information-dynamics-toolkit

JIDT: Java Information Dynamics Toolkit for studying information-theoretic measures of computation in complex systems

R_Examples
Examples of using the toolkit in R
R, examples

Updated May 27, 2015 by joseph.lizier

Demos > R code examples

R code examples

This page describes a basic set of demonstration scripts for using the toolkit in R. The .r files can be found at demos/r in the distributions (from the V1.1 release onwards).

Please see UseInR for instructions on how to begin using the java toolkit from inside R.

Note that these examples use the rJava R library -- you will need to alter them if you want to use another R-Java interface (though I believe this is the standard one).

This page contains the following code examples:

Example 1 - Transfer entropy on binary data

example1TeBinaryData.r - Simple transfer entropy (TE) calculation on binary data using the discrete TE calculator:

```
# Load the rJava library and start the JVM
library("rJava")
.jinit()

# Change location of jar to match yours:
#  IMPORTANT -- If using the default below, make sure you have set the working directory
#   in R (e.g. with setwd()) to the location of this file (i.e. demos/r) !!
.jaddClassPath("../../infodynamics.jar")

# Generate some random binary data:
sourceArray<-sample(0:1, 100, replace=TRUE)
destArray<-c(0L, sourceArray[1:99]); # Need 0L to keep as integer array
sourceArray2<-sample(0:1, 100, replace=TRUE)

# Create a TE calculator and run it:
teCalc<-.jnew("infodynamics/measures/discrete/TransferEntropyCalculatorDiscrete", 2L, 1L)
.jcall(teCalc,"V","initialise") # V for void return value
.jcall(teCalc,"V","addObservations",sourceArray, destArray)

result1 <- .jcall(teCalc,"D","computeAverageLocalOfObservations")
cat("For copied source, result should be close to 1 bit : ", result1, "\n")

# Now look at the unrelated source:
.jcall(teCalc,"V","initialise") # V for void return value
.jcall(teCalc,"V","addObservations",sourceArray2, destArray)
result2 <- .jcall(teCalc,"D","computeAverageLocalOfObservations")
cat("For random source, result should be close to 0 bits: ", result2, "\n")
```

Example 2 - Transfer entropy on multidimensional binary data

example2TeMultidimBinaryData.r - Simple transfer entropy (TE) calculation on multidimensional binary data using the discrete TE calculator.

This example is important for R users, because it shows how to handle multidimensional arrays from R to Java (this is not as simple as single dimensional arrays in example 1 - it requires using extra calls to convert the array).

```
# Load the rJava library and start the JVM
library("rJava")
.jinit()

# Change location of jar to match yours:
#  IMPORTANT -- If using the default below, make sure you have set the working directory
#   in R (e.g. with setwd()) to the location of this file (i.e. demos/r) !!
.jaddClassPath("../../infodynamics.jar")

# Create many columns in a multidimensional array (2 rows by 100 columns),
#  where the next time step (row 2) copies the value of the column on the left
#  from the previous time step (row 1):
twoDTimeSeriesRtime1 <- sample(0:1, 100, replace=TRUE)
twoDTimeSeriesRtime2 <- c(twoDTimeSeriesRtime1[100], twoDTimeSeriesRtime1[1:99])
twoDTimeSeriesR <- rbind(twoDTimeSeriesRtime1, twoDTimeSeriesRtime2)

# Create a TE calculator and run it:
teCalc<-.jnew("infodynamics/measures/discrete/TransferEntropyCalculatorDiscrete", 2L, 1L)
.jcall(teCalc,"V","initialise") # V for void return value
# Add observations of transfer across one cell to the right per time step:
twoDTimeSeriesJava <- .jarray(twoDTimeSeriesR, "[I", dispatch=TRUE)
.jcall(teCalc,"V","addObservations", twoDTimeSeriesJava, 1L)
result2D <- .jcall(teCalc,"D","computeAverageLocalOfObservations")
cat("The result should be close to 1 bit here, since we are executing copy operations of what is effectively a random
```

