A short introduction to JIDT: An information-theoretic toolkit for studying the dynamics of complex systems

Joseph T. Lizier
The University of Sydney
Max Planck Institute for Mathematics in the Sciences
Leipzig
July 17, 2015
Weapon of choice

I DON'T ALWAYS ANALYSE COMPLEX SYSTEMS

BUT WHEN I DO, I USE INFORMATION THEORY
Java Information Dynamics Toolkit (JIDT)

On google code → github (https://lizier.me/jidt/)

JIDT provides a standalone, open-source (GPL v3 licensed) implementation of information-theoretic measures of information processing in complex systems, i.e. information storage, transfer and modification.

JIDT includes implementations:

- Principally for transfer entropy, mutual information, their conditional variants, active information storage etc;
- For both discrete and continuous-valued data;
- Using various types of estimators (e.g. Kraskov-Stögbauer-Grassberger, linear-Gaussian, etc.).
Java Information Dynamics Toolkit (JIDT)

JIDT is written in Java but directly usable in Matlab/Octave, Python, R, Julia, Clojure, etc.

JIDT requires almost zero installation.

JIDT is distributed with:

- A paper describing its design and usage;
- A full tutorial and exercises (this is an excerpt!)
- Full Javadocs;
- A suite of demonstrations, including in each of the languages listed above.
1 Information dynamics
   • Information theory
   • The information dynamics framework
   • Estimation techniques

2 Overview of JIDT

3 Demos

4 Wrap-up
(Shannon) entropy is a measure of uncertainty. Intuitively: if we want to know the value of a variable, there is uncertainty in what that value is before we inspect it (measured in bits).
Conditional entropy

Uncertainty in one variable $X$ in the context of the known measurement of another variable $Y$.

Intuitively: how much uncertainty is there in $X$ after we know the value of $Y$?

e.g. How uncertain are we about the web server is running if we know the IT guy is at lunch?

\[
H(X|Y) = H(X, Y) - H(Y) = - \sum_{x \in X} \sum_{y \in Y} p(x, y) \log p(x|y)
\]
Information is a measure of uncertainty reduction. Intuitively: common information is the amount that knowing the value of one variable tells us about another.
Information-theoretic quantities

Shannon entropy

\[ H(X) = - \sum_x p(x) \log_2 p(x) = \langle - \log_2 p(x) \rangle \]

Conditional entropy

\[ H(X|Y) = - \sum_{x,y} p(x,y) \log_2 p(x|y) \]

Mutual information (MI)

\[ I(X; Y) = H(X) + H(Y) - H(X, Y) = \sum_{x,y} p(x,y) \log_2 \frac{p(x|y)}{p(x)} = \langle \log_2 \frac{p(x|y)}{p(x)} \rangle \]

Conditional MI

\[ I(X; Y|Z) = H(X|Z) + H(Y|Z) - H(X, Y|Z) = \langle \log_2 \frac{p(x|y,z)}{p(x|z)} \rangle \]
Local measures

We can write local (or point-wise) information-theoretic measures for specific observations/configurations \( \{x, y, z\} \):

\[
\begin{align*}
    h(x) &= - \log_2 p(x), \\
    h(x|y) &= - \log_2 p(x|y), \\
    i(x; y) &= \log_2 \frac{p(x|y)}{p(x)}, \\
    i(x; y|z) &= \log_2 \frac{p(x|y, z)}{p(x|z)}
\end{align*}
\]

- We have \( H(X) = \langle h(x) \rangle \) and \( I(X; Y) = \langle i(x; y) \rangle \), etc.
- If \( X, Y, Z \) are time-series, local values measure dynamics over time.
What can we do with these measures?

- Measure the diversity in agent strategies (Miramontes, 1995; Prokopenko et al., 2005).
- Measure long-range correlations as we approach a phase-transition (Ribeiro et al., 2008).
- Feature selection for machine learning (Wang et al., 2014).
- Quantify the information held in a response about a stimulus, and indeed about specific stimuli (DeWeese and Meister, 1999).
- Measure the common information in the behaviour of two agents (Sperati et al., 2008).
- Guide self-organisation (Prokopenko et al., 2006).
- ...

