h5features Documentation

Release 1.1.0

Thomas Schatz, Mathieu Bernard, Roland Thiolliere

October 03, 2016

Note: The source code is available at http://www.github.com/bootphon/h5features.

Table of contents

Package overview

Note: In the following code samples, the h5features package is imported as:

```
import h5features as h5f
```

1.1 Brief

The h5features package allows you to easily **interface your code with a HDF5 file**. It is designed to efficiently **read and write large features datasets**. It is a wrapper on h5py and and is used for exemple in the ABXpy package.

- Package organization:

  The **main classes** composing the package are `h5f.Writer` and `h5f.Reader`, which respectively write to and read from HDF5 files, and `h5f.Data` which interface that data with your code.

- Data structure:

  The **h5features data** is structured as a follows

  - a list of **items** represented by their names (files names for exemple),
  - for each item, some attached **features** as a numpy array,
  - some **labels** information attached to features, also as numpy arrays.

- File structure:

  In a h5features file, **data is stored as a HDF5 group**. The underlying group structure directly follows data organization. A h5features *group* mainly stores a *version* attribute and the following datasets: *items*, *labels*, *features* and *index*.

1.2 Description

The h5features package provides efficient and flexible I/O on a (potentially large) collection of (potentially small) 2D datasets with one fixed dimension (the 'feature' dimension, identical for all datasets) and one variable dimension (the 'label' dimension, possibly different for each dataset). For example, the collection of datasets can correspond to speech features (e.g. MFC coefficients) extracted from a collection of speech recordings with variable durations. In this case, the 'label' dimension corresponds to time and the meaning of the 'feature' dimension depends on the type of speech features used.

The h5features package can handle small or large collections of small or large datasets, but the case that motivated its design is that of large collections of small datasets. This is a common case in speech signal processing, for example, where features are often extracted separately for each sentence in multi-hours recordings of speech signal. If the features are stored in individual files, the number of files becomes problematic. If the features are stored in a single big file which does not support partial I/O, the size of the file becomes problematic. To solve this problem, h5features is built on top of h5py, a python binding of the HDF5 library, which supports partial I/O. All the items in the collection of datasets are stored in a single file and an indexing structure allows for efficient I/O on single items or on contiguous groups of items. h5features also indexes the 'label' dimension of each individual dataset and allow partial I/O along it. To continue our speech features example, this means that it is possible to load just the features for a specific time-interval in a specific utterance (corresponding to a word or phone of interest for instance). The labels indexing the 'label' dimension typically correspond to center-times or time-intervals associated to each feature vector in a dataset.

1.3 Command line converter

The scipt `convert2h5features` allows you to simply convert a set of files to a single h5features file. Supported files format are numpy NPZ and Octave/Matlab mat files.

For more info on that script, have a:

```
$ convert2h5features --help
```

1.4 Basic usage

```
import h5features as h5f

########################
# Prelude to the exemple
########################

def generate_data(nitem, nfeat=2, dim=10, labeldim=1, base='item'):
    """Returns a randomly generated h5f.Data instance.

    - nitem is the number of items to generate.
    - nfeat is the number of features to generate for each item.
    - dim is the dimension of the features vectors.
    - base is the items basename
    - labeldim is the dimension of the labels vectors.
    """
    import numpy as np

    # A list of item names
    items = [base + '_' + str(i) for i in range(nitem)]

    # A list of features arrays
    features = [np.random.randn(nfeat, dim) for _ in range(nitem)]

    # A list on 1D or 2D times arrays
    if labeldim == 1:
        labels = [np.linspace(0, 1, nfeat)] * nitem
    else:
        t = np.linspace(0, 1, nfeat)
        labels = [np.array([t+i for i in range(labeldim)])] * nitem
```

```
    # Format data as required by the writer
    return h5f.Data(items, labels, features, check=True)

########################
# Writing data to a file
########################

# Generate some data for 100 items
data = generate_data(100)

# Initialize a writer, write the data in a group called 'group1' and
# close the file
writer = h5f.Writer('exemple.h5')
writer.write(data, 'group1')
writer.close()

# More pythonic, the with statement
with h5f.Writer('exemple.h5') as writer:
    # Write the same data to a second group
    writer.write(data, 'group2')

    # You can append new data to an existing group if all items have
    # different names. Here we generate 10 more items and append them
    # to the group 2, which now stores 110 items.
    data2 = generate_data(10,  base='item2')
    writer.write(data2, 'group2', append=True)

    # If append is not True, existing data in the group is overwrited.
    data3 = generate_data(10, base='item3')
    writer.write(data3, 'group2', append=True)  # 120 items
    writer.write(data3, 'group2')               # 10 items


##########################
# Reading data from a file
##########################

# Initialize a reader and load the entire group. A notable difference
# with the Writer is that a Reader is attached to a specific group of
# a file. This allows optimized read operations.
rdata = h5f.Reader('exemple.h5', 'group1').read()

# Hopefully we read the same data we just wrote
assert rdata == data

# Some more advance reading facilities
with h5f.Reader('exemple.h5', 'group1') as reader:
    # Same as before, read the whole data
    whole_data = reader.read()

    # Read the first item stored on the group.
    first_item = reader.items.data[0]
    rdata = reader.read(first_item)
    assert len(rdata.items()) == 1

    # Read an interval composed of the 10 first items.
    tenth_item = reader.items.data[9]
    rdata = reader.read(first_item, tenth_item)
```

```
    assert len(rdata.items()) == 10


####################
# Playing with labels
####################

# Previous exemples shown writing and reading labels associated to 1D
# times information (each feature vector correspond to a single
# timestamp, e.g. the center of a time window). In more advanced
# processing you may want to store 2D times information (e.g. begin
# and end of a time window). For now non-numerical labels or not
# supported.

data = generate_data(100, labeldim=2)
h5f.Writer('exemple.h5').write(data, 'group3')

rdata = h5f.Reader('exemple.h5', 'group3').read()
assert rdata == data

# Remove the writed file
from os import remove
remove('exemple.h5')
```