Example 3 - Transfer entropy on continuous data using kernel estimators

example3TeContinuousDataKernel.r - Simple transfer entropy (TE) calculation on continuous-valued data using the (box) kernel-estimator TE calculator.

```
# Load the rJava library and start the JVM
library("rJava")
.jinit()

# Change location of jar to match yours:
#  IMPORTANT -- If using the default below, make sure you have set the working directory
#   in R (e.g. with setwd()) to the location of this file (i.e. demos/r) !!
.jaddClassPath("../../infodynamics.jar")

# Generate some random normalised data.
numObservations<-1000
covariance<-0.4
sourceArray<-rnorm(numObservations)
destArray = c(0, covariance*sourceArray[1:numObservations-1] + (1-covariance)*rnorm(numObservations-1, 0, 1))
sourceArray2<-rnorm(numObservations) # Uncorrelated source

# Create a TE calculator and run it:
teCalc<-.jnew("infodynamics/measures/continuous/kernel/TransferEntropyCalculatorKernel")
.jcall(teCalc,"V","setProperty", "NORMALISE", "true") # Normalise the individual variables
.jcall(teCalc,"V","initialise", 1L, 0.5) # Use history length 1 (Schreiber k=1), kernel width of 0.5 normalised units
.jcall(teCalc,"V","setObservations", sourceArray, destArray)
# For copied source, should give something close to expected value for correlated Gaussians:
result <- .jcall(teCalc,"D","computeAverageLocalOfObservations")
cat("TE result ",  result, "bits; expected to be close to ", log(1/(1-covariance^2))/log(2), " bits for these correla

.jcall(teCalc,"V","initialise") # Initialise leaving the parameters the same
.jcall(teCalc,"V","setObservations", sourceArray2, destArray)
# For random source, it should give something close to 0 bits
result2 <- .jcall(teCalc,"D","computeAverageLocalOfObservations")
cat("TE result ",  result2, "bits; expected to be close to 0 bits for uncorrelated Gaussians but will be biased upwar

# We can get insight into the bias by examining the null distribution:
nullDist <- .jcall(teCalc,"Linfodynamics/utils/EmpiricalMeasurementDistribution;",
                "computeSignificance", 100L)
cat("Null distribution for unrelated source and destination",
    "(i.e. the bias) has mean", .jcall(nullDist, "D", "getMeanOfDistribution"),
    "bits and standard deviation", .jcall(nullDist, "D", "getStdOfDistribution"), "\n")
```

Example 4 - Transfer entropy on continuous data using Kraskov estimators

example4TeContinuousDataKraskov.r - Simple transfer entropy (TE) calculation on continuous-valued data using the Kraskov-estimator TE calculator.

```
# Load the rJava library and start the JVM
library("rJava")
.jinit()

# Change location of jar to match yours:
#  IMPORTANT -- If using the default below, make sure you have set the working directory
```

```
#   in R (e.g. with setwd()) to the location of this file (i.e. demos/r) !!
.jaddClassPath("../../infodynamics.jar")

# Generate some random normalised data.
numObservations<-1000
covariance<-0.4
sourceArray<-rnorm(numObservations)
destArray = c(0, covariance*sourceArray[1:numObservations-1] + (1-covariance)*rnorm(numObservations-1, 0, 1))
sourceArray2<-rnorm(numObservations) # Uncorrelated source

# Create a TE calculator:
teCalc<-.jnew("infodynamics/measures/continuous/kraskov/TransferEntropyCalculatorKraskov")
.jcall(teCalc,"V","setProperty", "k", "4") # Use Kraskov parameter K=4 for 4 nearest points

# Perform calculation with correlated source:
.jcall(teCalc,"V","initialise", 1L) # Use history length 1 (Schreiber k=1)
.jcall(teCalc,"V","setObservations", sourceArray, destArray)
result <- .jcall(teCalc,"D","computeAverageLocalOfObservations")
# Note that the calculation is a random variable (because the generated
#  data is a set of random variables) - the result will be of the order
#  of what we expect, but not exactly equal to it; in fact, there will
#  be a large variance around it.
cat("TE result ",  result, "nats; expected to be close to ", log(1/(1-covariance^2)), " nats for these correlated Gau

# Perform calculation with uncorrelated source:
.jcall(teCalc,"V","initialise") # Initialise leaving the parameters the same
.jcall(teCalc,"V","setObservations", sourceArray2, destArray)
result2 <- .jcall(teCalc,"D","computeAverageLocalOfObservations")
cat("TE result ",  result2, "nats; expected to be close to 0 nats for uncorrelated Gaussians\n")

# We can also compute the local TE values for the time-series samples here:
#  (See more about utility of local TE in the CA demos)
localTE <- .jcall(teCalc,"[D","computeLocalOfPreviousObservations")
cat("Notice that the mean of locals", sum(localTE)/(numObservations-1),
        "nats equals the above result\n")
```

