pydrobert-kaldi

Sean Robertson

Feb 26, 2023

CONTENTS:

1 Documentation **3**

2 Input/Output **5**

3 Logging and CLI **7**

4 Installation **9**

5 License **11**

6 How to Cite **13**

7 Command-Line Interface **15**
- 7.1 write-table-to-pickle . . . 15
- 7.2 write-pickle-to-table . . . 16
- 7.3 compute-error-rate . . . 16
- 7.4 normalize-feat-lens . . . 17
- 7.5 write-table-to-torch-dir . . . 18
- 7.6 write-torch-dir-to-table . . . 19

8 Locale and Kaldi **21**

9 pydrobert.kaldi API **23**
- 9.1 pydrobert.kaldi.eval . . . 23
  - 9.1.1 pydrobert.kaldi.eval.util . . . 23
- 9.2 pydrobert.kaldi.io . . . 24
  - 9.2.1 pydrobert.kaldi.io.argparse . . . 25
  - 9.2.2 pydrobert.kaldi.io.corpus . . . 28
  - 9.2.3 pydrobert.kaldi.io.duck_streams . . . 35
  - 9.2.4 pydrobert.kaldi.io.enums . . . 37
  - 9.2.5 pydrobert.kaldi.io.table_streams . . . 40
  - 9.2.6 pydrobert.kaldi.io.util . . . 43
- 9.3 pydrobert.kaldi.logging . . . 45

10 Indices and tables **47**

Python Module Index **49**

Index **51**

Some Kaldi bindings for Python. I started this project because I wanted to seamlessly incorporate Kaldi's I/O mechanism into the gamut of Python-based data science packages (e.g. Theano, Tensorflow, CNTK, PyTorch, etc.). The code base is expanding to wrap more of Kaldi's feature processing and mathematical functions, but is unlikely to include modelling or decoding.

Eventually, I plan on adding hooks for Kaldi audio features and pre-/post- processing. However, I have no plans on porting any code involving modelling or decoding.

This is student-driven code, so don't expect a stable API. I'll try to use semantic versioning, but the best way to keep functionality stable is by forking.

CHAPTER

ONE

DOCUMENTATION

- Latest
- v0.6.3

CHAPTER

TWO

INPUT/OUTPUT

Most I/O can be performed with the `pydrobert.kaldi.io.open` function:

```
from pydrobert.kaldi import io
with io.open('scp:foo.scp', 'bm') as f:
    for matrix in f:
        ...
```

`open` is a factory function that determines the appropriate underlying stream to open, much like Python's built-in `open`. The data types we can read (here, a `BaseMatrix`) are listed in `pydrobert.kaldi.io.enums.KaldiDataType`. Big data types, like matrices and vectors, are piped into Numpy arrays. Passing an extended filename (e.g. paths to files on discs, `'-'` for stdin/stdout, `'gzip -c a.ark.gz |'`, etc.) opens a stream from which data types can be read one-by-one and in the order they were written. Alternatively, prepending the extended filename with `'ark[,[option_a[,option_b...]]:'` or `'scp[,...]:'` and specifying a data type allows one to open a Kaldi table for iterator-like sequential reading (`mode='r'`), dict-like random access reading (`mode='r+'`), or writing (`mode='w'`). For more information on the open function, consult the docstring.

The submodule `pydrobert.kaldi.io.corpus` contains useful wrappers around Kaldi I/O to serve up batches of data to, say, a neural network:

```
train = ShuffledData('scp:feats.scp', 'scp:labels.scp', batch_size=10)
for feat_batch, label_batch in train:
    ...
```

CHAPTER

THREE

LOGGING AND CLI

By default, Kaldi error, warning, and critical messages are piped to standard error. The `pydrobert.kaldi.logging` submodule provides hooks into python's native logging interface: the `logging` module. The :class:`KaldiLogger` can handle stack traces from Kaldi C++ code, and there are a variety of decorators to finagle the kaldi logging patterns to python logging patterns, or vice versa.

You'd likely want to explicitly handle logging when creating new kaldi-style commands for command line. `pydrobert.kaldi.io.argparse` provides :class:`KaldiParser`, an :class:`ArgumentParser` tailored to Kaldi inputs/outputs. It is used by a few command-line entry points added by this package. See the Command-Line Interface page for details.

CHAPTER

FOUR

INSTALLATION

Prepackaged binaries of tagged versions of `pydrobert-kaldi` are available for most 64-bit platforms (Windows, Glibc Linux, OSX) and most active Python versions (3.7-3.11) on both conda and PyPI.

To install via conda-forge

```
conda install -c conda-forge pydrobert-kaldi
```

If you only want to rely on Anaconda depenedencies, you can install from the `sdrobert` channel instead. There is not yet a 3.11 build there.

To install via PyPI

```
pip install pydrobert-kaldi
```

You can also try building the cutting-edge version. To do so, you'll need to first install SWIG 4.0 and an appropriate C++ compiler, then

```
pip install git+https://github.com/sdrobert/pydrobert-kaldi.git
```

The current version does not require a BLAS install, though it likely will in the future as more is wrapped.

CHAPTER

FIVE

LICENSE

This code is licensed under Apache 2.0.

Code found under the `src/` directory has been primarily copied from Kaldi. `setup.py` is also strongly influenced by Kaldi's build configuration. Kaldi is also covered by the Apache 2.0 license; its specific license file was copied into `src/COPYING_Kaldi_Project` to live among its fellows.

CHAPTER

SIX

HOW TO CITE

Please see the pydrobert page for more details.

CHAPTER

SEVEN

COMMAND-LINE INTERFACE

7.1 write-table-to-pickle

```
write-table-to-pickle -h
usage: write-table-to-pickle [-h] [-v VERBOSE] [--config CONFIG] [--print-args PRINT_
↪ARGS] [-i IN_TYPE] [-o OUT_TYPE] rspecifier value_out [key_out]

Write a kaldi table to pickle file(s)

    The inverse is write-pickle-to-table


positional arguments:
  rspecifier            The table to read
  value_out             A path to write (key,value) pairs to, or just values if key_out␣
↪was set. If it ends in ".gz", the file will be gzipped
  key_out               A path to write keys to. If it ends in ".gz", the file will be␣
↪gzipped

optional arguments:
  -h, --help            show this help message and exit
  -v VERBOSE, --verbose VERBOSE
                        Verbose level (higher->more logging)
  --config CONFIG
  --print-args PRINT_ARGS
  -i IN_TYPE, --in-type IN_TYPE
                        The type of kaldi data type to read. Defaults to base matrix
  -o OUT_TYPE, --out-type OUT_TYPE
                        The numpy data type to cast values to. The default is dependent␣
↪on the input type. String types will be written as (tuples of) strings
```

7.2 write-pickle-to-table

```
write-pickle-to-table -h
usage: write-pickle-to-table [-h] [-v VERBOSE] [--config CONFIG] [--print-args PRINT_
↪ARGS] [-o OUT_TYPE] value_in [key_in] wspecifier

Write pickle file(s) contents to a table

    The inverse is write-table-to-pickle


positional arguments:
  value_in              A path to read (key,value) pairs from, or just values if key_in␣
↪was set. If it ends in ".gz", the file is assumed to be gzipped
  key_in                A path to read keys from. If it ends in ".gz", the file is␣
↪assumed to be gzipped
  wspecifier            The table to write to

optional arguments:
  -h, --help            show this help message and exit
  -v VERBOSE, --verbose VERBOSE
                        Verbose level (higher->more logging)
  --config CONFIG
  --print-args PRINT_ARGS
  -o OUT_TYPE, --out-type OUT_TYPE
                        The type of kaldi data type to read. Defaults to base matrix
```

7.3 compute-error-rate

```
compute-error-rate -h
usage: compute-error-rate [-h] [-v VERBOSE] [--config CONFIG] [--print-args PRINT_ARGS]␣
↪[--print-tables PRINT_TABLES] [--strict STRICT] [--insertion-cost INSERTION_COST]
                          [--deletion-cost DELETION_COST] [--substitution-cost␣
↪SUBSTITUTION_COST] [--include-inserts-in-cost INCLUDE_INSERTS_IN_COST]
                          [--report-accuracy REPORT_ACCURACY]
                          ref_rspecifier hyp_rspecifier [out_path]

Compute error rates between reference and hypothesis token vectors

    Two common error rates in speech are the word (WER) and phone (PER), though the
    computation is the same. Given a reference and hypothesis sequence, the error rate
    is

        error_rate = (substitutions + insertions + deletions) / (ref_tokens * 100)

    Where the number of substitutions (e.g. "A B C -> A D C"), deletions (e.g. "A B C ->
    A C"), and insertions (e.g. "A B C -> A D B C") are determined by Levenshtein
    distance.
```

(continues on next page)

(continued from previous page)

```
positional arguments:
  ref_rspecifier        Rspecifier pointing to reference (gold standard) transcriptions
  hyp_rspecifier        Rspecifier pointing to hypothesis transcriptions
  out_path              Path to print results to. Default is stdout.

optional arguments:
  -h, --help            show this help message and exit
  -v VERBOSE, --verbose VERBOSE
                        Verbose level (higher->more logging)
  --config CONFIG
  --print-args PRINT_ARGS
  --print-tables PRINT_TABLES
                        If set, will print breakdown of insertions, deletions, and subs␣
↪to out_path
  --strict STRICT       If set, missing utterances will cause an error
  --insertion-cost INSERTION_COST
                        Cost (in terms of edit distance) to perform an insertion
  --deletion-cost DELETION_COST
                        Cost (in terms of edit distance) to perform a deletion
  --substitution-cost SUBSTITUTION_COST
                        Cost (in terms of edit distance) to perform a substitution
  --include-inserts-in-cost INCLUDE_INSERTS_IN_COST
                        Whether to include insertions in error rate calculations
  --report-accuracy REPORT_ACCURACY
                        Whether to report accuracy (1 - error_rate) instead of the error␣
↪rate
```

