information-dynamics-toolkit

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

PythonExamples

Examples of using the toolkit in Python

python, Phase-Deploy, examples, Featured

Updated Oct 2, 2014 by joseph.lizier

Demos > Python code examples

Python code examples

This page describes a basic set of demonstration scripts for using the toolkit in Python. The .py files can be found at demos/python in the svn or main distributions. We plan to have other more complicated examples available from the main Demos page in future.

Please see UseInPython for instructions on how to begin using the java toolkit from inside python.

Note that these examples use JPype -- you will need to alter them if you want to use another Python-Java interface.

This page contains the following code examples:

Example 1 - Transfer entropy on binary data

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

```
from jpype import *
import random

# Change location of jar to match yours:
jarLocation = "../../infodynamics.jar"
# Start the JVM (add the "-Xmx" option with say 1024M if you get crashes due to not enough memory space)
startJVM(getDefaultJVMPath(), "-ea", "-Djava.class.path=" + jarLocation)

# Generate some random binary data.
sourceArray = [random.randint(0,1) for r in xrange(100)]
destArray = [0] + sourceArray[0:99];
sourceArray2 = [random.randint(0,1) for r in xrange(100)]

# Create a TE calculator and run it:
teCalcClass = JPackage("infodynamics.measures.discrete").TransferEntropyCalculatorDiscrete
teCalc = teCalcClass(2,1)
teCalc.initialise()
# Since we have simple arrays of ints, we can directly pass these in:
teCalc.addObservations(sourceArray, destArray)
print("For copied source, result should be close to 1 bit : %.4f" % teCalc.computeAverageLocalOfObservations())
teCalc.initialise()
teCalc.addObservations(sourceArray2, destArray)
print("For random source, result should be close to 0 bits: %.4f" % teCalc.computeAverageLocalOfObservations())

shutdownJVM()
```

Example 2 - Transfer entropy on multidimensional binary data

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

This example is important for Python JPype users, because it shows how to handle multidimensional arrays from Python to Java.

```
from jpype import *
import random

# Change location of jar to match yours:
jarLocation = "../../infodynamics.jar"
```

```
# Start the JVM (add the "-Xmx" option with say 1024M if you get crashes due to not enough memory space)
startJVM(getDefaultJVMPath(), "-ea", "-Djava.class.path=" + jarLocation)

# Create many columns in a multidimensional array, e.g. for fully random values:
# twoDTimeSeriesOctave = [[random.randint(0,1) for y in xrange(2)] for x in xrange(10)] # for 10 rows (time-steps) fo

# However here we want 2 rows by 100 columns where the next time step (row 2) is to copy the
# value of the column on the left from the previous time step (row 1):
numObservations = 100
row1 = [random.randint(0,1) for r in xrange(numObservations)]
row2 = [row1[numObservations-1]] + row1[0:numObservations-1] # Copy the previous row, offset one column to the right
twoDTimeSeriesPython = []
twoDTimeSeriesPython.append(row1)
twoDTimeSeriesPython.append(row2)
twoDTimeSeriesJavaInt = JArray(JInt, 2)(twoDTimeSeriesPython); # 2 indicating 2D array

# Create a TE calculator and run it:
teCalcClass = JPackage("infodynamics.measures.discrete").TransferEntropyCalculatorDiscrete
teCalc = teCalcClass(2,1)
teCalc.initialise()
# Add observations of transfer across one cell to the right per time step:
teCalc.addObservations(twoDTimeSeriesJavaInt, 1)
result2D = teCalc.computeAverageLocalOfObservations()
print(('The result should be close to 1 bit here, since we are executing copy ' + \
      'operations of what is effectively a random bit to each cell here: %.3f ' + \
      'bits from %d observations') % (result2D, teCalc.getNumObservations()))
```

