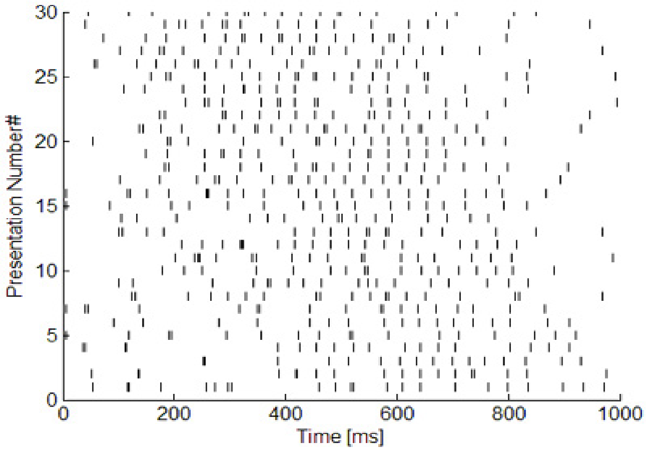
Interoperability among Data Processing Frameworks

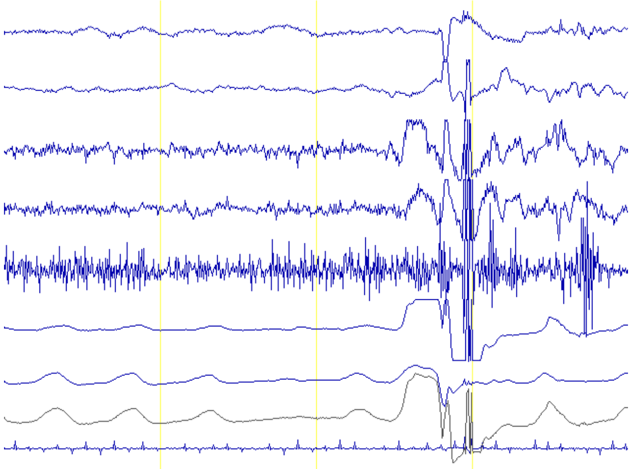
Reality or Wishful Thinking?

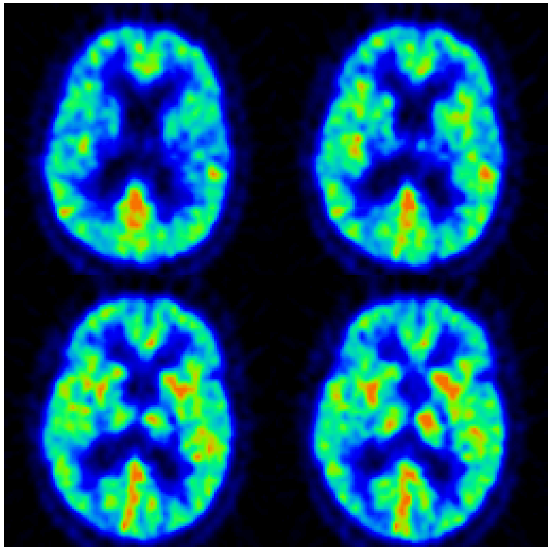
Tiziano Zito
Niko Wilbert, Rike-Benjamin Schuppner, Zbigniew Jędrzejewski-Szmek,
Laurenz Wiskott, Pietro Berkes

Python in Neuroscience Satellite to EuroSciPy
Ecole Normale Supérieure, Paris
August 30th, 2011

