mist

Andrew Banman

Mar 30, 2021

CONTENTS

CHAPTER

ONE

MIST

Mist is a Multivariable Information Theory-based relationship Search Tool. The Mist library API computes the shared information content of variables that may indicate a functional dependency. The type of IT measurement Mist uses is configurable, as are many search parameters.

Version 0.1.0

Mist for Python is available on PyPi: <https://pypi.org/project/libmist/>

Mist documentation is available on ReadTheDocs: <https://libmist.readthedocs.io>

CHAPTER

TWO

QUICK START

The easiest way to run Mist is through the Python module. The following minimal example sets up a Mist object for a simple Symmetric Delta search (the default IT measurement).

```
import libmist
mist = libmist.Mist()
mist.load_file('/path/to/data.csv')
mist.set_outfile('/dev/stdout')
mist.compute()
```

There are numerous functions to configure Mist – below are some of the most important. For a full list read the API documentation for mist::Mist.

```
mist.load_ndarray()      # load data from a Python.Numpy.ndarray (see docs for␣
↪restrictions)
mist.set_tuple_size()    # set the number of variables in each tuple
mist.set_measure()       # set the Information Theory Measure
mist.set_threads()       # set the number of computration threads
```

This Python syntax is virtually identical to the C++ code you would write for a program using the Mist library, as you can see in the examples directory.

CHAPTER

THREE

BUILDING

3.1 mist

These packages are required to build the mist library:

- CMake (minimum version 3.5) <https://cmake.org/download/>
- Boost (minimum version 1.58.0) <https://www.boost.org/users/download/>.

Run *cmake* in out-of-tree build directory:

```
mkdir /path/to/build
cd /path/to/build
cmake /path/to/mist
make
```

3.2 mist python library

Additional requirements:

- Python development packages (python3-dev or python-dev).
- Boost Python and Numpy components. For Boost newer than 1.63 use the integrated Boost.Numpy (libboost-numpy) package. For earlier versions install ndarray/Boost.Numpy <https://github.com/ndarray/Boost.NumPy>.

Run *cmake* with *BuildPython* set to *ON*:

```
mkdir /path/to/build
cd /path/to/build
cmake -DBuildPython:BOOL=ON /path/to/mist
make
```

Note: both the mist and ndarray/Boost.numpy builds use the default python version installed on the system. To use a different python version, change the FindPythonInterp, FindPythonLibs, and FindNumpy invocations in both packages to use the same python version.

3.3 Documentation (optional)

Additional Requirements

- Doxygen <http://www.doxygen.nl/download.html>
- Sphinx <https://www.sphinx-doc.org/en/master/usage/installation.html>
- Breathe <https://pypi.org/project/breathe/>
- sphinx_rtd_theme <https://github.com/rtfd/sphinx_rtd_theme>

Run *cmake* with *BuildDocs* set to *ON*:

```
mkdir /path/to/build
cd /path/to/build
cmake -DBuildDocs:BOOL=ON /path/to/mist
make Sphinx
```

And then run the build as above.

CHAPTER

FOUR

FOR DEVELOPERS

This project follows the Pitchfork Layout (PFL). Namespaces are encapsulated in separate directories. Any physical unit must only include headers within its namespace, the root namespace (core), or interface headers in other namespaces. The build system discourages violations by making it difficult to link objects across namespaces.

Documentation for this project is dynamically generated with Doxygen and Sphinx. Comments in the source following Javadoc style are included in the docs. Non-documented comments, e.g. implementation notes, developer advice, etc. follow standard C++ comment style. This README and other documents should be written in the intersection of Markdown and reStructuredText <https://gist.github.com/dupuy/1855764> for best interoperability.

CHAPTER

FIVE

CREDITS

Mist is written by Andrew Banman. It is based on software written by Nikita Sakhanenko. The ideas behind entropy-based functional dependency come from Information Theory research by David Galas, Nikita Sakhanenko, and James Kunert.

CHAPTER

SIX

API

mist is comprised of logically distinct components encapsulated by namespaces. Classes access other namespaces via an interface class. Users typically only need to be concerned with classes in the root namespace, whereas developers will need the rest.

6.1 mist

The root namespace includes composition classes and classes common to the sub-namespaces.

Warning: doxygenclass: Cannot find class “mist::Mist” in doxygen xml output for project “mist” from directory: build/xml

Warning: doxygenclass: Cannot find class “mist::Variable” in doxygen xml output for project “mist” from directory: build/xml

6.2 mist::algorithm

Algorithms to divide and conquer Information Theory computations.

Warning: doxygennamespace: Cannot find namespace “mist::algorithm” in doxygen xml output for project “mist” from directory: build/xml

6.3 mist::cache

Cache intermediate results for performance improvement.

Warning: doxygennamespace: Cannot find namespace “mist::cache” in doxygen xml output for project “mist” from directory: build/xml

6.4 mist::io

Input/Output

Warning: doxygennamespace: Cannot find namespace “mist::io” in doxygen xml output for project “mist” from directory: build/xml

6.5 mist::it

Information Theory definitions and algorithms.

Warning: doxygennamespace: Cannot find namespace “mist::it” in doxygen xml output for project “mist” from directory: build/xml