7.4 normalize-feat-lens

```
normalize-feat-lens -h
usage: normalize-feat-lens [-h] [-v VERBOSE] [--config CONFIG] [--print-args PRINT_ARGS]␣
↪[--type TYPE] [--tolerance TOLERANCE] [--strict STRICT]
                           [--pad-mode {zero,constant,edge,symmetric,mean}] [--side
↪{left,right,center}]
                           feats_in_rspecifier len_in_rspecifier feats_out_wspecifier

Ensure features match some reference lengths

    Incoming features are either clipped or padded to match reference lengths (stored as
    an int32 table), if they are within tolerance.


positional arguments:
  feats_in_rspecifier   The features to be normalized
  len_in_rspecifier     The reference lengths (int32 table)
  feats_out_wspecifier  The output features

optional arguments:
  -h, --help            show this help message and exit
  -v VERBOSE, --verbose VERBOSE
```

(continues on next page)

(continued from previous page)

```
                        Verbose level (higher->more logging)
  --config CONFIG
  --print-args PRINT_ARGS
  --type TYPE           The kaldi type of the input/output features
  --tolerance TOLERANCE
                        How many frames deviation from reference to tolerate before␣
↪error. The default is to be infinitely tolerant (a feat I'm sure we all desire)
  --strict STRICT       Whether missing keys in len_in and lengths beyond the threshold␣
↪cause an error (true) or are skipped with a warning (false)
  --pad-mode {zero,constant,edge,symmetric,mean}
                        If frames are being padded to the features, specify how they␣
↪should be padded. zero=zero pad, edge=pad with rightmost frame, symmetric=pad with
                        reverse of frame edges, mean=pad with mean feature values
  --side {left,right,center}
                        If an utterance needs to be padded or truncated, specify what␣
↪side of the utterance to do this on. left=beginning, right=end, center=distribute
                        evenly on either side
```

7.5 write-table-to-torch-dir

```
write-table-to-torch-dir -h
usage: write-table-to-torch-dir [-h] [-v VERBOSE] [--config CONFIG] [--print-args PRINT_
↪ARGS] [-i IN_TYPE] [-o {float,double,half,byte,char,short,int,long}]
                                [--file-prefix FILE_PREFIX] [--file-suffix FILE_SUFFIX]
                                rspecifier dir

Write a Kaldi table to a series of PyTorch data files in a directory

    Writes to a folder in the format:

        folder/
            <file_prefix><key_1><file_suffix>
            <file_prefix><key_2><file_suffix>
            ...

    The contents of the file "<file_prefix><key_1><file_suffix>" will be a PyTorch
    tensor corresponding to the entry in the table for "<key_1>"


positional arguments:
  rspecifier            The table to read
  dir                   The folder to write files to

optional arguments:
  -h, --help            show this help message and exit
  -v VERBOSE, --verbose VERBOSE
                        Verbose level (higher->more logging)
  --config CONFIG
  --print-args PRINT_ARGS
  -i IN_TYPE, --in-type IN_TYPE
```

(continues on next page)

(continued from previous page)

```
                        The type of table to read
  -o {float,double,half,byte,char,short,int,long}, --out-type {float,double,half,byte,
↪char,short,int,long}
                        The type of torch tensor to write. If unset, it is inferrred␣
↪from the input type
  --file-prefix FILE_PREFIX
                        The file prefix indicating a torch data file
  --file-suffix FILE_SUFFIX
                        The file suffix indicating a torch data file
```

7.6 write-torch-dir-to-table

```
write-torch-dir-to-table -h
usage: write-torch-dir-to-table [-h] [-v VERBOSE] [--config CONFIG] [--print-args PRINT_
↪ARGS] [-o OUT_TYPE] [--file-prefix FILE_PREFIX] [--file-suffix FILE_SUFFIX]
                                dir wspecifier

Write a data directory containing PyTorch data files to a Kaldi table

    Reads from a folder in the format:

        folder/
          <file_prefix><key_1><file_suffix>
          <file_prefix><key_2><file_suffix>
          ...

    Where each file contains a PyTorch tensor. The contents of the file
    "<file_prefix><key_1><file_suffix>" will be written as a value in a Kaldi table with
    key "<key_1>"


positional arguments:
  dir                   The folder to read files from
  wspecifier            The table to write to

optional arguments:
  -h, --help            show this help message and exit
  -v VERBOSE, --verbose VERBOSE
                        Verbose level (higher->more logging)
  --config CONFIG
  --print-args PRINT_ARGS
  -o OUT_TYPE, --out-type OUT_TYPE
                        The type of table to write to
  --file-prefix FILE_PREFIX
                        The file prefix indicating a torch data file
  --file-suffix FILE_SUFFIX
                        The file suffix indicating a torch data file
```

CHAPTER

EIGHT

LOCALE AND KALDI

After `v0.6.0`, `pydrobert.kaldi.io` no longer issues a `KaldiLocaleWarning` when the system locale doesn't match the POSIX standard. *The long story short is that locale shouldn't matter much to what pydrobert-kaldi does,* so I no longer bug you about it. If you're hunting an error, however, read on.

Most Kaldi shell scripts presume

```
export LC_ALL=C
```

has been called some time prior to running the current script. This sets the locale to POSIX-style, which is going to ensure your various shell commands sort stuff like C does. The Kaldi codebase is written in C, so it's definitely going to sort this way. Here's an example of some weirdness involving the `"s"` flag in the file rxspecifier. It basically tells Kaldi that table entries are in sorted order, which allows Kaldi to take some shortcuts to save on read/write costs.

```
# I've previously installed the German and Russian locales on Ubuntu:
# sudo locale-gen de_DE
# sudo locale-gen ru_RU

export LC_ALL=C

python -c "print('f\xe4n a'); print('foo b')" | \
  sort | \
  python -c "
from pydrobert.kaldi.io import open as kopen
with kopen('ark,s:-', 't', 'r+') as f:
    print(f['foo'])
"
# outputs: b
# sort sorts C-style ("foo" first), kaldi sorts C-style

python -c "print('f\xe4n a'); print('foo b')" | \
  LC_ALL=de_DE sort | \
  python -c "
from pydrobert.kaldi.io import open as kopen
with kopen('ark,s:-', 't', 'r+') as f:
    print(f['foo'])
"
# KeyError: 'foo'
# sort sorts German ("fän" first), kaldi sorts C-style

python -c "print('f\xe4n a'); print('foo b')" | \
  sort | \
```

(continues on next page)

(continued from previous page)

```
  LC_ALL=de_DE python -c "
from pydrobert.kaldi.io import open as kopen
with kopen('ark,s:-', 't', 'r+') as f:
    print(f['foo'])
"
# outputs: b
# sort sorts C-style, kaldi ignores German encoding and sorts C-style
```

These examples will lead to exceptions which can be caught and debugged. One can come up with more insidious errors which don't fail, mind you.

For the most part, however, this is a non-issue, at least for *pydrobert-kaldi*. The only situation the library might mess up in that I know of involves sorting table keys, and the table keys are (as far as I can tell) exclusively ASCII. Also as far as I can tell, even locales which contain characters visually identical to those in the Latin alphabet are nonetheless encoded outside of the ASCII range. For example:

```
export LC_ALL=C
echo $'M\nC' | LC_ALL=ru_RU sort
# outputs: C, M
# these are the ASCII characters
echo $'\n' | LC_ALL=ru_RU sort
# outputs: M, C
# these are UTF characters 'U+0421' and 'U+0043', respectively
```

Besides UTF, ISO-8859-1 maintains a contiguous ASCII range. Technically there's no guarantee that this will be the case for all encodings, though any such encoding would probably break all sorts of legacy code. If you have a counterexample of a Kaldi recipe that does otherwise, please let me know and I'll mention it here.

Other than that, the library is quite agnostic to locale. An error involving locales is, more likely than not, something that occurred before or after the library was called.

CHAPTER

NINE

PYDROBERT.KALDI API

Python access to kaldi

9.1 pydrobert.kaldi.eval

Tools related to evaluating models

9.1.1 pydrobert.kaldi.eval.util

Utilities for evaluation

pydrobert.kaldi.eval.util.**edit_distance**(*ref*, *hyp*, *insertion_cost=1*, *deletion_cost=1*, *substitution_cost=1*, *return_tables=False*)

Levenshtein (edit) distance

Parameters

- **ref** (Sequence) – Sequence of tokens of reference text (source)
- **hyp** (Sequence) – Sequence of tokens of hypothesis text (target)
- **insertion_cost** (int) – Penalty for *hyp* inserting a token to ref
- **deletion_cost** (int) – Penalty for *hyp* deleting a token from ref
- **substitution_cost** (int) – Penalty for *hyp* swapping tokens in ref
- **return_tables** (bool) – See below

Returns

distances (int or (int, dict, dict, dict, dict)) – Returns the edit distance of *hyp* from *ref*. If *return_tables* is *True*, this returns a tuple of the edit distance, a dict of insertion counts, a dict of deletion , a dict of substitution counts per ref token, and a dict of counts of ref tokens. Any tokens with count 0 are excluded from the dictionary.

9.2 pydrobert.kaldi.io

Interfaces for Kaldi's readers and writers

This subpackage contains a factory function, *open()*, which is intended to behave similarly to python's built-in *open()* factory. *open()* gives the specifics behind Kaldi's different read/write styles. Here, they are described in a general way.

Kaldi's streams can be very exotic, including regular files, file offsets, stdin/out, and pipes.

Data can be read/written from a binary or text stream in the usual way: specific data types have specific encodings, and data are packed/unpacked in that fashion. While an appropriate style for a fixed sequence of data, variables sequences of data are encoded using the table analogy.

Kaldi uses the table analogy to store and retrieve indexed data. In a nutshell, Kaldi uses archive ("ark") files to store binary or text data, and script files ("scp") to point *into* archives. Both use whitespace- free strings as keys. Scripts and archives do not have any built-in type checking, so it is necessary to specify the input/output type when the files are opened.

A full account of Kaldi IO can be found on Kaldi's website under Kaldi I/O Mechanisms.

See also:

pydrobert.kaldi.io.enums.KaldiDataType
For more information on the types of streams that can be read or written

class pydrobert.kaldi.io.**KaldiIOBase**(*path*)

Bases: object

IOBase for kaldi readers and writers

Similar to io.IOBase, but without a lot of the assumed functionality.

