UseInPython

Edit New page

Jump to bottom

Joseph Lizier edited this page on 8 Sep 2021 · 23 revisions

How to use the toolkit in Python

Introduction

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

In summary, you will need to install the following dependencies:

- A Java SE / JDK
- Python jpype package, in particular jpype1 if using python3 (see below)
- Python numpy package (see below)

Several longer examples of using the toolkit in Python can be viewed at PythonExamples.

Using Java objects in Python

There are several alternatives for using Java objects in Python, e.g. JPype and Jython.

Here, we focus on using JPype for several reasons, including:

- It is run as an import to ordinary python code in the standard interpreter (jython runs as an alternative python interpreter, and so is not always up to date with the main python interpreter)
- It is packaged for ubuntu (which is my platform of choice) as `python-jpype` As such, here we will only describe use via JPype, though of course you could use an alternative method quite easily.

Using JPype

The first step is to install JPype. Ensure that you don't mix 32-bit python with 64-bit java and vice-versa (as recommended by jpype)!

For Python 2:

- On ubuntu one simply installs `python-jpype` via the ubuntu package manager.
- On other platforms, install as you would other python extensions (e.g. via `pip install jpype`).

For Python 3, you may need to install a slightly different version of JPype:

- On ubuntu linux 18.04 you can install `python3-jpype` via the ubuntu package manager. Otherwise (for earlier versions) it has been reported that `pip3 install jpype1` (You might need to `sudo` that) will install a python 3 compatible JPype (thanks to Michael Wibral). `jpype1-py3` is another available package, however it seems that `jpype1` is more efficient.
- Installation of jpype1 using the Anaconda Python stack also works very well via `conda install -c conda-forge jpype1`, e.g. I have observed this to work from command line with anaconda 3 on windows 10.
- On windows it has been reported that this works inside anaconda: `pip install -i https://pypi.anaconda.org/pypi/simple jpype1` (thanks to Jonathan de Vries).

You can then run your java code fairly simply in python with JPype:

1. Import the relevant packages from jPype, e.g.: `from jpype import *`
2. Start the JVM and tell it where our jar file is, e.g.: `startJVM(getDefaultJVMPath(), "-Djava.class.path=" + jarLocation)`
3. Create a reference to the package and class that you wish to create an instance of, e.g.: `teCalcClass = JPackage("infodynamics.measures.discrete").TransferEntropyCalculatorDiscrete`
4. Create an instance of the class, e.g.: `teCalc = teCalcClass(2,1)`
5. Use the object, e.g.: `teCalc.initialise()`
6. Shutdown the JVM when it's no longer required, e.g.: `shutdownJVM()`

Array conversion from java to python requires some extra code, but is perfectly do-able - see sections 6 and 7 of the JPype user guide. *For example*, to convert a one-dimensional python array `myArray` to a java one dimensional array of doubles, use `JArray(JDouble, 1)(myArray)` -- see e.g. Example 3 in PythonExamples. In reverse, Java arrays returned from JIDT calls to Python will be of JArray type, which (as described in the JPype user guide) "behave like Python lists, except that their size is fixed, and that the contents are of a specific type". You can handle these similarly to Python lists, see e.g. simple Python example 4.

Numpy Alternatively, if you use numpy.array objects these are (*somewhat*) directly accepted by the java methods:

- For doubles (continuous variables), yes you can pass numpy arrays straight through -- see e.g. Example 6 and Example 9 in PythonExamples -- these work in both python 2 and python 3, and indeed I think this is preferable at least in python3 with jpype1. An exception

here is that JPype 0.7 doesn't seem to like to send 2D arrays to overloaded Java methods which have method signatures for either 1D or 2D double arrays -- to handle that, you could convert using explicit conversions to Java arrays as above (*preferred*), e.g. `JArray(JDouble, 2)(myArray.tolist())` for a 2D array, or where I have provided non-overloaded options (`setObservations1D` or `setObservations2D`) then directly use those (may be deprecated in future). You can always run the AutoAnalyser GUI to get the currently recommended approach for your type of data.

- For ints (discrete variables) and it seems booleans, you can only pass numpy arrays straight through in python 2 with jpype, whereas with python 3 with jpype1 you need to convert the numpy array to a list and then a jpype.JArray -- see e.g. the code and comments in Example 1 The AutoAnalyser demo requires the numpy library in order to perform array conversion.

For static methods, you can call them on the reference to the class itself.

Several longer examples of using the toolkit in Python can be viewed at PythonExamples.

Numpy linalg -- Finally, @olivercliff reports that there is a known issue with jpype being used in conjunction with the numpy linalg solver, as detailed on the numpy repo and on the jpype repo. The current fix is to set the number of threads for numpy to 1 using:

```
import os
os.environ['OMP_NUM_THREADS'] = '1'
```

    - 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
```