→ Information theory is useful for answering specific questions about information content, shared information, and where and to what extent information about some variable is mirrored.
Information dynamics

We talk about computation as:

- Memory
- Signalling
- Processing

Distributed computation is any process involving these features:

- Time evolution of cellular automata
- Information processing in the brain
- Gene regulatory networks computing cell behaviours
- Flocks computing their collective heading
- Ant colonies computing the most efficient routes to food
- The universe is computing its own future!
Information dynamics

We *talk* about computation as:

- Memory
- Signalling
- Processing

Idea: quantify computation via:

- Information storage
- Information transfer
- Information modification

Distributed computation is any process involving these features:

- Time evolution of cellular automata
- Information processing in the brain
- Gene regulatory networks computing cell behaviours
- Flocks computing their collective heading
- Ant colonies computing the most efficient routes to food
- The universe is computing its own future!

General idea: by quantifying intrinsic computation in the language it is normally described in, we can understand how nature computes and why it is complex.
Key question: how is the next state of a variable in a complex system **computed**?

Q: Where does the information in $x_{n+1}$ come from, and how can we measure it?

Q: How much was stored, how much was transferred, can we partition them or do they overlap?

Complex system as a multivariate time-series of states
Active information storage (Lizier et al., 2012)

How much information about the next observation $X_{n+1}$ of process $X$ can be found in its past state $X_n^{(k)} = \{X_{n-k+1} \ldots X_{n-1}, X_n\}$?

Active information storage:

$$A_X = I(X_{n+1}; X_n^{(k)})$$

$$= \left\langle \log_2 \frac{p(x_{n+1}|x_n^{(k)})}{p(x_{n+1})} \right\rangle$$

Average information from past state that is in use in predicting the next value.
Active information storage (Lizier et al., 2012)

How much information about the next observation $X_{n+1}$ of process $X$ can be found in its past state $X_n^{(k)} = \{X_{n-k+1} \ldots X_{n-1}, X_n\}$?

Active information storage:

$$A_X = I(X_{n+1}; X_n^{(k)}) = \left\langle \log_2 \frac{p(x_{n+1}|x_n^{(k)})}{p(x_{n+1})} \right\rangle$$

Average information from past state that is in use in predicting the next value.
Information transfer

How much information about the state transition $X^{(k)}_n \rightarrow X_{n+1}$ of $X$ can be found in the past state $Y^{(l)}_n$ of a source process $Y$?

**Transfer entropy:** (Schreiber, 2000)

$$T_{Y \rightarrow X} = I(Y^{(l)}_n ; X_{n+1} \mid X^{(k)}_n) = \left< \log_2 \frac{p(x_{n+1} \mid x^{(k)}_n, y^{(l)}_n)}{p(x_{n+1} \mid x^{(k)}_n)} \right>$$

Average info from source that helps predict next value in context of past.

Storage and transfer are **complementary:**

$$H_X = A_X + T_{Y \rightarrow X} + \text{higher order terms}$$
Transfer entropy measures directed coupling between time-series. Intuitively: the amount of information that a source variable tells us about a destination, in the context of the destination’s current state.

e.g. How much does knowing the IT guy is at lunch tell us about the web server running, given its previous state?
Information dynamics in CAs

Domains and blinkers are the dominant information storage entities.

Gliders are the dominant information transfer entities.
Discrete: plug-in estimator

For discrete variables $x$ and $y$, to compute $H(X, Y)$

1. estimate: $p(x, y) = \frac{\text{count}(X=x, Y=y)}{N}$, where $N$ is our sample size;
2. plug-in each estimated PDF to $H(X, Y)$ to get $\hat{H}(X, Y)$
Continuous variables → Differential entropy

Gaussian model: covariances → entropies (Cover and Thomas, 1991)

\[ H(X) = \frac{1}{2} \ln \left( (2\pi e)^d \mid \Omega_X \right) \]
Continuous variables → Differential entropy

1. **Gaussian model**: covariances → entropies (Cover and Thomas, 1991)

   \[
   H(X) = \frac{1}{2} \ln \left( (2\pi e)^d \left| \Omega_X \right| \right)
   \]

2. **(Box-)kernel estimation**: PDFs from counts within a radius \( r \)

   ![Graph showing continuous variables and differential entropy calculation](image)
Continuous variables ➔ Differential entropy

1. **Gaussian model**: covariances ➔ entropies (Cover and Thomas, 1991)

\[ H(X) = \frac{1}{2} \ln \left( (2\pi e)^d \mid \Omega_X \right) \]

2. **(Box-)kernel estimation**: PDFs from counts within a radius \( r \)

3. **Kraskov et al. (2004) (KSG)**: PDFs from counts within a radius determined by \( k \)-nearest neighbours, plus bias-correction. Best of breed
Why JIDT?