Parameters
path (str) – The path passed to "func:*pydrobert.kaldi.io.open*. One of an rspecifier, wspecifier, rxfilename, or wxfilename

path

The opened path

table_type

The type of table that's being read/written (or NotATable)

xfilenames

The extended file names being read/written. For tables, this excludes the 'ark:' and 'scp:' prefixes from path. Usually there will be only one extended file name, unless the path uses the special 'ark,scp:' format to write both an archive and script at the same time

xtypes

The type of extended file name opened. Usually there will be only one extended file name, unless the path uses the special 'ark,scp:' format to write both an archive and script at the same time

binary

Whether this stream encodes binary data (or text)

closed

Whether this stream is closed

permissive

Whether invalid values will be treated as non-existent (tables only)

once

Whether each entry will only be read once (readable tables only)

sorted

Whether keys are sorted (readable tables only)

called_sorted

Whether entries will be read in sorted order (readable tables only)

background

Whether reading is not being performed on the main thread (readable tables only)

flush

Whether the stream is flushed after each write operation (writable tables only)

abstract close()

Close and flush the underlying IO object

This method has no effect if the file is already closed

abstract readable()

Return whether this object was opened for reading

abstract writable()

Return whether this object was opened for writing

pydrobert.kaldi.io.**open**(*path*, *kaldi_dtype=None*, *mode='r'*, *error_on_str=True*, *utt2spk=''*, *value_style='b'*, *header=True*, *cache=False*)

Factory function for initializing and opening kaldi streams

This function provides a general interface for opening kaldi streams. Kaldi streams are either simple input/output of kaldi objects (the basic/duck stream) or key-value readers and writers (tables).

When *path* starts with `'ark:'` or `'scp:'` (possibly with modifiers before the colon), a table is opened. Otherwise, a basic stream is opened.

See also:

pydrobert.kaldi.io.table_streams.open_table_stream
For information on opening tables

pydrobert.kaldi.io.duck_streams.open_duck_stream
For information on opening basic streams

9.2.1 pydrobert.kaldi.io.argparse

Contains a custom ArgumentParser, KaldiParser, and a number of arg types

class pydrobert.kaldi.io.argparse.**KaldiParser**(*prog=None*, *usage=None*, *description=None*, *epilog=None*, *parents=()*, *formatter_class=<class 'argparse.HelpFormatter'>*, *prefix_chars='-'*, *fromfile_prefix_chars=None*, *argument_default=None*, *conflict_handler='error'*, *add_help=True*, *add_verbose=True*, *add_config=True*, *update_formatters=True*, *add_print_args=True*, *logger=None*, *version=None*)

Bases: ArgumentParser

Kaldi-compatible wrapper for argument parsing

KaldiParser intends to make command-line entry points in python more compatible with kaldi command-line scripts. It makes the following changes to argparse.ArgumentParser:

1. Creates a logging.Formatter instance that formats messages similarly to kaldi using the *prog* keyword as the program name.
2. Sets the default help and usage locations to sys.stderr (instead of sys.stdout)
3. Registers 'kaldi_bool', 'kaldi_rspecifier', 'kaldi_wspecifier', 'kaldi_wxfilename', 'kaldi_rxfilename', 'kaldi_config', 'kaldi_dtype', and 'numpy_dtype' as argument types
4. Registers 'kaldi_verbose' as an action
5. Adds *logger*, *update_formatters*, *add_config*, and *add_verbose* parameters to initialization (see below)
6. Wraps *parse_args* and *parse_known_args* with kaldi_vlog_level_cmd_decorator (so loggers use the right level names on error)

KaldiParser differs from kaldi's command line parsing in a few key ways. First, though '=' syntax is supported, the parser will also group using the command-line splitting (on unquoted whitespace). For the *KaldiParser*, --foo bar and --foo=bar are equivalent (assuming foo takes one optional argument), whereas, in Kaldi, --foo bar would be parsed as the boolean flag --foo followed by a positional with value bar. This ambiguity is the source of the next difference: boolean flags. Because kaldi command-line parsing splits around =, it can use --foo=true and --foo interchangeably. To avoid gobbling up a positional argument, *KaldiParser* allows for only one type of boolean flag syntax. For the former, use action='store_true' in *add_argument*. For the latter, use type='kaldi_bool'.

Parameters

- **prog** (Optional[str]) – Name of the program. Defaults to sys.argv[0]
- **usage** (Optional[str]) – A usage message. Default: auto-generated from arguments
- **description** (Optional[str]) – A description of what the program does
- **epilog** (Optional[str]) – Text following the argument descriptions
- **parents** (Sequence[ArgumentParser]) – Parsers whose arguments should be copied into this one
- **formatter_class** (type) – Class for printing help messages
- **prefix_chars** (str) – Characters that prefix optional arguments
- **fromfile_prefix_chars** (Optional[str]) – Characters that prefix files containing additional arguments
- **argument_default** (Optional[Any]) – The default value for all arguments
- **conflict_handler** (str) – String indicating how to handle conflicts
- **add_help** (bool) – Add a -h/--help option
- **add_verbose** (bool) – Add a -v/--verbose option. The option requires an integer argument specifying a verbosiy level at the same degrees as Kaldi. The level will be converted to the appropriate python level when parsed
- **add_config** (bool) – Whether to add the standard --config option to the parser. If True, a first-pass will extract all config file options and put them at the beginning of the argument string to be re-parsed.

- **add_print_args** (bool) – Whether to add the standard `--print-args` to the parser. If `True`, a first-pass of the will search for the value of `--print-args` and, if `True`, will print that value to stderr (only on *parse_args*, not *parse_known_args*)
- **update_formatters** (bool) – If *logger* is set, the logger's handlers' formatters will be set to a kaldi-style formatter
- **logger** (Optional[Logger]) – Errors will be written to this logger when parse_args fails. If *add_verbose* has been set to `True`, the logger will be set to the appropriate python level if verbose is set (note: the logger will be set to the default level - `INFO` - on initialization)
- **version** (Optional[str]) – A version string to use for logs. If not set, `pydrobert.kaldi.__version__` will be used by default

logger

The logger this parse was printing out to

formatter

A log formatter that formats with kaldi-style headers

add_config

Whether this parser has a `--config` flag

add_print_args

Whether this parser has a `--print-args` flag

version

Version string used by this parser and *logger*

error(*message*)

Prints a usage message incorporating the message to stderr and exits.

If you override this in a subclass, it should not return – it should either exit or raise an exception.

parse_known_args(***kwargs*)

print_help(*file=None*)

print_usage(*file=None*)

class pydrobert.kaldi.io.argparse.**KaldiVerbosityAction**(*option_strings*, *dest*, *default=20*, *required=False*, *help='Verbose level (higher->more logging)'*, *metavar=None*)

Bases: Action

Read kaldi-style verbosity levels, setting logger to python level

Kaldi verbosities tend to range from [-3, 9]. This action takes in a kaldi verbosity level and converts it to python logging levels with *pydrobert.kaldi.logging.kaldi_lvl_to_logging_lvl()*

If the parser has a *logger* attribute, the *logger* will be set to the new level.

pydrobert.kaldi.io.argparse.**kaldi_bool_arg_type**(*string*)

argument type for bool strings of "true","t","false", or "f"

pydrobert.kaldi.io.argparse.**kaldi_config_arg_type**(*string*)

Encapsulate parse_kaldi_config_file as an argument type

pydrobert.kaldi.io.argparse.**kaldi_dtype_arg_type**(*string*)

argument type for string reps of KaldiDataType

pydrobert.kaldi.io.argparse.**kaldi_rspecifier_arg_type**(*string*)

argument type to make sure string is a valid rspecifier

pydrobert.kaldi.io.argparse.**kaldi_rxfilename_arg_type**(*string*)

argument type to make sure string is a valid extended readable file

pydrobert.kaldi.io.argparse.**kaldi_wspecifier_arg_type**(*string*)

argument type to make sure string is a valid wspecifier

pydrobert.kaldi.io.argparse.**kaldi_wxfilename_arg_type**(*string*)

argument type to make sure string is a valid extended readable file

pydrobert.kaldi.io.argparse.**numpy_dtype_arg_type**(*string*)

argument type for string reps of numpy dtypes

pydrobert.kaldi.io.argparse.**parse_kaldi_config_file**(*file_path*, *allow_space=True*)

Return a list of arguments from a kaldi config file

Parameters

- **file_path** (`str`) – Points to the config file in question
- **allow_spaces** (`bool`, *optional*) – If `True`, treat the first space on a line as splitting key and value if no equals sign exists on the line. If `False`, no equals sign will chunk the whole line (as if a boolean flag). Kaldi does not split on spaces, but python does. Note that *allow_spaces* does not split the entire line on spaces, unlike shell arguments.

9.2.2 pydrobert.kaldi.io.corpus

Submodule for corpus iterators

class pydrobert.kaldi.io.corpus.**Data**(*table*, **additional_tables*, ***kwargs*)

Bases: `Iterable`, `Sized`

Metaclass for data iterables

A template for providing iterators over kaldi tables. They can be used like this

```
>>> data = DataSubclass(
...     'scp:feats.scp', 'scp:labels.scp', batch_size=10)
>>> for feat_batch, label_batch in data:
>>>     pass  # do something
>>> for feat_batch, label_batch in data:
>>>     pass  # do something again
```

Where *DataSubclass* is some subclass of this virtual class. Calling `iter()` on an instance (which occurs implicitly in for-loops) will generate a new iterator over the entire data set.

The class takes an arbitrary positive number of positional arguments on initialization, each a table to open. Each argument is one of:

1. An rspecifier (ideally for a script file). Assumed to be of type `KaldiDataType.BaseMatrix`
2. A sequence of length 2: the first element is the rspecifier, the second the rspecifier's `KaldiDataType`
3. A sequence of length 3: the first element is the rspecifier, the second the rspecifier's `KaldiDataType`, and the third is a dictionary to be passed as keyword arguments to the *pydrobert.kaldi.io.open()* function

All tables are assumed to index data using the same keys.

