree (or, see the `Makefile` associated with this document).

Cross-tools setup

```
make xtools
```

This rule creates a separate subdirectory `cross-tools` in the build area for the cross-tools. It unpacks the cross-compiler tools into the `cross-tools` subdirectory.

Prepare source code

```
make prepsrc
```

This rule creates a separate subdirectory `WinR` to carry out the cross-building process of R and unpacks the R sources into it. As of 1.7.0, the source for `pcre` and `bzip2` are included in the R source tar archive; before that, we needed to worry about them.

Build Linux R if needed

```
make LinuxR
```

This step is required, as of R-1.7.0, if you don't have a current version of Linux R on your system and you don't have the permission to install one for system wide use. This rule will build and install a Linux R in `$RCB/LinuxR/R`.

Building R

Configuring

If a current Linux R is available from the system and on your search path, then run the following command:

```
make mkrules
```

This rule modifies the file `src/gnuwin32/Mkrules` from the R source tree to set `BUILD=CROSS` and `HEADER=$RCB/cross-tools/mingw32/include`.

If a current Linux R is not available from the system, and a Linux R has just been built by `make LinuxR` from the end of the last section:

```
make LinuxFresh=YES mkrules
```

This rule will set `R_EXE=$RCB/LinuxR/R/bin/R`, in addition to the variables above.

Compiling

Now we can cross-compile R:

```
make R
```

The environment variable $PATH is modified in this make target to ensure that the cross-tools are at the beginning of the search path. This step as well as initiation of the compile process is accomplished by the `R` makefile target. This may take a while to finish. If everything goes smoothly, a compressed file `Win-R-1.7.0.tgz` will be created in the `RCrossBuild/WinR` directory.

This gzip'd tar-file contains the executables and supporting files for R which will run on a Microsoft Windows machine. To install, transfer and unpack it on the desired machine. Since there is no InstallShield-style installer executable, one will have to manually create any desired desktop shortcuts to the executables in the `bin` directory, such as `Rgui.exe`. Remember, though, this isn't necessarily the same as the R for Windows version on CRAN!

Building R packages and bundles

Now we have reached the point of interest. As one might recall, the primary goal of this document is to be able to build binary packages for R libraries which can be simply installed for R running on Microsoft Windows. All the work up to this point simply obtains the required working build of R for Microsoft Windows!

Now, create the `pkgsrc` subdirectory to put the package sources into and the `WinRlibs` subdirectory to put the windows binaries into.

We will use the `geepack` package as an example for cross-building. First, put the source `geepack_0.2-4.tar.gz` into the subdirectory `pkgsrc`, and then do

```
make pkg-geepack_0.2-4
```

If there is no error, the Windows binary `geepack.zip` will be created in the `WinRlibs` subdirectory, which is then ready to be shipped to a Windows machine for installation.

We can easily build bundled packages as well. For example, to build the packages in bundle `VR`, we place the source `VR_7.0-11.tar.gz` into the `pkgsrc` subdirectory, and then do

```
make bundle-VR_7.0-11
```

The Windows binaries of packages `MASS`, `class`, `nnet`, and `spatial` in the `VR` bundle will appear in the `WinRlibs` subdirectory.

This Makefile assumes a tarred and gzipped source for an R package, which ends with "`.tar.gz`". This is usually created through the `R CMD build` command. It takes the version number together with the package name.

The Makefile

The associated Makefile is used to automate many of the steps. Since many Linux distributions come with the **make** utility as part of their installation, it hopefully will help rather than confuse people cross-compiling R. The Makefile is written in a format similar to shell commands in order to show what exactly each step does.

The commands can also be cut-and-pasted out of the Makefile with minor modification (such as, change $$ to $ for environmental variable names), and run manually.

Possible pitfalls

We have very little experience with cross-building packages (for instance, Matrix) that depend on external libraries such as atlas, blas, lapack, or Java libraries. Native Windows building, or at least a substantial amount of testing, may be required in these cases. It is worth experimenting, though!

Acknowledgments

We are grateful to Ben Bolker, Stephen Eglen, and Brian Ripley for helpful discussions.

Jun Yan
University of Wisconsin–Madison, U.S.A.
jyan@stat.wisc.edu

A.J. Rossini
University of Washington, U.S.A.
rossini@u.washington.edu

Analysing Survey Data in R

by Thomas Lumley

Introduction

Survey statistics has always been a somewhat specialised area due in part to its view of the world. In the rest of the statistical world data are random and we model their distributions. In traditional survey statistics the population data are fixed and only the sampling is random. The advantage of this 'design-based' approach is that the sampling, unlike the population, is under the researcher's control.

The basic question of survey statistics is

> If we did exactly this analysis on the whole population, what result would we get?

If individuals were sampled independently with equal probability the answer would be very straightforward. They aren't, and it isn't.

To simplify the operations of a large survey it is routine to sample in clusters. For example, a random sample of towns or counties can be chosen and then individuals or families sampled from within those areas. There can be multiple levels of cluster sampling, eg, individuals within families within towns. Cluster sampling often results in people having an unequal chance of being included, for example, the same number of individuals might be surveyed regardless of the size of the town.

For statistical efficiency and to make the results more convincing it is also usual to stratify the population and sample a prespecified number of individuals or clusters from each stratum, ensuring that all strata are fairly represented. Strata for a national survey might divide the country into geographical regions and then subdivide into rural and urban. Ensuring that smaller strata are well represented may also involve sampling with unequal probabilities.

Finally, unequal probabilities might be deliberately used to increase the representation of groups that are small or particularly important. In US health surveys there is often interest in the health of the poor and of racial and ethnic minority groups, who might thus be oversampled.

The resulting sample may be very different in