h5features Documentation

Release 1.2.2

Thomas Schatz, Mathieu Bernard, Roland Thiolliere

Dec 12, 2018

Contents

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

Table of contents

CHAPTER 1

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)]
```

(continues on next page)

(continued from previous page)

```
    # 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
```

(continues on next page)

(continued from previous page)

```
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

2.2.1 Dependancies

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

On Debian/Ubuntu:

```
sudo apt-get install python3-numpy python3-scipy python3-h5py
```

Using Python anaconda:

```
conda install numpy scipy h5py
```

2.2.2 Setup

To install the package, run:

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

2.3 Testing

This package is continuously integrated with travis. You can follow the build status here.

For testing it on your local machine, make sure you have *pytest* installed:

```
pip install pytest
```

Then simply run from the root directory:

```
pytest -v ./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*:

```
pip install Sphinx mock sphinx_rtd_theme

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

Provides the Data class to the h5features package.

class h5features.data.**Data**(*items*, *labels*, *features*, *sparsity=None*, *check=True*)
Bases: object

This class manages h5features data.

append(*data*)
Append a Data instance to self

clear()
Erase stored data

dict_features()
Returns a items/features dictionary.

dict_labels()
Returns a items/labels dictionary.

features()
Returns the stored features as a list of numpy arrays.

init_group(*group*, *chunk_size*)
Initializes a HDF5 group compliant with the stored data.

This method creates the datasets 'items', 'labels', 'features' and 'index' and leaves them empty.

Parameters

- **group** (*h5py.Group*) – The group to initializes.
- **chunk_size** (*float*) – The size of a chunk in the file (in MB).

is_appendable_to(*group*)
Returns True if the data can be appended in a given group.

is_empty()

items()
Returns the stored items as a list of str.

labels()
Returns the stored labels as a list.

write_to(*group*, *append=False*)
Write the data to the given group.

Parameters

- **group** (*h5py.Group*) – The group to write the data on. It is assumed that the group is already existing or initialized to store h5features data (i.e. the method `Data.init_group` have been called.
- **append** (*bool*) – If False, any existing data in the group is overwrited. If True, the data is appended to the end of the group and we assume `Data.is_appendable_to` is True for this group.

3.1.2 h5features.reader module

Provides the Reader class to the h5features package.

class h5features.reader.**Reader**(*filename*, *groupname=None*)
Bases: object

This class provides an interface for reading from h5features files.

A *Reader* object wrap a h5features file. When created it loads items and index from file. The read() method then allows fast access to features and times data.

Parameters

- **filename** (*str*) – Path to the HDF5 file to read from.
- **groupname** (*str*) – Name of the group to read from in the file. If None, guess there is one and only one group in *filename*.

Raises **IOError** – if *filename* is not an existing HDF5 file or if *groupname* is not a valid group in *filename*.

close()

index_read(*index*)
Read data from its indexed coordinate

read(*from_item=None*, *to_item=None*, *from_time=None*, *to_time=None*)
Retrieve requested data coordinates from the h5features index.

Parameters

- **from_item** (*str*) – Optional. Read the data starting from this item. (defaults to the first stored item)
- **to_item** (*str*) – Optional. Read the data until reaching the item. (defaults to from_item if it was specified and to the last stored item otherwise).
- **from_time** (*float*) – Optional. (defaults to the beginning time in from_item) The specified times are included in the output.

- **to_time** (*float*) – Optional. (defaults to the ending time in to_item) the specified times are included in the output.

Returns An instance of h5features.Data read from the file.

3.1.3 h5features.writer module

Provides the Writer class to the h5features module.

class h5features.writer.**Writer**(*filename*, *chunk_size=0.1*, *version='1.1'*, *mode='a'*)
Bases: object

This class provides an interface for writing to h5features files.

Parameters

- **filename** (*str*) – The name of the HDF5 file to write on. For clarity you should use a '.h5' or '.h5f' extension but this is not required by the package.
- **chunk_size** (*float*) – Optional. The size in Mo of a chunk in the file. Default is 0.1 Mo. A chunk size below 8 Ko is not allowed as it results in poor performances.
- **version** (*str*) – Optional. The file format version to write, default is to write the latest version.
- **mode** (*char*) – Optional. The mode for overwriting an existing file, 'a' to append data to the file, 'w' to overwrite it

Raises **IOError** – if the file exists but is not HDF5, if the file can be opened, if the mode is not 'a' or 'w', if the chunk size is below 8 Ko or if the requested version is not supported.

close()
Close the HDF5 file.

write(*data*, *groupname='h5features'*, *append=False*)
Write h5features data in a specified group of the file.