Example 3 - Transfer entropy on continuous data using kernel estimators

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

```
from jpype import *
import random
import math

# Change location of jar to match yours:
jarLocation = "../../infodynamics.jar"
# Start the JVM (add the "-Xmx" option with say 1024M if you get crashes due to not enough memory space)
startJVM(getDefaultJVMPath(), "-ea", "-Djava.class.path=" + jarLocation)

# Generate some random normalised data.
numObservations = 1000;
covariance=0.4;
# Source array of random normals:
sourceArray = [random.normalvariate(0,1) for r in xrange(numObservations)];
# Destination array of random normals with partial correlation to previous value of sourceArray
destArray = [0] + [sum(pair) for pair in zip([covariance*y for y in sourceArray[0:numObservations-1]], \
                                              [(1-covariance)*y for y in [random.normalvariate(0,1) for r in xrange(nu
# Uncorrelated source array:
sourceArray2 = [random.normalvariate(0,1) for r in xrange(numObservations)];
# Create a TE calculator and run it:
teCalcClass = JPackage("infodynamics.measures.continuous.kernel").TransferEntropyCalculatorKernel
teCalc = teCalcClass();
teCalc.setProperty("NORMALISE", "true"); # Normalise the individual variables
teCalc.initialise(1, 0.5); # Use history length 1 (Schreiber k=1), kernel width of 0.5 normalised units
teCalc.setObservations(JArray(JDouble, 1)(sourceArray), JArray(JDouble, 1)(destArray));
# For copied source, should give something close to 1 bit:
result = teCalc.computeAverageLocalOfObservations();
print("TE result %.4f bits; expected to be close to %.4f bits for these correlated Gaussians but biased upwards" % \
    (result, math.log(1/(1-math.pow(covariance,2)))/math.log(2)));
teCalc.initialise(); # Initialise leaving the parameters the same
teCalc.setObservations(JArray(JDouble, 1)(sourceArray2), JArray(JDouble, 1)(destArray));
# For random source, it should give something close to 0 bits
result2 = teCalc.computeAverageLocalOfObservations();
print("TE result %.4f bits; expected to be close to 0 bits for uncorrelated Gaussians but will be biased upwards" % \
    result2);
```

Example 4 - Transfer entropy on continuous data using Kraskov estimators

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

```
from jpype import *
import random
import math

# Change location of jar to match yours:
jarLocation = "../../infodynamics.jar"
# Start the JVM (add the "-Xmx" option with say 1024M if you get crashes due to not enough memory space)
startJVM(getDefaultJVMPath(), "-ea", "-Djava.class.path=" + jarLocation)
```

```
# Generate some random normalised data.
numObservations = 1000;
covariance=0.4;
# Source array of random normals:
sourceArray = [random.normalvariate(0,1) for r in xrange(numObservations)];
# Destination array of random normals with partial correlation to previous value of sourceArray
destArray = [0] + [sum(pair) for pair in zip([covariance*y for y in sourceArray[0:numObservations-1]], \
                                              [(1-covariance)*y for y in [random.normalvariate(0,1) for r in xrange(nu
# Uncorrelated source array:
sourceArray2 = [random.normalvariate(0,1) for r in xrange(numObservations)];
# Create a TE calculator and run it:
teCalcClass = JPackage("infodynamics.measures.continuous.kraskov").TransferEntropyCalculatorKraskov
teCalc = teCalcClass();
teCalc.setProperty("NORMALISE", "true"); # Normalise the individual variables
teCalc.initialise(1); # Use history length 1 (Schreiber k=1)
teCalc.setProperty("k", "4"); # Use Kraskov parameter K=4 for 4 nearest points
# Perform calculation with correlated source:
teCalc.setObservations(JArray(JDouble, 1)(sourceArray), JArray(JDouble, 1)(destArray));
result = teCalc.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.
print("TE result %.4f nats; expected to be close to %.4f nats for these correlated Gaussians" % \
    (result, math.log(1/(1-math.pow(covariance,2)))));
# Perform calculation with uncorrelated source:
teCalc.initialise(); # Initialise leaving the parameters the same
teCalc.setObservations(JArray(JDouble, 1)(sourceArray2), JArray(JDouble, 1)(destArray));
result2 = teCalc.computeAverageLocalOfObservations();
print("TE result %.4f nats; expected to be close to 0 nats for these uncorrelated Gaussians" % result2);
```