If *batch_size* is set, data are stacked in batches along a new axis. The keyword arguments *batch_axis*, *batch_pad_mode*, and any remaining keywords are sent to this module's *batch_data()* function. If *batch_size* is None or 0, samples are returned one-by-one. Data are always cast to numpy arrays before being returned. Consult that function for more information on batching.

If only one table is specified and neither *axis_lengths* or *add_key* is specified, iterators will be of a batch of the table's data directly. Otherwise, iterators yield "batches" of tuples containing "sub-batches" from each respective data source. Sub-batches belonging to the same batch share the same subset of ordered keys.

If *add_key* is True, a sub-batch of referrent keys is added as the first element of a batch tuple.

For batched sequence-to-sequence tasks, it is often important to know the original length of data before padding. Setting *axis_lengths* adds one or more sub-batches to the end of a batch tuple with this information. These sub-batches are filled with signed 32-bit integers. *axis_lengths* can be one of:

1. An integer specifying an axis from the first table to get the lengths of.
2. A pair of integers. The first element is the table index, the second is the axis index in that table.
3. A sequence of pairs of integers. Sub-batches will be appended to the batch tuple in that order

Note that axes in *axis_lengths* index the axes in individual samples, not the batch. For instance, if `batch_axis == 0` and `axis_lengths == 0`, then the last sub-batch will refer to the pre-padded value of sub-batch 0's axis 1 (`batch[0].shape[1]`).

The length of this object is the number of batches it serves per epoch.

Parameters

- **table** – The first table specifier
- **additional_tables** – Table specifiers past the first. If not empty, will iterate over tuples of sub-batches
- **add_key** – If True, will insert sub-samples into the 0th index of each sample sequence that specify the key that this sample was indexed by. Defaults to False
- **axis_lengths** – If set, sub-batches of axis lengths will be appended to the end of a batch tuple
- **batch_axis** – The axis or axes (in the case of multiple tables) along which samples are stacked in (sub-)batches. batch_axis should take into account axis length and key sub-batches when applicable. Defaults to 0
- **batch_cast_to_array** – A numpy type or sequence of types to cast each (sub-)batch to. None values indicate no casting should occur. *batch_cast_to_array* should take into acount axis length and key sub-batches when applicable
- **batch_kwargs** – Additional keyword arguments to pass to `batch_data`
- **batch_pad_mode** – If set, pads samples in (sub-)batches according to this numpy.pad() strategy when samples do not have the same length
- **batch_size** – The number of samples per (sub-)batch. Defaults to None, which means samples are served without batching
- **ignore_missing** – If True and some provided table does not have some key, that key will simply be ignored. Otherwise, a missing key raises a ValueError. Default to False

`table_specifiers`

A tuple of triples indicating `(rspecifier, kaldi_dtype, open_kwargs)` for each table

add_key

Whether a sub-batch of table keys has been prepended to existing sub-batches

axis_lengths

A tuple of pairs for each axis-length sub-batch requested. Each pair is `(sub_batch_idx, axis)`.

batch_axis

A tuple of length num_sub indicating which axis (sub-)samples will be arrayed along in a given (sub-)batch when all (sub-)samples are (or are cast to) fixed length numpy arrays of the same type

batch_cast_to_array

A tuple of length *num_sub* indicating what numpy types, if any (sub-)samples should be cast to. Values of `None` indicate no casting should be done on that (sub-)sample

batch_kwargs

Additional keyword arguments to pass to `batch_data`

batch_pad_mode

If set, pads samples in (sub-)batches according to this `numpy.pad()` strategy when samples do not have the same length

batch_size

The number of samples per (sub-)batch

ignore_missing

If `True` and some provided table does not have some key, that key will simply be ignored. Otherwise, a missing key raises a ValueError

num_sub

The number of sub-batches per batch. If > 1, batches are yielded as tuples of sub-batches. This number accounts for key, table, and axis-length sub-batches

batch_generator(*repeat=False*)

A generator which yields batches of data

Parameters
repeat (`bool`) – Whether to stop generating after one epoch (False) or keep restart and continue generating indefinitely

Yields
batch (`np.array` or `tuple`) – A batch if `self.num_sub == 1`, otherwise a tuple of sub-batches. If self.batch_size does not divide an epoch's worth of data evenly, the last batch of every epoch will be smaller

property num_batches

the number of batches yielded per epoch

This number takes into account the number of terms missing if `self.ignore_missing == True`

Type
int

abstract property num_samples

the number of samples yielded per epoch

This number takes into account the number of terms missing if `self.ignore_missing == True`

Type
int

sample_generator(*repeat=False*)

A generator which yields individual samples from data

Parameters

repeat (bool) – Whether to stop generating after one epoch (False) or keep restart and continue generating indefinitely

Yields

sample (np.array or tuple) – A sample if self.num_sub == 1, otherwise a tuple of sub-samples

abstract sample_generator_for_epoch()

A generator which yields individual samples from data for an epoch

An epoch means one pass through the data from start to finish. Equivalent to sample_generator(False).

Yields

sample (np.array or tuple) – A sample if self.num_sub == 1, otherwise a tuple of sub-samples

class pydrobert.kaldi.io.corpus.**SequentialData**(*table*, **additional_tables*, ***kwargs*)

Bases: *Data*

Provides iterators to read data sequentially

Tables are always assumed to be sorted so reading can proceed in lock-step.

Warning: Each time an iterator is requested, new sequential readers are opened. Be careful with stdin!

Parameters

- **table** – The first table specifier
- **additional_tables** – Table specifiers past the first. If not empty, will iterate over tuples of sub-batches
- **add_key** – If True, will insert sub-samples into the 0th index of each sample sequence that specify the key that this sample was indexed by. Defaults to False
- **axis_lengths** – If set, sub-batches of axis lengths will be appended to the end of a batch tuple
- **batch_axis** – The axis or axes (in the case of multiple tables) along which samples are stacked in (sub-)batches. batch_axis should take into account axis length and key sub-batches when applicable. Defaults to 0
- **batch_cast_to_array** – A numpy type or sequence of types to cast each (sub-)batch to. None values indicate no casting should occur. *batch_cast_to_array* should take into acount axis length and key sub-batches when applicable
- **batch_kwargs** – Additional keyword arguments to pass to batch_data
- **batch_pad_mode** – If set, pads samples in (sub-)batches according to this numpy.pad() strategy when samples do not have the same length
- **batch_size** – The number of samples per (sub-)batch. Defaults to None, which means samples are served without batching
- **ignore_missing** – If True and some provided table does not have some key, that key will simply be ignored. Otherwise, a missing key raises a ValueError. Default to False

table_specifiers

A tuple of triples indicating `(rspecifier, kaldi_dtype, open_kwargs)` for each table

add_key

Whether a sub-batch of table keys has been prepended to existing sub-batches

axis_lengths

A tuple of pairs for each axis-length sub-batch requested. Each pair is `(sub_batch_idx, axis)`.

batch_axis

A tuple of length num_sub indicating which axis (sub-)samples will be arrayed along in a given (sub-)batch when all (sub-)samples are (or are cast to) fixed length numpy arrays of the same type

batch_cast_to_array

A tuple of length *num_sub* indicating what numpy types, if any (sub-)samples should be cast to. Values of `None` indicate no casting should be done on that (sub-)sample

batch_kwargs

Additional keyword arguments to pass to `batch_data`

batch_pad_mode

If set, pads samples in (sub-)batches according to this `numpy.pad()` strategy when samples do not have the same length

batch_size

The number of samples per (sub-)batch

ignore_missing

If `True` and some provided table does not have some key, that key will simply be ignored. Otherwise, a missing key raises a ValueError

num_sub

The number of sub-batches per batch. If > 1, batches are yielded as tuples of sub-batches. This number accounts for key, table, and axis-length sub-batches

property num_samples

the number of samples yielded per epoch

This number takes into account the number of terms missing if `self.ignore_missing == True`

Type
int

sample_generator_for_epoch()

A generator which yields individual samples from data for an epoch

An epoch means one pass through the data from start to finish. Equivalent to `sample_generator(False)`.

Yields
sample (`np.array` or `tuple`) – A sample if `self.num_sub == 1`, otherwise a tuple of sub-samples

class pydrobert.kaldi.io.corpus.**ShuffledData**(*table*, **additional_tables*, ***kwargs*)

Bases: *Data*

Provides iterators over shuffled data

A master list of keys is either provided by keyword argument or inferred from the first table. Every new iterator requested shuffles that list of keys and returns batches in that order. Appropriate for training data.

Notes

For efficiency, it is highly recommended to use scripts to access tables rather than archives.