Example 5 - Multivariate transfer entropy on binary data

example5TeBinaryMultivarTransfer.r - Multivariate transfer entropy (TE) calculation on binary data using the discrete TE calculator.

```
# Load the rJava library and start the JVM
library("rJava")
.jinit()

# Change location of jar to match yours:
#  IMPORTANT -- If using the default below, make sure you have set the working directory
#   in R (e.g. with setwd()) to the location of this file (i.e. demos/r) !!
.jaddClassPath("../../infodynamics.jar")

# Generate some random binary data.
numObservations <- 100
sourceArray<-matrix(sample(0:1,numObservations*2, replace=TRUE),numObservations,2)
sourceArray2<-matrix(sample(0:1,numObservations*2, replace=TRUE),numObservations,2)
# Destination variable takes a copy of the first bit of the source in bit 1,
#  and an XOR of the two bits of the source in bit 2:
destArray <- cbind( c(0L, sourceArray[1:numObservations-1,1]), # column 1
                    c(0L, 1L*xor(sourceArray[1:numObservations-1,1],
                                 sourceArray[1:numObservations-1,2]))) # column 2

# Convert the 2D arrays to Java format:
sourceArrayJava <- .jarray(sourceArray, "[I", dispatch=TRUE)
sourceArray2Java <- .jarray(sourceArray2, "[I", dispatch=TRUE)
destArrayJava <- .jarray(destArray, "[I", dispatch=TRUE)

# Create a TE calculator and run it:
teCalc<-.jnew("infodynamics/measures/discrete/TransferEntropyCalculatorDiscrete", 4L, 1L)
.jcall(teCalc,"V","initialise") # V for void return value
# We need to construct the joint values for the dest and source before we pass them in,
#  and need to use the matrix conversion routine when calling from Matlab/Octave:
mUtils<-.jnew("infodynamics/utils/MatrixUtils")
.jcall(teCalc,"V","addObservations",
                .jcall(mUtils,"[I","computeCombinedValues", sourceArrayJava, 2L),
                .jcall(mUtils,"[I","computeCombinedValues", destArrayJava, 2L))
result<-.jcall(teCalc,"D","computeAverageLocalOfObservations")
cat("For source which the 2 bits are determined from, result should be close to 2 bits : ", result, "\n")

.jcall(teCalc,"V","initialise")
.jcall(teCalc,"V","addObservations",
                .jcall(mUtils,"[I","computeCombinedValues", sourceArray2Java, 2L),
                .jcall(mUtils,"[I","computeCombinedValues", destArrayJava, 2L))
result2<-.jcall(teCalc,"D","computeAverageLocalOfObservations")
cat("For random source, result should be close to 0 bits in theory: ", result2, "\n");
cat("Result for random source is inflated towards 0.3 due to finite observation length ",
    .jcall(teCalc,"I","getNumObservations"), "\n",
```

```
    "One can verify that the answer is consistent with that from a\n",
    "random source by checking: teCalc.computeSignificance(1000); ans.pValue\n");
```