Example 5 - Multivariate transfer entropy on binary data

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

```
from jpype import *
import random
from operator import xor

# Change location of jar to match yours:
jarLocation = "../../infodynamics.jar"
# Start the JVM (add the "-Xmx" option with say 1024M if you get crashes due to not enough memory space)
startJVM(getDefaultJVMPath(), "-ea", "-Djava.class.path=" + jarLocation)

# Generate some random binary data.
numObservations = 100
sourceArray = [[random.randint(0,1) for y in xrange(2)] for x in xrange(numObservations)] # for 10 rows (time-steps)
sourceArray2= [[random.randint(0,1) for y in xrange(2)] for x in xrange(numObservations)] # for 10 rows (time-steps)
# 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 = [[0, 0]]
for j in range(1,numObservations):
        destArray.append([sourceArray[j-1][0], xor(sourceArray[j-1][0], sourceArray[j-1][1])])

# Create a TE calculator and run it:
teCalcClass = JPackage("infodynamics.measures.discrete").TransferEntropyCalculatorDiscrete
teCalc = teCalcClass(4,1)
teCalc.initialise()
# We need to construct the joint values of the dest and source before we pass them in,
#  and need to use the matrix conversion routine when calling from Matlab/Octave:
mUtils= JPackage('infodynamics.utils').MatrixUtils
teCalc.addObservations(mUtils.computeCombinedValues(sourceArray, 2), \
                mUtils.computeCombinedValues(destArray, 2))
result = teCalc.computeAverageLocalOfObservations()
print('For source which the 2 bits are determined from, result should be close to 2 bits : %.3f' % result)
teCalc.initialise()
teCalc.addObservations(mUtils.computeCombinedValues(sourceArray2, 2), \
                mUtils.computeCombinedValues(destArray, 2))
result2 = teCalc.computeAverageLocalOfObservations()
print('For random source, result should be close to 0 bits in theory: %.3f' % result2)
print('The result for random source is inflated towards 0.3 due to finite observation length (%d). One can verify tha
```

Example 6 - Dynamic dispatch with Mutual info calculator

example6DynamicCallingMutualInfo.py - This example shows how to write Python 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.

Note -- users of the v1.0 distribution will need to separately download the readFloatsFile.py module, which was accidentally not included in this release.

```
from jpype import *
```

```python
import random
import string
import numpy
import readFloatsFile

# Change location of jar to match yours:
jarLocation = "../../infodynamics.jar"
# Start the JVM (add the "-Xmx" option with say 1024M if you get crashes due to not enough memory space)
startJVM(getDefaultJVMPath(), "-ea", "-Djava.class.path=" + jarLocation)

#---------------------
# 1. Properties for the calculation (these are dynamically changeable):
# 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 = [0,1] # array indices start from 0 in python
variable2Columns = [2,3]
# 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([0,1]
#  implementingClass = "infodynamics.measures.continuous.kernel.MutualInfoCalculatorMultiVariateKernel"
#  implementingClass = "infodynamics.measures.continuous.gaussian.MutualInfoCalculatorMultiVariateGaussian"
implementingClass = "infodynamics.measures.continuous.kraskov.MutualInfoCalculatorMultiVariateKraskov1"

#---------------------
# 2. Load in the data
data = readFloatsFile.readFloatsFile(datafile)
# As numpy array:
A = numpy.array(data)
# Pull out the columns from the data set which correspond to each of variable 1 and 2:
variable1 = A[:,variable1Columns]
variable2 = A[:,variable2Columns]

#--------------------
# 3. Dynamically instantiate an object of the given class:
# (in fact, all java object creation in python 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/lateBindingDemo where we have to handle this manually)
indexOfLastDot = string.rfind(implementingClass, ".")
implementingPackage = implementingClass[:indexOfLastDot]
implementingBaseName = implementingClass[indexOfLastDot+1:]
miCalcClass = eval('JPackage(\'%s\').%s' % (implementingPackage, implementingBaseName))
miCalc = miCalcClass()

#--------------------
# 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:
miCalc.initialise(len(variable1Columns), len(variable2Columns))
# b. Supply the observations to compute the PDFs from:
miCalc.setObservations(variable1, variable2)
# c. Make the MI calculation:
miValue = miCalc.computeAverageLocalOfObservations()

print("MI calculator %s computed the joint MI as %.5f\n" % (implementingClass, miValue))
```