Parameters

- **table** – The first table specifier
- **additional_tables** – Table specifiers past the first. If not empty, will iterate over tuples of sub-batches
- **add_key** – If `True`, will insert sub-samples into the 0th index of each sample sequence that specify the key that this sample was indexed by. Defaults to `False`
- **axis_lengths** – If set, sub-batches of axis lengths will be appended to the end of a batch tuple
- **batch_axis** – The axis or axes (in the case of multiple tables) along which samples are stacked in (sub-)batches. batch_axis should take into account axis length and key sub-batches when applicable. Defaults to `0`
- **batch_cast_to_array** – A numpy type or sequence of types to cast each (sub-)batch to. `None` values indicate no casting should occur. *batch_cast_to_array* should take into acount axis length and key sub-batches when applicable
- **batch_kwargs** – Additional keyword arguments to pass to `batch_data`
- **batch_pad_mode** – If set, pads samples in (sub-)batches according to this `numpy.pad()` strategy when samples do not have the same length
- **batch_size** – The number of samples per (sub-)batch. Defaults to `None`, which means samples are served without batching
- **ignore_missing** – If `True` and some provided table does not have some key, that key will simply be ignored. Otherwise, a missing key raises a ValueError. Default to `False`
- **key_list** – A master list of keys. No other keys will be queried. If not specified, the key list will be inferred by passing through the first table once
- **rng** – Either a `numpy.random.RandomState` object or a seed to create one. It will be used to shuffle the list of keys

table_specifiers

A tuple of triples indicating `(rspecifier, kaldi_dtype, open_kwargs)` for each table

add_key

Whether a sub-batch of table keys has been prepended to existing sub-batches

axis_lengths

A tuple of pairs for each axis-length sub-batch requested. Each pair is `(sub_batch_idx, axis)`.

batch_axis

A tuple of length num_sub indicating which axis (sub-)samples will be arrayed along in a given (sub-)batch when all (sub-)samples are (or are cast to) fixed length numpy arrays of the same type

batch_cast_to_array

A tuple of length *num_sub* indicating what numpy types, if any (sub-)samples should be cast to. Values of `None` indicate no casting should be done on that (sub-)sample

batch_kwargs

Additional keyword arguments to pass to `batch_data`

batch_pad_mode

If set, pads samples in (sub-)batches according to this `numpy.pad()` strategy when samples do not have the same length

batch_size

The number of samples per (sub-)batch

ignore_missing

If `True` and some provided table does not have some key, that key will simply be ignored. Otherwise, a missing key raises a ValueError

num_sub

The number of sub-batches per batch. If > 1, batches are yielded as tuples of sub-batches. This number accounts for key, table, and axis-length sub-batches

key_list

The master list of keys

rng

Used to shuffle the list of keys every epoch

table_holders

A tuple of table readers opened in random access mode

property num_samples

the number of samples yielded per epoch

This number takes into account the number of terms missing if `self.ignore_missing == True`

Type
int

sample_generator_for_epoch()

int : the number of samples yielded per epoch

This number takes into account the number of terms missing if `self.ignore_missing == True`

pydrobert.kaldi.io.corpus.**batch_data**(*input_iter*, *subsamples=True*, *batch_size=None*, *axis=0*, *cast_to_array=None*, *pad_mode=None*, ***pad_kwargs*)

Generate batched data from an input generator

Takes some fixed number of samples from *input_iter*, encapsulates them, and yields them.

If *subsamples* is `True`, data from *input_iter* are expected to be encapsulated in fixed-length sequences (e.g. `(feat, label, len)`). Each sample will be batched separately into a sub-batch and returned in a tuple (e.g. `(feat_batch, label_batch, len_batch)`).

The format of a (sub-)batch depends on the properties of its samples:

1. If *cast_to_array* applies to this sub-batch, cast it to a numpy array of the target type.
2. If all samples in the (sub-)batch are numpy arrays of the same type and shape, samples are stacked in a bigger numpy array along the axis specified by *axis* (see Parameters).
3. If all samples are numpy arrays of the same type but variable length and *pad_mode* is specified, pad all sample arrays to the right such that they all have the same (supremum) shape, then perform 2.
4. Otherwise, simply return a list of samples as-is (ignoring axis).

Parameters

- **input_iter** (`Iterator`) – An iterator over samples

- **subsamples** (bool) – *input_iter* yields tuples to be divided into different sub-batches if True
- **batch_size** (Optional[int]) – The size of batches, except perhaps the last one. If not set or 0, will yield samples (casting and encapsulating in tuples when necessary)
- **axis** (int) – Where to insert the batch index/indices into the shape/shapes of the inputs. If a sequence, *subsamples* must be True and *input_iter* should yield samples of the same length as axis. If an int and subsamples is True, the same axis will be used for all sub-samples.
- **cast_to_array** (Union[dtype, Sequence, None]) – Dictates whether data should be cast to numpy arrays and of what type. If a sequence, *subsamples* must be True and *input_iter* should yield samples of the same length as *cast_to_array*. If a single value and *subsamples* is True, the same value will be used for all sub-samples. Value(s) of None indicate no casting should be done for this (sub-)sample. Other values will be used to cast (sub-)samples to numpy arrays
- **pad_mode** (Union[str, Callable, None]) – If set, inputs within a batch will be padded on the end to match the largest shapes in the batch. How the inputs are padded matches the argument to numpy.pad(). If not set, will raise a ValueError if they don't all have the same shape
- **pad_kwargs** – Additional keyword arguments are passed along to numpy.pad() if padding.

See also:

numpy.pad
: For different pad modes and options

9.2.3 pydrobert.kaldi.io.duck_streams

Submodule for reading and writing one-by-one, like (un)packing c structs

class pydrobert.kaldi.io.duck_streams.**KaldiInput**(*path*, *header=True*)

Bases: *KaldiIOBase*

A kaldi input stream from which objects can be read one at a time

Parameters

- **path** (str) – An extended readable file path
- **header** (bool) – If False, no attempt will be made to look for the "binary" header in the stream; it will be assumed binary

close()

Close and flush the underlying IO object

This method has no effect if the file is already closed

read(*kaldi_dtype*, *value_style='b'*, *read_binary=None*)

Read in one object from the stream

Parameters

- **kaldi_dtype** (*KaldiDataType*) – The type of object to read
- **value_style** (Literal['b', 's', 'd']) – 'wm' readers can provide not only the audio buffer ('b') of a wave file, but its sampling rate ('s'), and/or duration (in sec, 'd'). Setting *value_style* to some combination of 'b', 's', and/or 'd' will cause the reader to

return a tuple of that information. If *value_style* is only one character, the result will not be contained in a tuple

- **read_binary** (bool, *optional*) – If set, the object will be read as either binary (True) or text (False). The default behaviour is to read according to the *binary* attribute. Ignored if there's only one way to read the data

readable()

Return whether this object was opened for reading

writable()

Return whether this object was opened for writing

class pydrobert.kaldi.io.duck_streams.**KaldiOutput**(*path*, *header=True*)

Bases: *KaldiIOBase*

A kaldi output stream from which objects can be written one at a time

Parameters

- **path** (str) – An extended writable file path
- **header** (bool) – Whether to write a header when opening the binary stream (True) or not.

close()

readable()

Return whether this object was opened for reading

writable()

Return whether this object was opened for writing

write(*obj*, *kaldi_dtype=None*, *error_on_str=True*, *write_binary=True*)

Write one object to the stream

Parameters

- **obj** (Any) – The object to write
- **kaldi_dtype** (Optional[*KaldiDataType*]) – The type of object to write
- **error_on_str** (bool) – Token vectors (`'tv'`) accept sequences of whitespace-free ASCII/UTF strings. A `str` is also a sequence of characters, which may satisfy the token requirements. If *error_on_str* is True, a ValueError is raised when writing a `str` as a token vector. Otherwise a `str` can be written
- **write_binary** (bool) – The object will be written as binary (True) or text (False)

Raises

ValueError – If unable to determine a proper data type

See also:

pydrobert.kaldi.io.util.infer_kaldi_data_type
Illustrates how different inputs are mapped to data types

pydrobert.kaldi.io.duck_streams.**open_duck_stream**(*path*, *mode='r'*, *header=True*)

Open a "duck" stream

"Duck" streams provide an interface for reading or writing kaldi objects, one at a time. Essentially: remember the order things go in, then pull them out in the same order.

Duck streams can read/write binary or text data. It is mostly up to the user how to read or write data, though the following rules establish the default:

1. An input stream that does not look for a 'binary header' is binary
2. An input stream that looks for and finds a binary header when opening is binary
3. An input stream that looks for but does not find a binary header when opening is a text stream
4. An output stream is always binary. However, the user may choose not to write a binary header. The resulting input stream will be considered a text stream when 3. is satisfied

Parameters

- **path** (str) – The extended file name to be opened. This can be quite exotic. More details can be found on the Kaldi website.
- **mode** (Literal['r', 'r+', 'w']) – Whether to open the stream for input ('r') or output ('w'). 'r+' is equivalent to 'r'
- **header** (bool) – Setting this to True will either check for a 'binary header' in an input stream, or write a binary header for an output stream. If False, no check/write is performed

9.2.4 pydrobert.kaldi.io.enums

Kaldi enumerations, including data types and xspecifier types

class pydrobert.kaldi.io.enums.**KaldiDataType**(*value*)

Bases: Enum

Enumerates the data types stored and retrieved by Kaldi I/O

This enumerable lists the types of data written and read to various readers and writers. It is used in the factory method *pydrobert.kaldi.io.open()* to dictate the subclass created.

Notes

The "base float" mentioned in this documentation is the same type as `kaldi::BaseFloat`, which was determined when Kaldi was built. The easiest way to determine whether this is a double (64-bit) or a float (32-bit) is by checking the value of `KaldiDataType.BaseVector.is_double()`

Base = 'b'

Inputs/outputs are single base floats

BaseMatrix = 'bm'

Inputs/outputs are 2D numpy arrays of the base float

BasePairVector = 'bpv'

Inputs/outputs are tuples of pairs of the base float

BaseVector = 'bv'

Inputs/outputs are 1D numpy arrays of the base float

Bool = 'B'

Inputs/outputs are single booleans

Double = 'd'

Inputs/outputs are single 64-bit floats

`DoubleMatrix = 'dm'`

Inputs/outputs are 2D numpy arrays of 64-bit floats

`DoubleVector = 'dv'`

Inputs/outputs are 1D numpy arrays of 64-bit floats

`FloatMatrix = 'fm'`

Inputs/outputs are 2D numpy arrays of 32-bit floats

`FloatVector = 'fv'`

Inputs/outputs are 1D numpy arrays of 32-bit floats

`Int32 = 'i'`

Inputs/outputs are single 32-bit ints

`Int32PairVector = 'ipv'`

Inputs/outputs are tuples of pairs of 32-bit ints

`Int32Vector = 'iv'`

Inputs/outputs are tuples of 32-bit ints

`Int32VectorVector = 'ivv'`

Inputs/outputs are tuples of tuples of 32-bit ints

`Token = 't'`

Inputs/outputs are individual whitespace-free ASCII or unicode words

`TokenVector = 'tv'`

Inputs/outputs are tuples of tokens

`WaveMatrix = 'wm'`

Inputs/outputs are wave file data, cast to base float 2D arrays

Wave matrices have the shape `(n_channels, n_samples)`. Kaldi will read PCM wave files, but will always convert the samples the base floats.