Parameters

- **data** (*dict*) – A *h5features.Data* instance to be writed on disk.
- **groupname** (*str*) – Optional. The name of the group in which to write the data.
- **append** (*bool*) – Optional. This parameter has no effect if the *groupname* is not an existing group in the file. If set to True, try to append new data in the group. If False (default) erase all data in the group before writing.

Raises **IOError** – if append requested but not possible.

3.1.4 h5features.converter module

Provides the Converter class to the h5features package.

class h5features.converter.**Converter**(*filename*, *groupname='h5features'*, *chunk=0.1*)
Bases: object

This class allows convertion from various formats to h5features.

- A *Converter* instance owns an h5features file and write converted input files to it, in a specified group.
- An input file is converted to h5features using the *convert* method, which choose a concrete conversion method based on the input file extension.

- Supported extensions are:
  - **.npz** for numpy NPZ files
  - **.mat** for Octave/Matlab files
  - **.h5** for h5features files. In this later case, the files are simply converted to latest version of the h5features data format

  Parameters

  - **filename** (*str*) – The h5features to write in.
  - **groupname** (*str*) – The group to write in *filename*
  - **chunk** (*float*) – Size a chunk in *filename*, in MBytes.

close()
Close the converter and release the owned h5features file.

convert(*infile*, *item=None*)
Convert an input file to h5features based on its extension.

Raises

- **IOError** – if *infile* is not a valid file.
- **IOError** – if *infile* extension is not supported.

h5features_convert(*infile*)
Convert a h5features file to the latest h5features version.

mat_convert(*infile*, *item*)
Convert a Octave/Matlab file to h5features.

npz_convert(*infile*, *item*)
Convert a numpy NPZ file to h5features.

3.1.5 h5features.h5features module

Provides the read() and write() wrapper functions.

Note: For compatibility with h5features 1.0, this legacy top-level API have been conserved in this module. Except for use in legacy code, it is **better not to use it**. Use instead the *h5features.writer* and *h5features.reader* modules.

---

h5features.h5features.**read**(*filename*, *groupname=None*, *from_item=None*, *to_item=None*, *from_time=None*, *to_time=None*, *index=None*)
Reads in a h5features file.

Parameters

- **filename** (*str*) – Path to a hdf5 file potentially serving as a container for many small files
- **groupname** (*str*) – HDF5 group to read the data from. If None, guess there is one and only one group in *filename*.
- **from_item** (*str*) – Optional. Read the data starting from this item. (defaults to the first stored item)

- **to_item** (*str*) – Optional. Read the data until reaching the item. (defaults to from_item if it was specified and to the last stored item otherwise)
- **from_time** (*float*) – Optional. (defaults to the beginning time in from_item) the specified times are included in the output
- **to_time** (*float*) – Optional. (defaults to the ending time in to_item) the specified times are included in the output
- **index** (*int*) – Optional. For faster access. TODO Document and test this.

Returns

A tuple (times, features) such as:

- time is a dictionary of 1D arrays values (keys are items).
- features: A dictionary of 2D arrays values (keys are items) with the 'feature' dimension along the columns and the 'time' dimension along the lines.

Note: Note that all the files that are present on disk between to_item and from_item will be loaded and returned. It's the responsibility of the user to make sure that it will fit into RAM memory.

---

h5features.h5features.**simple_write**(*filename*, *group*, *times*, *features*, *item='item'*, *mode='a'*)

Simplified version of *write()* when there is only one item.

h5features.h5features.**write**(*filename*, *groupname*, *items*, *times*, *features*, *dformat='dense'*, *chunk_size=0.1*, *sparsity=0.1*, *mode='a'*)

Write h5features data in a HDF5 file.

This function is a wrapper to the Writer class. It has three purposes:

- Check parameters for errors (see details below),
- Create Items, Times and Features objects
- Send them to the Writer.

Parameters

- **filename** (*str*) – HDF5 file to be written, potentially serving as a container for many small files. If the file does not exist, it is created. If the file is already a valid HDF5 file, try to append the data in it.
- **groupname** (*str*) – Name of the group to write the data in, or to append the data to if the group already exists in the file.
- **items** (*list of str*) – List of files from which the features where extracted. Items must not contain duplicates.
- **times** (*list of 1D or 2D numpy arrays*) – Time value for the features array. Elements of a 1D array are considered as the center of the time window associated with the features. A 2D array must have 2 columns corresponding to the begin and end timestamps of the features time window.
- **features** (*list of 2D numpy arrays*) – Features should have time along the lines and features along the columns (accomodating row-major storage in hdf5 files).
- **dformat** (*str*) – Optional. Which format to store the features into (sparse or dense). Default is dense.
- **chunk_size** (*float*) – Optional. In Mo, tuning parameter corresponding to the size of a chunk in the h5file. Ignored if the file already exists.