Example 6 - Dynamic dispatch with Mutual info calculator

example6DynamicCallingMutualInfo.r - This example shows how to write R code to take advantage of the common interfaces defined for various information-theoretic calculators. Here, we use the common form of the `infodynamics.measures.continuous.MutualInfoCalculatorMultiVariate` interface (which is never named here) to write common code into which we can plug one of three concrete implementations (kernel estimator, Kraskov estimator or linear-Gaussian estimator) by dynamically supplying the class name of the concrete implementation.

```
# Load the rJava library and start the JVM
library("rJava")
.jinit()

# Change location of jar to match yours:
#  IMPORTANT -- If using the default below, make sure you have set the working directory
#   in R (e.g. with setwd()) to the location of this file (i.e. demos/r) !!
.jaddClassPath("../../infodynamics.jar")

#---------------------
# 1. Properties for the calculation (these are dynamically changeable, you could
#    load them in from another properties file):
# The name of the data file (relative to this directory)
datafile <- "../data/4ColsPairedNoisyDependence-1.txt"
# List of column numbers for variables 1 and 2:
#  (you can select any columns you wish to be contained in each variable)
variable1Columns <- c(1,2) # array indices start from 1 in R
variable2Columns <- c(3,4)
# The name of the concrete implementation of the interface
#  infodynamics.measures.continuous.MutualInfoCalculatorMultiVariate
#  which we wish to use for the calculation.
# Note that one could use any of the following calculators (try them all!):
#  implementingClass <- "infodynamics/measures/continuous/kraskov/MutualInfoCalculatorMultiVariateKraskov1" # MI([1,2
#  implementingClass <- "infodynamics/measures/continuous/kernel/MutualInfoCalculatorMultiVariateKernel"
#  implementingClass <- "infodynamics/measures/continuous/gaussian/MutualInfoCalculatorMultiVariateGaussian"
implementingClass <- "infodynamics/measures/continuous/kraskov/MutualInfoCalculatorMultiVariateKraskov1"

#---------------------
# 2. Load in the data
data <- read.csv(datafile, header=FALSE, sep="")
# Pull out the columns from the data set which correspond to each of variable 1 and 2:
variable1 <- data[, variable1Columns]
variable2 <- data[, variable2Columns]
# Extra step to extract the raw values from these data.frame objects:
variable1 <- apply(variable1, 2, function(x) as.numeric(x))
variable2 <- apply(variable2, 2, function(x) as.numeric(x))

#---------------------
# 3. Dynamically instantiate an object of the given class:
# (in fact, all java object creation in octave/matlab is dynamic - it has to be,
#  since the languages are interpreted. This makes our life slightly easier at this
#  point than it is in demos/java/example6LateBindingMutualInfo where we have to handle this manually)
miCalc<-.jnew(implementingClass)

#---------------------
# 4. Start using the MI calculator, paying attention to only
#  call common methods defined in the interface type
#  infodynamics.measures.continuous.MutualInfoCalculatorMultiVariate
#  not methods only defined in a given implementation class.
# a. Initialise the calculator to use the required number of
#   dimensions for each variable:
.jcall(miCalc,"V","initialise", length(variable1Columns), length(variable2Columns))
# b. Supply the observations to compute the PDFs from:
.jcall(miCalc,"V","setObservations",
        .jarray(variable1, "[D", dispatch=TRUE),
        .jarray(variable2, "[D", dispatch=TRUE))
# c. Make the MI calculation:
miValue <- .jcall(miCalc,"D","computeAverageLocalOfObservations")

cat("MI calculator", implementingClass, "\n computed the joint MI as ",
        miValue, "\n")
```