Though Kaldi can read wave files of different types and sample rates, Kaldi will only write wave files as PCM16 sampled at 16k.

`property is_basic`

whether data are stored in kaldi with Read/WriteBasicType

Type
bool

`property is_double`

whether this data type is double precision (64-bit)

Type
bool

`property is_floating_point`

whether this type has a floating point representation

Type
bool

property is_matrix

whether this type is a numpy matrix type

Type
bool

property is_num_vector

whether this is a numpy vector

Type
bool

class pydrobert.kaldi.io.enums.**RxfilenameType**(*value*)

Bases: Enum

The type of stream to read, based on an extended filename

FileInput = 1

Input is from a file on disk with no offset

InvalidInput = 0

An invalid stream

OffsetFileInput = 4

Input is from a file on disk, read from a specific offset

PipedInput = 3

Input is being piped from a command

StandardInput = 2

Input is being piped from stdin

class pydrobert.kaldi.io.enums.**TableType**(*value*)

Bases: Enum

The type of table a stream points to

ArchiveTable = 1

The stream points to an archive (keys and values)

BothTables = 3

The stream points simultaneously to a script and archive

This is a special pattern for writing. The archive stores keys and values; the script stores keys and points to the locations in the archive

NotATable = 0

The stream is not a table

ScriptTable = 2

The stream points to a script (keys and extended file names)

class pydrobert.kaldi.io.enums.**WxfilenameType**(*value*)

Bases: Enum

The type of stream to write, based on an extended filename

FileOutput = 1

Output to a file on disk

InvalidOutput = 0

An invalid stream

PipedOutput = 3

Output is being piped to some command

StandardOutput = 2

Output is being piped to stdout

9.2.5 pydrobert.kaldi.io.table_streams

Submodule containing table readers and writers

class pydrobert.kaldi.io.table_streams.**KaldiRandomAccessReader**(*path*, *kaldi_dtype*, *utt2spk=''*)

Bases: *KaldiTable*, Container

Read-only access to values of table by key

KaldiRandomAccessReader objects can access values of a table through either the *get()* method or square bracket access (e.g. `a[key]`). The presence of a key can be checked with "in" syntax (e.g. `key in a`). Unlike a `dict`, the extent of a *KaldiRandomAccessReader* is not known beforehand, so neither iterators nor length methods are implemented.

Parameters

- **path** (`str`) – An rspecifier to read tables from
- **kaldi_dtype** (*KaldiDataType*) – The data type to read
- **utt2spk** (`str`) – If set, the reader uses *utt2spk* as a map from utterance ids to speaker ids. The data in *path*, which are assumed to be referenced by speaker ids, can then be refrenced by utterance. If *utt2spk* is unspecified, the keys in *path* are used to query for data.

utt2spk

The path to the map from utterance ids to speaker ids, if set

Type
`str` or `None`

get(*key*, *default=None*)

Raises
IOError – If closed

readable()

Return whether this object was opened for reading

writable()

Return whether this object was opened for writing

class pydrobert.kaldi.io.table_streams.**KaldiSequentialReader**(*path*, *kaldi_dtype*)

Bases: *KaldiTable*, Iterator

Abstract class for iterating over table entries

KaldiSequentialReader iterates over key-value pairs. The default behaviour (i.e. that in a for-loop) is to iterate over the values in order of access. Similar to `dict` instances, *items()*, *values()*, and *keys()* return iterators over their respective domains. Alternatively, the *move()* method moves to the next pair, at which point the *key()* and *value()* methods can be queried.

Though it is possible to mix and match access patterns, all methods refer to the same underlying iterator (the *KaldiSequentialReader*)

Parameters

- **path** (str) – An rspecifier to read the table from
- **kaldi_dtype** (*KaldiDataType*) – The data type to read

Yields

object or (str, object) – Values or key, value pairs

abstract done()

bool: True when closed or pairs are exhausted

items()

Returns iterator over key, value pairs

abstract key()

return current pair's key, or None if done

Raises

IOError – If closed

keys()

Returns iterator over keys

abstract move()

Move iterator forward

Returns

moved (bool) – True if moved to new pair. False if done

Raises

IOError – If closed

readable()

Return whether this object was opened for reading

abstract value()

return current pair's value, or None if done

Raises

IOError – If closed

values()

Returns iterator over values

writable()

Return whether this object was opened for writing

class pydrobert.kaldi.io.table_streams.**KaldiTable**(*path*, *kaldi_dtype*)

Bases: *KaldiIOBase*

Base class for interacting with tables

All table readers and writers are subclasses of *KaldiTable*. Tables must specify the type of data being read ahead of time

Parameters

- **path** (str) – An rspecifier or wspecifier

- **kaldi_dtype** (*KaldiDataType*) – The type of data type this table contains

kaldi_dtype

The table's data type

Type
KaldiDataType

Raises
IOError – If unable to open table

class pydrobert.kaldi.io.table_streams.**KaldiWriter**(*path*, *kaldi_dtype*)

Bases: *KaldiTable*

Write key-value pairs to tables

Parameters

- **path** (str) – An rspecifier to write the table to
- **kaldi_dtype** (*pydrobert.kaldi.io.enums.KaldiDataType*) – The data type to write

readable()

Return whether this object was opened for reading

writable()

Return whether this object was opened for writing

abstract write(*key*, *value*)

Write key value pair

Parameters

- **key** (str) –
- **value** (Any) –

Notes

For Kaldi's table writers, pairs are written in order without backtracking. Uniqueness is not checked.

pydrobert.kaldi.io.table_streams.**open_table_stream**(*path*, *kaldi_dtype*, *mode='r'*, *error_on_str=True*, *utt2spk=''*, *value_style='b'*, *cache=False*)

Factory function to open a kaldi table

This function finds the correct *KaldiTable* according to the args *kaldi_dtype* and *mode*. Specific combinations allow for optional parameters outlined by the table below

mode	*kaldi_dtype*	additional kwargs
'r'	'wm'	value_style='b'
'r+'	•	utt2spk=''
'r+'	'wm'	value_style='b'
'w'	'tv'	error_on_str=True

Parameters

- **path** (str) – The specifier used by kaldi to open the script. Generally these will take the form '{ark|scp}:<path_to_file>', though they can take much more interesting forms (like pipes). More information can be found on the Kaldi website
- **kaldi_dtype** (*KaldiDataType*) – The type of data the table is expected to handle
- **mode** (Literal['r', 'r+', 'w']) – Specifies the type of access to be performed: read sequential, read random, or write. They are implemented by subclasses of *KaldiSequentialReader*, *KaldiRandomAccessReader*, or *KaldiWriter*, resp.
- **error_on_str** (bool) – Token vectors ('tv') accept sequences of whitespace-free ASCII/UTF strings. A str is also a sequence of characters, which may satisfy the token requirements. If *error_on_str* is True, a ValueError is raised when writing a str as a token vector. Otherwise a str can be written
- **utt2spk** (str) – If set, the reader uses *utt2spk* as a map from utterance ids to speaker ids. The data in *path*, which are assumed to be referenced by speaker ids, can then be refrenced by utterance. If *utt2spk* is unspecified, the keys in *path* are used to query for data
- **value_style** (str) –

  Wave readers can provide not only the audio buffer ('b') of a wave file, but
  its sampling rate ('s'), and/or duration (in sec, 'd'). Setting *value_style* to some combination of 'b', 's', and/or 'd' will cause the reader to return a tuple of that information. If *value_style* is only one character, the result will not be contained in a tuple.

  cache
  Whether to cache all values in a dict as they are retrieved. Only applicable to random access readers. This can be very expensive for large tables and redundant if reading from an archive directly (as opposed to a script).

Returns
table (*KaldiTable*) – A table, opened.

Raises
IOError – On failure to open

9.2.6 pydrobert.kaldi.io.util

Kaldi I/O utilities

pydrobert.kaldi.io.util.**infer_kaldi_data_type**(*obj*)

Infer the appropriate kaldi data type for this object

The following map is used (in order):

Object	KaldiDataType
an int	Int32
a boolean	Bool
a float*	Base
str	Token
2-dim numpy array float32	FloatMatrix
1-dim numpy array float32	FloatVector
2-dim numpy array float64	DoubleMatrix
1-dim numpy array float64	DoubleVector
1-dim numpy array of int32	Int32Vector
2-dim numpy array of int32*	Int32VectorVector
(matrix-like, float or int)	WaveMatrix**
an empty container	BaseMatrix
container of str	TokenVector
1-dim py container of ints	Int32Vector
2-dim py container of ints*	Int32VectorVector
2-dim py container of pairs of floats	BasePairVector
matrix-like python container	DoubleMatrix
vector-like python container	DoubleVector

*The same data types could represent a `Double` or an `Int32PairVector`, respectively. Care should be taken in these cases.

**The first element is the wave data, the second its sample frequency. The wave data can be a 2d numpy float array of the same precision as `KaldiDataType.BaseMatrix`, or a matrix-like python container of floats and/or ints.