- **sparsity** (*float*) – Optional. Tuning parameter corresponding to the expected proportion (in [0, 1]) of non-zeros elements on average in a single frame.
- **mode** (*char*) – Optional. The mode for overwriting an existing file, 'a' to append data to the file, 'w' to overwrite it

Raises

- **IOError** – if the filename is not valid or parameters are inconsistent.
- **NotImplementedError** – if dformat == 'sparse'

3.2 Low-level modules

3.2.1 h5features.entry module

Provides the Entry class to the h5features package.

class h5features.entry.**Entry**(*name*, *data*, *dim*, *dtype*, *check=True*)
Bases: object

The Entry class is the base class of h5features.Data entries.

It provides a shared interface to the classes Items, Times and Features which all together compose a Data.

append(*entry*)
Append an entry to self

clear()
Erase stored data

is_appendable(*entry*)
Return True if entry can be appended to self

h5features.entry.**nb_per_chunk**(*item_size*, *item_dim*, *chunk_size*)
Return the number of items that can be stored in one chunk.

Parameters

- **item_size** (*int*) – Size of an item's scalar componant in Bytes (e.g. for np.float64 this is 8)
- **item_dim** (*int*) – Items dimension (length of the second axis)
- **chunk_size** (*float*) – The size of a chunk given in MBytes.

3.2.2 h5features.features module

Provides Features class to the h5features module.

class h5features.features.**Features**(*data*, *check=True*, *sparsetodense=False*)
Bases: *h5features.entry.Entry*

This class manages features in h5features files

Parameters

- **data** (*list of 2D numpy arrays*) – Features must have time along the lines and features along the columns (accomodating row-major storage in hdf5 files).

- **sparsetodense** (*bool*) – If True convert sparse matrices to dense when writing. Used for compatibility with 1.0.

Raises **IOError** – if features are badly formatted.

create_dataset(*group*, *chunk_size*)
Initialize the features subgoup

is_appendable_to(*group*)
Return True if features are appendable to a HDF5 group

is_sparse()
Return True if features are sparse matrices

write_to(*group*, *append=False*)
Write stored features to a given group

class h5features.features.**SparseFeatures**(*data*, *sparsity*, *check=True*)
Bases: *h5features.features.Features*

This class is specialized for managing sparse matrices as features

create_dataset(*group*, *chunk_size*)
Initializes sparse specific datasets

write_to(*group*, *append=False*)
Write stored features to a given group

h5features.features.**contains_empty**(*features*)
Check features data are not empty

Parameters **features** (*list of numpy arrays.*) – The features data to check.

Returns True if one of the array is empty, False else.

h5features.features.**parse_dformat**(*dformat*, *check=True*)
Return *dformat* or raise if it is not 'dense' or 'sparse'

h5features.features.**parse_dim**(*features*, *check=True*)
Return the features dimension, raise if error

Raise IOError if features have not all the same positive dimension. Return dim (int), the features dimension.

h5features.features.**parse_dtype**(*features*, *check=True*)
Return the features scalar type, raise if error

Raise IOError if all features have not the same data type. Return dtype, the features scalar type.

3.2.3 h5features.index module

Provides indexing facilities to the h5features package.

This index typically allows a faster read access in large datasets and is transparent to the user.

Because the h5features package is designed to handle large datasets, features and times data is internally stored in a compact *indexed* representation.

h5features.index.**create_index**(*group*, *chunk_size*)
Create an empty index dataset in the given group.

h5features.index.**cumindex**(*features*)
Return the index computed from features.

h5features.index.**read_index**(*group*, *version='1.1'*)
Return the index stored in a h5features group.

Parameters

- **group** (*h5py.Group*) – The group to read the index from.
- **version** (*str*) – The h5features version of the *group*.

Returns a 1D numpy array of features indices.

h5features.index.**write_index**(*data*, *group*, *append*)
Write the data index to the given group.

Parameters

- **data** (*h5features.Data*) – The that is being indexed.
- **group** (*h5py.Group*) – The group where to write the index.
- **append** (*bool*) – If True, append the created index to the existing one in the *group*. Delete any existing data in index if False.

3.2.4 h5features.items module

Provides the Items class to the h5features package.

class h5features.items.**Items**(*data*, *check=True*)
Bases: *h5features.entry.Entry*

This class manages items in h5features files.