JIDT is unique in the combination of features it provides:

- Large array of measures, including all conditional/multivariate forms of the transfer entropy, and complementary measures such as active information storage.
- Wide variety of estimator types and applicability to both discrete and continuous data
Measure-estimator combinations

As of V1.2.1 distribution:

<table>
<thead>
<tr>
<th>Measure</th>
<th>Discrete estimator</th>
<th>Continuous estimators</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Name</td>
<td>Notation</td>
</tr>
<tr>
<td>Entropy</td>
<td>$H(X)$</td>
<td>✓</td>
</tr>
<tr>
<td>Entropy rate</td>
<td>$H_{\mu}X$</td>
<td>✓</td>
</tr>
<tr>
<td>Mutual information (MI)</td>
<td>$I(X; Y)$</td>
<td>✓</td>
</tr>
<tr>
<td>Conditional MI</td>
<td>$I(X; Y</td>
<td>Z)$</td>
</tr>
<tr>
<td>Multi-information</td>
<td>$I(X)$</td>
<td>✓</td>
</tr>
<tr>
<td>Transfer entropy (TE)</td>
<td>$T_{Y \rightarrow X}$</td>
<td>✓</td>
</tr>
<tr>
<td>Conditional TE</td>
<td>$T_{Y \rightarrow X</td>
<td>Z}$</td>
</tr>
<tr>
<td>Active information storage</td>
<td>$A_X$</td>
<td>✓</td>
</tr>
<tr>
<td>Predictive information</td>
<td>$E_X$</td>
<td>✓</td>
</tr>
<tr>
<td>Separable information</td>
<td>$S_X$</td>
<td>✓</td>
</tr>
</tbody>
</table>
Why JIDT?

JIDT is unique in the combination of features it provides:

- Large array of measures, including all conditional/multivariate forms of the transfer entropy, and complementary measures such as active information storage.
- Wide variety of estimator types and applicability to both discrete and continuous data
- Local measurement for all estimators;
- Statistical significance calculations for MI, TE;
- No dependencies on other installations (except Java);
- Lots of demos and information on website/wiki:
  - [https://code.google.com/p/information-dynamics-toolkit/](https://code.google.com/p/information-dynamics-toolkit/)
The Java implementation of JIDT gives us several fundamental features:

- Platform agnostic, requiring only a JVM;
- Object-oriented code, with a hierachical design to interfaces for each measure, allowing dynamic swapping of estimators for the same measure;
- JIDT can be directly called from Matlab/Octave, Python, R, Julia, Clojure, etc, adding efficiency for higher level code;
- Automatic generation of Javadoc.
The distribution

Installation is just a matter of unzipping!

Contents:

- license-gplv3.txt - GNU GPL v3 license;
- infodynamics.jar library file;
- Documentation
- Source code in java/source folder
- Unit tests in java/unittests folder
- build.xml ant build script
- Demonstrations of the code in demos folder.
Documentation

Included in the distribution:

- readme.txt;
- InfoDynamicsToolkit.pdf – a pre-print of the publication introducing JIDT;
- tutorial folder – a full tutorial presentation and sample exercise (also via JIDT wiki)
- javadocs folder – documents the methods and various options for each estimator class;
- PDFs describing each demo in the demos folder;

Also see:

- The wiki pages on the JIDT website
- Our email discussion list jidt-discuss on Google groups.
Architecture for calculators on continuous data

Interfaces

- ConditionalMutualInfoCalculatorMultiVariate
  - infodynamics.measures.continuous

Abstract classes

- ConditionalMutualInfoMultiVariateCommon
  - infodynamics.measures.continuous

Child classes

- ConditionalMutualInfoCalculatorMultiVariateKraskov
  - infodynamics.measures.continuous.kraskov

- ConditionalMutualInfoCalculatorMultiVariateKraskov1
  - infodynamics.measures.continuous.kraskov

Used under CC-BY license from: Lizier, Frontiers in Robotics & AI, 1:11, 2014

Return to JIDT contents
JIDT is distributed with the following demos:

- Auto-analyser GUI (code generator)
- Simple Java Demos
- Recreation of Schreiber’s original transfer entropy examples;
- Information dynamics in Cellular Automata;
- Detecting interaction lags;
- Interregional coupling;
- Behaviour of null/surrogate distributions;

All have documentation provided to help run them.
Auto Analyser GUI (code generator)

GUI apps in the demos/AutoAnalyser folder:
runTEAutoAnalyser.sh and runMIAutoAnalyser.sh – Computing TE and MI could not be easier
GUI apps in the demos/AutoAnalyser folder: runTEAutoAnalyser.sh and runMIAutoAnalyser.sh – Computing TE and MI could not be easier

Just follow the GUI:

1. Select estimator
2. Select data file
3. Identify source/target columns in data
4. Fill out properties (use tool tip for descriptions)
5. Click "Compute"
Auto Analyser GUI (code generator)

Clicking "compute" then gives you:

1. The resulting TE and MI, and

![Code output]

```java
package infodynamics.demos.autoanalysis;

import infodynamics.utils.ArrayFileReader;
import infodynamics.utils.MatrixUtils;
import infodynamics.measures.discrete.*;

public class GeneratedCalculator {

    public static void main(String[] args) throws Exception {

        // 0. Load/prepare the data:
        String dataFile = "/home/joseph/Dropbox/Work/Investigations/JavaCode/sharedProject/Data/2columns200K2.txt";
        ArrayFileReader afr = new ArrayFileReader(dataFile);
        int[][] data = afr.getInt2DMatrix();
        int[] source = MatrixUtils.selectColumn(data, 0);
        int[] dest = MatrixUtils.selectColumn(data, 1);

        // 1. Construct the calculator:
        TransferEntropyCalculatorDiscrete calc = new TransferEntropyCalculatorDiscrete(2, 1);
        // 2. No other properties to set for discrete calculators.
        // 3. Initialise the calculator for (re-)use:
        calc.initialise();
        // 4. Supply the sample data:
        calc.addObservations(source, dest);
        // 5. Compute the estimate:
        double result = calc.computeAverageLocalNumberOfObservations();

        System.out.println("TE_D Discrete(col_0 -> col_1) = 0.0003 bits\n", result);
    }
}
```
Auto Analyser GUI (code generator)

Clicking "compute" then gives you:

1. The resulting TE and MI, and
2. Code to generate this calculation in Java, Python and Matlab
Simple Java Demo 1 – Discrete Data

```java
int arrayLengths = 100;
RandomGenerator rg = new RandomGenerator();
// Generate some random binary data:
int[] sourceArray = rg.generateRandomInts(arrayLengths, 2);
int[] destArray = new int[arrayLengths];
destArray[0] = 0;
System.arraycopy(sourceArray, 0, destArray, 1, arrayLengths - 1);
// Create a TE calculator and run it:
TransferEntropyCalculatorDiscrete teCalc = new TransferEntropyCalculatorDiscrete(2, 1);
teCalc.initialise();
teCalc.addObservations(sourceArray, destArray);
double result = teCalc.computeAverageLocalOfObservations();
```

Data representation:

1. Discrete data as `int[]` arrays with values in range 0...`base` – 1, where e.g. `base` = 2 for binary.
2. Continuous data as `double[]` arrays.
3. For time-series measures, the arrays are indexed by time.
Simple Java Demo 1 – Discrete Data – Usage Paradigm

```
int arrayLengths = 100;
RandomGenerator rg = new RandomGenerator();
// Generate some random binary data:
int[] sourceArray = rg.generateRandomInts(arrayLengths, 2);
int[] destArray = new int[arrayLengths];
destArray[0] = 0;
System.arraycopy(sourceArray, 0, destArray, 1, arrayLengths - 1);
// Create a TE calculator and run it:
TransferEntropyCalculatorDiscrete teCalc = new TransferEntropyCalculatorDiscrete(2, 1);
teCalc.initialise();
teCalc.addObservations(sourceArray, destArray);
double result = teCalc.computeAverageLocalOfObservations();
```

1. **Construct** the calculator, providing parameters
   1. Always check Javadocs for which parameters are required.
   2. Here the parameters are the number of possible discrete symbols per sample (2, binary), and history length for TE ($k = 1$).
   3. Constructor syntax is different for Matlab/Octave/Python.
 Initialise the calculator prior to:

1. use, or
2. re-use (e.g. looping back from line 12 back to line 10 to examine different data).
3. This clears PDFs ready for new samples.
Simple Java Demo 1 – Discrete Data – Usage Paradigm

```java
int arrayLengths = 100;
RandomGenerator rg = new RandomGenerator();
// Generate some random binary data:
int[] sourceArray = rg.generateRandomInts(arrayLengths, 2); int[] destArray = new int[arrayLengths];
destArray[0] = 0;
System.arraycopy(sourceArray, 0, destArray, 1, arrayLengths - 1);
// Create a TE calculator and run it:
TransferEntropyCalculatorDiscrete teCalc = new TransferEntropyCalculatorDiscrete(2, 1);
teCalc.initialise();
tecalc.addObservations(sourceArray, destArray);
double result = teCalc.computeAverageLocalOfObservations();
```

Supply the data to the calculator to construct PDFs:

1. `addObservations()` may be called multiple times;
2. Convert arrays into Java format:
   - From Matlab/Octave using our `octaveToJavaIntArray(array)`, etc., scripts.
   - From Python using `JArray(JInt, numDims)(array)`, etc.
Simple Java Demo 1 – Discrete Data – Usage Paradigm

```java
int arrayLengths = 100;
RandomGenerator rg = new RandomGenerator();
// Generate some random binary data:
int[] sourceArray = rg.generateRandomInts(arrayLengths, 2);
int[] destArray = new int[arrayLengths];
destArray[0] = 0;
System.arraycopy(sourceArray, 0, destArray, 1, arrayLengths - 1);
// Create a TE calculator and run it:
TransferEntropyCalculatorDiscrete teCalc = new TransferEntropyCalculatorDiscrete(2, 1);
teCalc.initialise();
teCalc.addObservations(sourceArray, destArray);
double result = teCalc.computeAverageLocalOfObservations();
```

4 Compute the measure:

- Value is always returned in bits for discrete calculators.
- Result here approaches 1 bit since destination copies the (random) source.
- Other computations include:
  1. `computeLocalOfPreviousObservations()` for local values
  2. `computeSignificance()` to compute $p$-values of measures of predictability (see Appendix A5 of paper for description).
Discrete Data – Usage Paradigm

Construct

initialise()

addObservations()

More data?

yes

no

Compute
Many other demos – e.g. local dynamics in CAs

See PDF documentation for demos/octave/CellularAutomata/ to recreate, e.g. run GsoChapterDemo2013.m.
Sleep apnea heart-breath interaction

Recreates and extends Schreiber’s original transfer entropy calculations 2000 between heart rate & breath rate measurements (data from Rigney et al. (1993) in demos/data/SFI-heartRate_breathVol_bloodOx.txt) in:

1. demos/java/infodynamics/java/schreiberTransferEntropyExamples/-HeartBreathRateKraskovRunner.java
2. demos/octave/SchreiberTransferEntropyExamples/-runHeartBreathRateKraskov.m
3. demos/python/SchreiberTransferEntropyExamples/-runHeartBreathRateKraskov.py
Sleep apnea heart-breath interaction

Box-kernel estimation (k=1)    KSG estimation (k=2)
Summary

We’ve used JIDT in lots of interesting ways:

- Capturing information cascades in swarms;
- Detecting stimulus changes in cat visual cortex;
- Differentiating ASD and control patients using information storage properties;
- Feature selection in machine learning; etc.
Summary

We’ve used JIDT in lots of interesting ways:

- Capturing information cascades in swarms;
- Detecting stimulus changes in cat visual cortex;
- Differentiating ASD and control patients using information storage properties;
- Feature selection in machine learning; etc.

What I want you to take away from this short introduction:

- Understand measures of information dynamics;
- Understand what you could use JIDT for;
- Know how to get JIDT;
- Know how and where to seek support information (wiki, demos, tutorial, Javadocs, mailing list, twitter, ...).
Final messages

We’re seeking PhD students for our Complex Systems group at University of Sydney
Final messages

We’re seeking PhD students for our Complex Systems group at University of Sydney

We’re seeking PhD students for our Complex Systems group at University of Sydney


References I


References II


