UseInR

Joseph Lizier edited this page on 20 Apr · 3 revisions

How to use the toolkit in R

Introduction

The java code from this toolkit can easily be used in R. First we describe calling Java code from R, then specifically describe the use of this toolkit.

Several longer examples of using the toolkit in R can be viewed at R_Examples.

Using Java objects in R

rJava offers a standard R package for an R to Java interface. It can easily be installed from CRAN (Comprehensive R Archive Network), and is also packaged for ubuntu (my platform of choice) as `r-cran-rjava` . Further details on rJava are available at http://cran.r-project.org/web/packages/rJava/rJava.pdf

As such, we only describe using JIDT in R via rJava here, though it may be possible with other mechanisms.

Using rJava

The first step is to install rJava. This may be done by either:

1. Running `install.packages("rJava")` inside R; or
2. Installing rJava via your OS package manager: for me on ubuntu, this was `sudo apt-get install r-cran-rjava` . Getting rJava working on ubuntu 12.04 only worked for me via the ubuntu package manager (apparently the installation via the ubuntu package manager solves some issues that installing from R does not - see here for more details).

One or more post installation steps may then be required: for me on ubuntu, this was to run `sudo R CMD javareconf` in my shell (without having done this, I experienced some errors on trying to load rJava in R). I did not need to clear JAVA_HOME as some pages suggest. See further installation notes at http://www.rforge.net/rJava/

You can then run your Java code fairly simply in R with rJava:

1. Load the rJava library: `library("rJava")`
2. Start the JVM: `.jinit()` -- if you are seeing OutOfMemoryError though you likely need to allocate more memory to the JVM by changing this line to e.g. `.jinit(parameters="-Xmx512m")` to allocate 512 MB.
3. Tell the JVM where our `infodynamics.jar` file is, using e.g. `.jaddClassPath("PATH/infodynamics.jar")`. Note that if you wish to use a relative path to the jar you will need to have set the working directory in your R environment, either by starting R from the desired working directory, or setting it with `setwd()` or via your R GUI.
4. Create an instance of the calculator you wish to use, e.g. `teCalc<-.jnew("infodynamics/measures/discrete/TransferEntropyCalculatorDiscrete", 2L, 1L)`. Notice first that the separator for package names to be used here is `/` rather than `.` as is normally used in Java. Next, notice that numeric integer arguments must have the `L` suffix to explicitly cast them as integers (otherwise R casts these as doubles and cannot locate the Java method with the matching signature).
5. Call methods on the object, e.g. `.jcall(teCalc,"V","addObservations",sourceArray, destArray)`. Notice here how the second argument to the `.jcall` method specifies the return type for the method (here `"V"` for void) -- see the full list of these at http://www.rforge.net/rJava/ including how to specify this when the return type is a Java class (see e.g. example 3 at R_Examples).
6. Access attributes of the object directly, e.g. `nullDist$pValue` in R to access what in Java would be `nullDist.pValue`.

Array conversion is provided by the rJava package as well:

1. Single dimensional arrays (or lists in R) can generally simply be passed directly into the java method (e.g. see example 1 at R_Examples). An exception here is where R may confuse the type of the array as a double instead of integer; in this case you should make an explicit array conversion such as `.jarray(list1, "[I")` where the `"[I"` specifies conversion to an int array.
2. For multidimensional arrays, explicit conversion should be performed via `.jarray` with the `dispatch=TRUE` argument, e.g. `.jarray(twoDList, "[dispatch=TRUE)` (e.g. see example 2 and example 5 at R_Examples).

3. For `data.frame` objects in R, be aware that an extra step is involved in order to extract the raw data first before converting to a Java array; see Example 6 at R_Examples
4. To convert Java arrays (e.g. method results) back into R format, use `.jevalArray(javaArray)`, adding an extra argument i.e. `.jevalArray(javaArray, simplify=TRUE)` for 2D arrays (see example 4 at R_Examples).

See further details here.

Pages 38

- Home
- Getting started
  - Downloads
  - Installation
  - Documentation
  - Tutorial
  - Demos
- ImplementedMeasures
- Demos
  - Auto analyser demo
  - Simple Java demos
  - Non-Java environments
    - Matlab/Octave demos
    - Python demos
    - R demos
    - Julia demos
    - Clojure demos
  - GPU
  - Cellular Automata
  - Schreiber Transfer entropy demos
  - Flocking/Swarming
  - Detecting interaction lags
  - Null distributions
  - Interregional transfer
- Course (long)
- Tutorial (short)
- Non-Java environments
  - Matlab/Octave
    - Array conversion to/from Octave
  - Python
  - R
  - Julia
  - Clojure

- FAQs
- Miscellaneous
  - Related toolkits
  - Road map for new features
  - Extra features
- For serious developers!
  - Unit tests
  - Ant scripts
  - Making a new release
- Publications resulting

Clone this wiki locally

```
https://github.com/jlizier/jidt.wiki.git
```