Parameters **data** (*list of str*) – A list of item names (e.g. files from which the features where extracted). Each name of the list must be unique.

Raises **IOError** – if data is empty or if one or more names are not unique in the list.

create_dataset(*group*, *chunk_size*)

is_appendable_to(*group*)

is_valid_interval(*lower*, *upper*)
Return False if [lower:upper] is not a valid subitems interval. If it is, then returns a tuple of (lower index, upper index)

write_to(*group*)
Write stored items to the given HDF5 group.

We assume that self.create() has been called.

h5features.items.**read_items**(*group*, *version='1.1'*, *check=False*)
Return an Items instance initialized from a h5features group.

3.2.5 h5features.labels module

Provides the Labels class to the h5features module.

class h5features.labels.**Labels**(*labels*, *check=True*)
Bases: *h5features.entry.Entry*

This class manages labels related operations for h5features files

Parameters

- **labels** (*list of numpy arrays*) – Each element of the list contains the labels of an h5features item. Empty list are not accepted. For all t in labels, we must have t.ndim to be either 1 or 2.
  - 1D arrays contain the center labelstamps of each frame of the related item.
  - 2D arrays contain the begin and end labelstamps of each items's frame, thus having t.ndim == 2 and t.shape[1] == 2.
- **check** (*bool*) – If True, raise on errors

Raises **IOError** – if the time format is not 1 or 2, or if labels arrays have different dimensions.

Returns The parsed labels dimension is either 1 or 2 for 1D or 2D labels arrays respectively.

static check(*labels*)
Raise IOError if labels are not correct

***labels* must be a list of sorted numpy arrays of equal** dimensions (must be 1D or 2D). In the case of 2D labels, the second axis must have the same shape for all labels.

create_dataset(*group*, *per_chunk*)

is_appendable_to(*group*)

static parse_dim(*labels*)
Return the labels vectors dimension

write_to(*group*)

3.2.6 h5features.version module

Provides versioning facilities to the h5features package.

This module manages the h5features **file format versions**, specified as strings in the format 'major.minor'. File format versions are independant of the h5feature package version (but actually follow the same numerotation scheme).

The module provides functions to list supported versions, read a version from a h5features file or check a specific version is supported.

h5features.version.**is_same_version**(*version*, *group*)
Return True if *version* and *read_version(group)* are equals.

h5features.version.**is_supported_version**(*version*)
Return True if the *version* is supported by h5features.

h5features.version.**read_version**(*group*)
Return the h5features version of a given HDF5 *group*.

Look for a 'version' attribute in the *group* and return its value. Return '0.1' if the version is not found. Raises an IOError if it is not supported.

h5features.version.**supported_versions**()
Return the list of file format versions supported by h5features.

CHAPTER 4

What's new ?

4.1 h5features-1.2.2

- bugfix: broken test on python-3.6.3.
- bugfix: missing files in MANIFEST.in for installation with *pip install h5features*.

4.2 h5features-1.2.1

- The script `convert2h5features` is now installed by the setup script.

4.3 h5features-1.2

- **Breaking change** Labels associated with features data must be sorted in increasing order. This is convenient to use with timestamps and improve reading huge datasets with long labels.
- **Breaking change** Appending new data to an exisiting item is no more allowed.

  Suppose a h5f file with 3 items `['a', 'b', 'c']`, in 1.1 it was possible to append 3 items `['c', 'd', 'e']`, giving a file with the 5 items `['a', 'b', 'c', 'd', 'e']`, where the item `'c'` being the concatenation of original and appended data. That facility was messy and is removed in 1.2.
- Bugfix when writing unidimensional features
- Bugfix when reading from time/to time in Reader
- Safely overwrite existing groups in h5features files with mode='w'
- Now more than 100 test cases

4.4 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!

4.5 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.

4.6 TODO list

4.6.1 For a future release

- Converter: Possibility to specify other names than 'labels', 'features' for the input files
- Test convertion from h5features old versions
- read/write bigger than RAM -> catch MemoryError when np.concatenate on writing.
- labels can be of arbitrary type (optionally sorted)
- Have a h5features.File class inspired by h5py.File
  - Make Data a dict with the following syntax:

```
reader = h5f.Reader(file, group)
reader['item'][from_time:to_time]
reader['item'].features
reader['item'].labels
reader.keys()
```

  - Make an Item class wrapping Labels and Features
- Implement sparse functionalities
- Handle h5py MPI driver for concurent reading
- Enable autochunking from h5py (with chunk=None)
- Allow data compression as an option for the writer

CHAPTER 5

License and copyright

This package is developed whithin the Bootphon project.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

Index