Data processing in neuroscience







Data processing libraries in Python

scikit-learn

PyBrain

MLPy

PyML

PyMVPA

Shogun

Milk

NLTK

Orange

Elefant

LibSVM

OpenCV

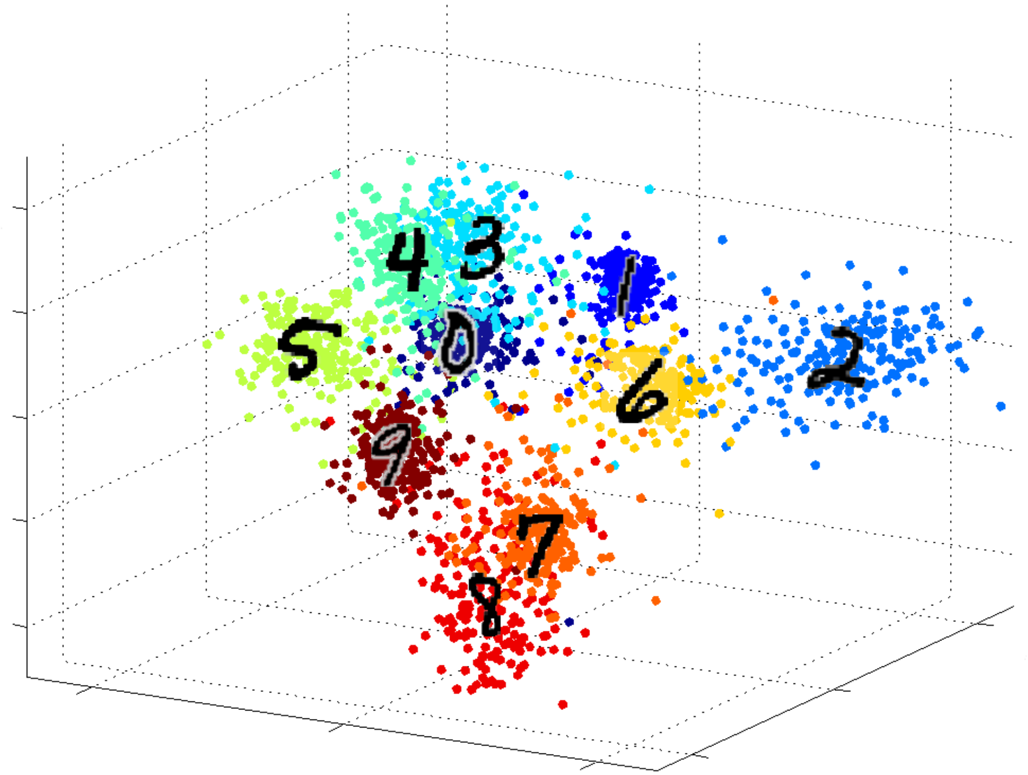
NiPYPE

Pylearn

MDP

… how many more?

Diversity: a source of joy for the user



Strategies I - wrap and be wrapped

- define and advertise a clear API
  (by inheritance / by convention?)
  (no, epydoc is not enough)

- numpy.ndarray for the I/O
  (enhanced array subclasses or proxy
  objects with a "asarray" method?)

- release as often as you want but you
  shall not break backward compatibility
  (you have more users than you think)

- your code shall be *inspectable*
  (duck-typing is not always an option)

Strategies II – softly depend

- hard dependencies are expensive

- no dependencies to specific versions

- no locally modified copies

- ask neuro.debian.net to package you

- soft dependencies are worth the effort

- hard-coded or dynamically generated wrappers?

MDP approach to interoperability

Building blocks: Node

- fundamental data processing element

  Node classes correspond to algorithms
- API: train

  support for multiple phases, batch,
  online, chunks, supervised, unsupervised
- API: <u>execute</u>

  map *n*-dim input to *m*-dim output
- API: <u>inverse</u>

  inverse of execute mapping
- data format: 2-dim numpy arrays
- automatic consistency checks and
  conversions (dimensions, dtype, ...)

MDP approach to interoperability

Building blocks: Flow

- combine nodes in a pipeline
- API: train, execute, inverse
- automatic training, execution, inversion
- automatic checks: dims and data formats
- Flow is a Python container (list) (syntactic sugar)
- feed on arrays or iterators
- crash recovery, checkpoints

MDP approach to interoperability

Building blocks: Network

- **Layer**

  combine nodes horizontally in parallel

- **Switchboard**

  routing between layers

- **FlowNode**

  encapsulate a Flow into a “super” Node

- **everything is a Node**

  combine as you want all acyclic graphs
  can be implemented

Embed and wrap MDP

- I/O by 2-dim numpy.ndarray

- API is stable (2004-?) and designed for straightforward embedding

- PyMVPA (sprint!)
  PyMCA
  Oger (sprint!)
  Chandler

MDP wraps and embeds

- scipy
- libsvm
- shogun
- parallel-python
- joblib
- scikit-learn...

MDP && scikit-learn

- wrappers dynamically generated (docs too!)

Pros:

- transparent
- forward compatible

Cons:

- API conventions are not always consistent
- force us to duck-typing
- API is not carved in stone
- manual intervention to get __all__
- # of output components

A future of interoperability

- diversity is good! no winner-take-all

- 3 simple communication rules:
  write! talk! link!

- heterogeneous sprints: induce interbreeding

- but...

A future of interoperability

- diversity is good! no winner-take-all

- 3 simple communication rules:
  write! talk! link!

- heterogeneous sprints: induce interbreeding

- but... stop talking, start coding!