CHAPTER 2

Installation

2.1 Getting the source

The source code is publicly available at https://github.com/bootphon/h5features

```
$ git clone https://github.com/bootphon/h5features.git
```

Note: In what follows we suppose your current directory is the root of the h5features package you just cloned:

```
$ cd h5features
```

2.2 Installing

To install the package, run:

```
$ python setup.py build
$ [sudo] python setup.py install
```

h5features relies on external dependencies. The setup script should install it automatically, but you may want to install it manually. The required packages are:

- h5py 2.3.0 or newer
- NumPy 1.8.0 or newer
- scipy 0.13.0 or newer
- pytest
- sphinx

2.3 Testing

This package is continuously integrated with travis. You can follow the build status here. For testing it on your local machine, simply run from the root directory:

```
$ py.test
```

2.4 Building the documentation

The documentation (the one you are currently reading) is builded with *sphinx*. The main HTML page is generated to *docs/build/html/index.html*:

```
$ python setup.py build_sphinx
```

Or:

```
$ cd docs && make html
```

CHAPTER 3

API Reference

3.1 Top-level modules

3.1.1 h5features.data module

3.1.2 h5features.reader module

3.1.3 h5features.writer module

3.1.4 h5features.converter module

3.1.5 h5features.h5features module

3.2 Low-level modules

3.2.1 h5features.entry module

3.2.2 h5features.features module

3.2.3 h5features.index module

3.2.4 h5features.items module

3.2.5 h5features.labels module

3.2.6 h5features.version module

What's new ?

4.1 What's new in h5features 1.1

The main goal of the 1.1 release is to provide a better, safer and clearer code than previous release whithout changing the front-end API.

- **Object oriented refactoring**

  An object oriented architecture have been coded. The main entry classes are Data, Reader and Writer.

- **Distinct overwrite/append mode**

  Appending to an existing file is now optional. This allow minor optimzations but that make sense when data is big.

- **Change in the HDF5 file structure**

  With *group* as the h5features root in a HDF5 file, the structure evolved from *group/[files, times, features, file_index]* to *group/[items, labels, features, index]*. These changes are done for clarity and consistency with code and usage.

- **Change in times/labels**

  You can now write 2D labels to h5features.

- **Test suite**

  The project is now endowed with a pytest suite of more than 50 unit tests.

- **Improved documentation**

  This is what you are reading now!

4.2 What's new in h5features 1.0

Over the previous development release (0.1), the 1.0 release changes the underlying HDF5 file structure, add a *version* attribute and improve the index facilities.

4.3 TODO list

These document the scheduled and/or requested changes to the h5features package.

4.3.1 For 1.1 release

- Test convertion from h5features old versions
- read/write bigger than RAM -> catch MemoryError when np.concatenate on writing.
- Data.__repr__

4.3.2 For a future release

- labels can be of arbitrary type
- Have a h5features.File class inspired by h5py.File
- Implement sparse functionalities
- Handle h5py MPI driver for concurent reading
- Enable autochunking from h5py (with chunk=None)

CHAPTER 5

License and copyright

This package is developed whithin the Bootphon project.

Copyright 2014, 2015 Thomas Schatz, Mathieu Bernard, Roland Thiolliere.

h5features is free software: you can redistribute it and/or modify it under the terms of the **GNU General Public License** as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

h5features is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with h5features. If not, see http://www.gnu.org/licenses/.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search