Returns
pydrobert.kaldi.io.enums.KaldiDataType or `None`

pydrobert.kaldi.io.util.**parse_kaldi_input_path**(*path*)

Determine the characteristics of an input stream by its path

Returns a 4-tuple of the following information:

1. If path is not an rspecifier (`TableType.NotATable`):
   a. Classify path as an rxfilename
   b. return a tuple of `(TableType, path, RxfilenameType, dict())`
2. else:
   a. Put all rspecifier options (once, sorted, called_sorted, permissive, background) into a dictionary
   b. Extract the embedded rxfilename and classify it
   c. return a tuple of `(TableType, rxfilename, RxfilenameType, options)`

Parameters
path (`str`) – A string that would be passed to `pydrobert.kaldi.io.open`

pydrobert.kaldi.io.util.**parse_kaldi_output_path**(*path*)

Determine the charactersistics of an output stram by its path

Returns a 4-tuple of the following information

1. If path is not a wspecifier (`TableType.NotATable`)

a. Classify path as a wxfilename

b. return a tuple of `(TableType, path, WxfilenameType, dict())`

2. If path is an archive or script

a. Put all wspecifier options (binary, flush, permissive) into a dictionary

b. Extract the embedded wxfilename and classify it

c. return a tuple of `(TableType, wxfilename, WxfilenameType, options)`

3. If path contains both an archive and a script (`TableType.BothTables`)

a. Put all wspecifier options (binary, flush, permissive) into a dictionary

b. Extract both embedded wxfilenames and classify them

c. return a tuple of `(TableType, (arch_wxfilename, script_wxfilename), (arch_WxfilenameType, script_WxfilenameType), options)`

Parameters
path (str) – A string that would be passed to *pydrobert.kaldi.io.open()*

9.3 pydrobert.kaldi.logging

Tie Kaldi's logging into python's builtin logging module

By default, Kaldi's warning, error, and critical messages are all piped directly to stderr. Any `logging.Logger` instance can register with `register_logger_for_kaldi` to receive Kaldi messages. If some logger is registered to receive Kaldi messages, messages will no longer be sent to stderr by default. Kaldi codes are converted to `logging` codes according to the following chart

logging	kaldi
CRITICAL(50+)	-3+
ERROR(40-49)	-2
WARNING(30-39)	-1
INFO(20-29)	0
DEBUG(10-19)	1
9 down to 1	2 up to 10

class pydrobert.kaldi.logging.**KaldiLogger**(*name*, *level=0*)

Bases: Logger

Logger subclass that overwrites log info with kaldi's

Setting the `Logger` class of the python module `logging` (thru `logging.setLoggerClass`) to `KaldiLogger` will allow new loggers to intercept messages from Kaldi and inject Kaldi's trace information into the record. With this injection, the logger will point to the location in Kaldi's source that the message originated from. Without it, the logger will point to a location within this submodule (`pydrobert.kaldi.logging`).

makeRecord(*name*, *lvl*, *fn*, *lno*, *msg*, *args*, *exc_info*, *func=None*, *extra=None*, *sinfo=None*)

Instances of the Logger class represent a single logging channel. A "logging channel" indicates an area of an application. Exactly how an "area" is defined is up to the application developer. Since an application can have any number of areas, logging channels are identified by a unique string. Application areas can be nested (e.g. an area of "input processing" might include sub-areas "read CSV files", "read XLS files" and "read Gnumeric files"). To cater for this natural nesting, channel names are organized into a namespace hierarchy where levels are separated by periods, much like the Java or Python package namespace. So in

the instance given above, channel names might be "input" for the upper level, and "input.csv", "input.xls" and "input.gnu" for the sub-levels. There is no arbitrary limit to the depth of nesting.

pydrobert.kaldi.logging.**deregister_logger_for_kaldi**(*name*)

Deregister logger previously registered w register_logger_for_kaldi

pydrobert.kaldi.logging.**kaldi_lvl_to_logging_lvl**(*lvl*)

Convert kaldi level to logging level

pydrobert.kaldi.logging.**kaldi_vlog_level_cmd_decorator**(*func*)

Decorator to rename, then revert, level names according to Kaldi[1]

See *pydrobert.kaldi.logging* for the conversion chart. After the return of the function, the level names before the call are reverted. This function is insensitive to renaming while the function executes

References

pydrobert.kaldi.logging.**logging_lvl_to_kaldi_lvl**(*lvl*)

Convert logging level to kaldi level

pydrobert.kaldi.logging.**register_logger_for_kaldi**(*logger*)

Register logger to receive Kaldi's messages

See module docstring for more info

Parameters

logger (str or logger) – Either the logger or its name. When a new message comes along from Kaldi, the callback will send a message to the logger

[1] Povey, D., et al (2011). The Kaldi Speech Recognition Toolkit. ASRU

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

pydrobert.kaldi, 23
pydrobert.kaldi.eval, 23
pydrobert.kaldi.eval.util, 23
pydrobert.kaldi.io, 24
pydrobert.kaldi.io.argparse, 25
pydrobert.kaldi.io.corpus, 28
pydrobert.kaldi.io.duck_streams, 35
pydrobert.kaldi.io.enums, 37
pydrobert.kaldi.io.table_streams, 40
pydrobert.kaldi.io.util, 43
pydrobert.kaldi.logging, 45

INDEX

A

add_config (*pydrobert.kaldi.io.argparse.KaldiParser attribute*), 27
add_key (*pydrobert.kaldi.io.corpus.Data attribute*), 29
add_key (*pydrobert.kaldi.io.corpus.SequentialData attribute*), 32
add_key (*pydrobert.kaldi.io.corpus.ShuffledData attribute*), 33
add_print_args (*pydrobert.kaldi.io.argparse.KaldiParser attribute*), 27
ArchiveTable (*pydrobert.kaldi.io.enums.TableType attribute*), 39
axis_lengths (*pydrobert.kaldi.io.corpus.Data attribute*), 30
axis_lengths (*pydrobert.kaldi.io.corpus.SequentialData attribute*), 32
axis_lengths (*pydrobert.kaldi.io.corpus.ShuffledData attribute*), 33

B

background (*pydrobert.kaldi.io.KaldiIOBase attribute*), 25
Base (*pydrobert.kaldi.io.enums.KaldiDataType attribute*), 37
BaseMatrix (*pydrobert.kaldi.io.enums.KaldiDataType attribute*), 37
BasePairVector (*pydrobert.kaldi.io.enums.KaldiDataType attribute*), 37
BaseVector (*pydrobert.kaldi.io.enums.KaldiDataType attribute*), 37
batch_axis (*pydrobert.kaldi.io.corpus.Data attribute*), 30
batch_axis (*pydrobert.kaldi.io.corpus.SequentialData attribute*), 32
batch_axis (*pydrobert.kaldi.io.corpus.ShuffledData attribute*), 33
batch_cast_to_array (*pydrobert.kaldi.io.corpus.Data attribute*), 30
batch_cast_to_array (*pydrobert.kaldi.io.corpus.SequentialData attribute*), 32
batch_cast_to_array (*pydrobert.kaldi.io.corpus.ShuffledData attribute*), 33
batch_data() (*in module pydrobert.kaldi.io.corpus*), 34
batch_generator() (*pydrobert.kaldi.io.corpus.Data method*), 30
batch_kwargs (*pydrobert.kaldi.io.corpus.Data attribute*), 30
batch_kwargs (*pydrobert.kaldi.io.corpus.SequentialData attribute*), 32
batch_kwargs (*pydrobert.kaldi.io.corpus.ShuffledData attribute*), 33
batch_pad_mode (*pydrobert.kaldi.io.corpus.Data attribute*), 30
batch_pad_mode (*pydrobert.kaldi.io.corpus.SequentialData attribute*), 32
batch_pad_mode (*pydrobert.kaldi.io.corpus.ShuffledData attribute*), 33
batch_size (*pydrobert.kaldi.io.corpus.Data attribute*), 30
batch_size (*pydrobert.kaldi.io.corpus.SequentialData attribute*), 32
batch_size (*pydrobert.kaldi.io.corpus.ShuffledData attribute*), 34
binary (*pydrobert.kaldi.io.KaldiIOBase attribute*), 24
Bool (*pydrobert.kaldi.io.enums.KaldiDataType attribute*), 37
BothTables (*pydrobert.kaldi.io.enums.TableType attribute*), 39

C

called_sorted (*pydrobert.kaldi.io.KaldiIOBase attribute*), 25
close() (*pydrobert.kaldi.io.duck_streams.KaldiInput method*), 35
close() (*pydrobert.kaldi.io.duck_streams.KaldiOutput method*), 36
close() (*pydrobert.kaldi.io.KaldiIOBase method*), 25
closed (*pydrobert.kaldi.io.KaldiIOBase attribute*), 24

D

Data (*class in pydrobert.kaldi.io.corpus*), 28

deregister_logger_for_kaldi() (*in module pydrobert.kaldi.logging*), 46
done() (*pydrobert.kaldi.io.table_streams.KaldiSequentialReader method*), 41
Double (*pydrobert.kaldi.io.enums.KaldiDataType attribute*), 37
DoubleMatrix (*pydrobert.kaldi.io.enums.KaldiDataType attribute*), 37
DoubleVector (*pydrobert.kaldi.io.enums.KaldiDataType attribute*), 38

E

edit_distance() (*in module pydrobert.kaldi.eval.util*), 23
error() (*pydrobert.kaldi.io.argparse.KaldiParser method*), 27

F

FileInput (*pydrobert.kaldi.io.enums.RxfilenameType attribute*), 39
FileOutput (*pydrobert.kaldi.io.enums.WxfilenameType attribute*), 39
FloatMatrix (*pydrobert.kaldi.io.enums.KaldiDataType attribute*), 38
FloatVector (*pydrobert.kaldi.io.enums.KaldiDataType attribute*), 38
flush (*pydrobert.kaldi.io.KaldiIOBase attribute*), 25
formatter (*pydrobert.kaldi.io.argparse.KaldiParser attribute*), 27

G

get() (*pydrobert.kaldi.io.table_streams.KaldiRandomAccessReader method*), 40

I

ignore_missing (*pydrobert.kaldi.io.corpus.Data attribute*), 30
ignore_missing (*pydrobert.kaldi.io.corpus.SequentialData attribute*), 32
ignore_missing (*pydrobert.kaldi.io.corpus.ShuffledData attribute*), 34
infer_kaldi_data_type() (*in module pydrobert.kaldi.io.util*), 43
Int32 (*pydrobert.kaldi.io.enums.KaldiDataType attribute*), 38
Int32PairVector (*pydrobert.kaldi.io.enums.KaldiDataType attribute*), 38
Int32Vector (*pydrobert.kaldi.io.enums.KaldiDataType attribute*), 38
Int32VectorVector (*pydrobert.kaldi.io.enums.KaldiDataType attribute*), 38
InvalidInput (*pydrobert.kaldi.io.enums.RxfilenameType attribute*), 39
InvalidOutput (*pydrobert.kaldi.io.enums.WxfilenameType attribute*), 39
is_basic (*pydrobert.kaldi.io.enums.KaldiDataType property*), 38
is_double (*pydrobert.kaldi.io.enums.KaldiDataType property*), 38
is_floating_point (*pydrobert.kaldi.io.enums.KaldiDataType property*), 38
is_matrix (*pydrobert.kaldi.io.enums.KaldiDataType property*), 38
is_num_vector (*pydrobert.kaldi.io.enums.KaldiDataType property*), 39
items() (*pydrobert.kaldi.io.table_streams.KaldiSequentialReader method*), 41

K

kaldi_bool_arg_type() (*in module pydrobert.kaldi.io.argparse*), 27
kaldi_config_arg_type() (*in module pydrobert.kaldi.io.argparse*), 27
kaldi_dtype (*pydrobert.kaldi.io.table_streams.KaldiTable attribute*), 42
kaldi_dtype_arg_type() (*in module pydrobert.kaldi.io.argparse*), 27
kaldi_lvl_to_logging_lvl() (*in module pydrobert.kaldi.logging*), 46
kaldi_rspecifier_arg_type() (*in module pydrobert.kaldi.io.argparse*), 27
kaldi_rxfilename_arg_type() (*in module pydrobert.kaldi.io.argparse*), 28
kaldi_vlog_level_cmd_decorator() (*in module pydrobert.kaldi.logging*), 46
kaldi_wspecifier_arg_type() (*in module pydrobert.kaldi.io.argparse*), 28
kaldi_wxfilename_arg_type() (*in module pydrobert.kaldi.io.argparse*), 28
KaldiDataType (*class in pydrobert.kaldi.io.enums*), 37
KaldiInput (*class in pydrobert.kaldi.io.duck_streams*), 35
KaldiIOBase (*class in pydrobert.kaldi.io*), 24
KaldiLogger (*class in pydrobert.kaldi.logging*), 45
KaldiOutput (*class in pydrobert.kaldi.io.duck_streams*), 36
KaldiParser (*class in pydrobert.kaldi.io.argparse*), 25
KaldiRandomAccessReader (*class in pydrobert.kaldi.io.table_streams*), 40
KaldiSequentialReader (*class in pydrobert.kaldi.io.table_streams*), 40
KaldiTable (*class in pydrobert.kaldi.io.table_streams*), 41

KaldiVerbosityAction (*class in pydrobert.kaldi.io.argparse*), 27
KaldiWriter (*class in pydrobert.kaldi.io.table_streams*), 42
key() (*pydrobert.kaldi.io.table_streams.KaldiSequentialReader method*), 41
key_list (*pydrobert.kaldi.io.corpus.ShuffledData attribute*), 34
keys() (*pydrobert.kaldi.io.table_streams.KaldiSequentialReader method*), 41

L

logger (*pydrobert.kaldi.io.argparse.KaldiParser attribute*), 27
logging_lvl_to_kaldi_lvl() (*in module pydrobert.kaldi.logging*), 46

M

makeRecord() (*pydrobert.kaldi.logging.KaldiLogger method*), 45
module
    pydrobert.kaldi, 23
    pydrobert.kaldi.eval, 23
    pydrobert.kaldi.eval.util, 23
    pydrobert.kaldi.io, 24
    pydrobert.kaldi.io.argparse, 25
    pydrobert.kaldi.io.corpus, 28
    pydrobert.kaldi.io.duck_streams, 35
    pydrobert.kaldi.io.enums, 37
    pydrobert.kaldi.io.table_streams, 40
    pydrobert.kaldi.io.util, 43
    pydrobert.kaldi.logging, 45
move() (*pydrobert.kaldi.io.table_streams.KaldiSequentialReader method*), 41

N

NotATable (*pydrobert.kaldi.io.enums.TableType attribute*), 39
num_batches (*pydrobert.kaldi.io.corpus.Data property*), 30
num_samples (*pydrobert.kaldi.io.corpus.Data property*), 30
num_samples (*pydrobert.kaldi.io.corpus.SequentialData property*), 32
num_samples (*pydrobert.kaldi.io.corpus.ShuffledData property*), 34
num_sub (*pydrobert.kaldi.io.corpus.Data attribute*), 30
num_sub (*pydrobert.kaldi.io.corpus.SequentialData attribute*), 32
num_sub (*pydrobert.kaldi.io.corpus.ShuffledData attribute*), 34
numpy_dtype_arg_type() (*in module pydrobert.kaldi.io.argparse*), 28

O

OffsetFileInput (*pydrobert.kaldi.io.enums.RxfilenameType attribute*), 39
once (*pydrobert.kaldi.io.KaldiIOBase attribute*), 24
open() (*in module pydrobert.kaldi.io*), 25
open_duck_stream() (*in module pydrobert.kaldi.io.duck_streams*), 36
open_table_stream() (*in module pydrobert.kaldi.io.table_streams*), 42

P

parse_kaldi_config_file() (*in module pydrobert.kaldi.io.argparse*), 28
parse_kaldi_input_path() (*in module pydrobert.kaldi.io.util*), 44
parse_kaldi_output_path() (*in module pydrobert.kaldi.io.util*), 44
parse_known_args() (*pydrobert.kaldi.io.argparse.KaldiParser method*), 27
path (*pydrobert.kaldi.io.KaldiIOBase attribute*), 24
permissive (*pydrobert.kaldi.io.KaldiIOBase attribute*), 24
PipedInput (*pydrobert.kaldi.io.enums.RxfilenameType attribute*), 39
PipedOutput (*pydrobert.kaldi.io.enums.WxfilenameType attribute*), 40
print_help() (*pydrobert.kaldi.io.argparse.KaldiParser method*), 27
print_usage() (*pydrobert.kaldi.io.argparse.KaldiParser method*), 27
pydrobert.kaldi
    module, 23
pydrobert.kaldi.eval
    module, 23
pydrobert.kaldi.eval.util
    module, 23
pydrobert.kaldi.io
    module, 24
pydrobert.kaldi.io.argparse
    module, 25
pydrobert.kaldi.io.corpus
    module, 28
pydrobert.kaldi.io.duck_streams
    module, 35
pydrobert.kaldi.io.enums
    module, 37
pydrobert.kaldi.io.table_streams
    module, 40
pydrobert.kaldi.io.util
    module, 43
pydrobert.kaldi.logging
    module, 45

R

read() (*pydrobert.kaldi.io.duck_streams.KaldiInput method*), 35
readable() (*pydrobert.kaldi.io.duck_streams.KaldiInput method*), 36
readable() (*pydrobert.kaldi.io.duck_streams.KaldiOutput method*), 36
readable() (*pydrobert.kaldi.io.KaldiIOBase method*), 25
readable() (*pydrobert.kaldi.io.table_streams.KaldiRandomAccessReader method*), 40
readable() (*pydrobert.kaldi.io.table_streams.KaldiSequentialReader method*), 41
readable() (*pydrobert.kaldi.io.table_streams.KaldiWriter method*), 42
register_logger_for_kaldi() (*in module pydrobert.kaldi.logging*), 46
rng (*pydrobert.kaldi.io.corpus.ShuffledData attribute*), 34
RxfilenameType (*class in pydrobert.kaldi.io.enums*), 39

S

sample_generator() (*pydrobert.kaldi.io.corpus.Data method*), 30
sample_generator_for_epoch() (*pydrobert.kaldi.io.corpus.Data method*), 31
sample_generator_for_epoch() (*pydrobert.kaldi.io.corpus.SequentialData method*), 32
sample_generator_for_epoch() (*pydrobert.kaldi.io.corpus.ShuffledData method*), 34
ScriptTable (*pydrobert.kaldi.io.enums.TableType attribute*), 39
SequentialData (*class in pydrobert.kaldi.io.corpus*), 31
ShuffledData (*class in pydrobert.kaldi.io.corpus*), 32
sorted (*pydrobert.kaldi.io.KaldiIOBase attribute*), 25
StandardInput (*pydrobert.kaldi.io.enums.RxfilenameType attribute*), 39
StandardOutput (*pydrobert.kaldi.io.enums.WxfilenameType attribute*), 40

T

table_holders (*pydrobert.kaldi.io.corpus.ShuffledData attribute*), 34
table_specifiers (*pydrobert.kaldi.io.corpus.Data attribute*), 29
table_specifiers (*pydrobert.kaldi.io.corpus.SequentialData attribute*), 31
table_specifiers (*pydrobert.kaldi.io.corpus.ShuffledData attribute*), 33
table_type (*pydrobert.kaldi.io.KaldiIOBase attribute*), 24
TableType (*class in pydrobert.kaldi.io.enums*), 39
Token (*pydrobert.kaldi.io.enums.KaldiDataType attribute*), 38
TokenVector (*pydrobert.kaldi.io.enums.KaldiDataType attribute*), 38

U

utt2spk (*pydrobert.kaldi.io.table_streams.KaldiRandomAccessReader attribute*), 40

V

value() (*pydrobert.kaldi.io.table_streams.KaldiSequentialReader method*), 41
values() (*pydrobert.kaldi.io.table_streams.KaldiSequentialReader method*), 41
version (*pydrobert.kaldi.io.argparse.KaldiParser attribute*), 27

W

WaveMatrix (*pydrobert.kaldi.io.enums.KaldiDataType attribute*), 38
writable() (*pydrobert.kaldi.io.duck_streams.KaldiInput method*), 36
writable() (*pydrobert.kaldi.io.duck_streams.KaldiOutput method*), 36
writable() (*pydrobert.kaldi.io.KaldiIOBase method*), 25
writable() (*pydrobert.kaldi.io.table_streams.KaldiRandomAccessReader method*), 40
writable() (*pydrobert.kaldi.io.table_streams.KaldiSequentialReader method*), 41
writable() (*pydrobert.kaldi.io.table_streams.KaldiWriter method*), 42
write() (*pydrobert.kaldi.io.duck_streams.KaldiOutput method*), 36
write() (*pydrobert.kaldi.io.table_streams.KaldiWriter method*), 42
WxfilenameType (*class in pydrobert.kaldi.io.enums*), 39

X

xfilenames (*pydrobert.kaldi.io.KaldiIOBase attribute*), 24
xtypes (*pydrobert.kaldi.io.KaldiIOBase attribute*), 24