ABXpy Documentation

Release

Author

September 08, 2014

Contents

Contents:

CHAPTER 1

ABXpy Package

1.1 ABXpy Package

ABX discrimination is a term that is used for three stimuli presented on an ABX trial. The third is the focus. The first two stimuli (A and B) are standard, S1 and S2 in a randomly chosen order, and the subjects' task is to choose which of the two is matched by the final stimulus (X). (Glottopedia)

This package contains the operations necessary to initialize, calculate and analyse the results of an ABX discrimination task.

1.1.1 Organisation

It is composed of 3 main modules and other submodules.

- task module is used for creating a new task and preprocessing.
- distance package is used for calculating the distances necessary for the score calculation.
- score module is used for computing the score of a task.
- analyze module is used for analysing the results.

The features can be calculated in numpy via external tools, and made compatible with this package with the npz2h5features function

1.1.2 The pipeline

#TODO the table doesnt render well, do a graphic version in a line

In	Module	Out
• data.item • parameters	task	• data.abx
• data.abx • data.features • distance	distance	• data.distance
• data.abx • data.distance	score	• data.score
• data.abx • data.score	analyse	• data.csv

See Files Format for a description of the files used as input and output.

1.2 `analyze` Module

`ABXpy.analyze.`**`analyze`**(*scorefile*, *taskfile*, *outfile*)

Analyse the results of a task

Parameters **task_file** : string, hdf5 file

the file containing the triplets and pairs of the task

score_file : string, hdf5 file

the file containing the score of a task

analyse_file: string, csv file

the file that will contain the analysis

`ABXpy.analyze.`**`collapse`**(*scorefile*, *taskfile*, *fid*)

Collapses the results for each triplets sharing the same on, across and by labels.

`ABXpy.analyze.`**`npdecode`**(*keys*, *max_ind*)

Vectorized implementation of the decoding of the labels: i = (a1*n2 + a2)*n3 + a3 ...

`ABXpy.analyze.`**`parse_args`**()

`ABXpy.analyze.`**`unique_rows`**(*arr*)

Numpy unique applied to the row only.

1.3 `score` Module

This module is used for computing the score of a task (see task Module on how to create a task)

This module contains the actual computation of the score. It requires a task and a distance, and redirect the output in a score file.

The main function takes a distance file and a task file as input to compute the score of the task on those distances. X closer to A is associated with a score of 1 and X closer to B with score of -1.

The distances between pairs in the distance file must be ordered the same way as the pairs in the task file, and the triplet score int the output file will be ordered the same way as the triplets in the task file.

1.3.1 Usage

Form the command line:

```
python score.py data.abx data.distance data.score
```

In python:

```
import ABXpy.task
import ABXpy.score
# create a new task:
myTask = ABXpy.task.Task('data.item', 'on_feature', 'across_feature', 'by_feature', filters=my_filte
myTask.generate_triplets()
#initialise distance
#TODO shouldn't this be available from score
# calculate the scores:
ABXpy.score('data.abx', 'myDistance.???', 'data.score')
```

ABXpy.score.**score**(*task_file*, *distance_file*, *score_file=None*, *score_group='scores'*)

Calculate the score of a task and put the results in a hdf5 file.

Parameters **task_file** : string

The hdf5 file containing the task (with the triplets and pairs generated)

distance_file : string

The hdf5 file containing the distances between the pairs

score_file : string, optional

The hdf5 file that will contain the results

1.4 `task` Module

This module is used for creating a new task and preprocessing.

This module contains the functions to specify and initialise a new ABX task, compute and display the statistics, and generate the ABX triplets and pairs.

It can also be used in a command line. See task –help for the documentation

1.4.1 Usage

Form the command line:

```
python task.py my_data.item -o column1 -a column2 column3 -b column4 column5 -f "[attr == 0 for attr
```

my_data.item is a special file containing an index of the database and a set of labels or attributes. See input format [#TODO insert hypertext]

In python:

```
import ABXpy.task
# create a new task and compute the statistics
myTask = ABXpy.task.Task('data.item', 'on_label', 'across_feature', 'by_label', filters=my_filters, r
print myTask.stats  # display statistics
myTask.generate_triplets()  # generate a h5db file 'data.abx'containing all the triplets and pairs
```

1.4.2 Example

#TODO this example is for the front page or ABX module, to move An example of ABX triplet:

A	B	X
on_1	on_2	on_1
ac_1	ac_1	ac_2
by	by	by

A and X share the same 'on' attribute; A and B share the same 'across' attribute; A,B and X share the same 'by' attribute

class ABXpy.task.**Task**(*db_name*, *on*, *across=None*, *by=None*, *filters=None*, *regressors=None*, *verbose=0*, *verify=True*, *features=None*)

Bases: object

Define an ABX task for a given database.

Parameters **db_name** : str

the filename of database on which the ABX task is applied.

on : str

the 'on' attribute of the ABX task. A and X share the same 'on' attribute and B has a different one.

across : list, optional

a list of strings containing the 'across' attributes of the ABX task. A and B share the same 'across' attributes and X has a different one.

by : list, optional

a list of strings containing the 'by' attributes of the ABX task. A,B and X share the same 'by' attributes.

filters : list, optional

a list of string specifying a filter on A, B or X.

regressors : list, optional

a list of string specifying a filter on A, B or X.

verbose : int, optional

display additionnal information is set superior to 0.

verify : str, optionnal

verify the correctness of the database file, do by default.

features : str, otpionnal

the features file. Add it to verify the consistency with the item file

Attributes

stats •nb_blocks the number of blocks of ABX triplets sharing the same 'on', 'across' and 'by' features. •nb_triplets the number of triplets considered. •nb_by_levels the number of blocks of ABX triplets sharing the same 'by' attribute.	(dict. Contain several statistics about the task. The main 3 attributes are:)

compute_nb_levels()

compute_statistics(*approximate=False*)
Compute the statistics of the task

The number of ABX triplets is exact in most cases if approximate is set to false. The other statistics can only be approxrimate in the case where there are A, B, X or ABX filters.

Parameters **Approximate** : bool

approximate the number of triplets

generate_pairs(*output=None*)
Generate the pairs associated to the triplet list

Note: This function is called by generate_triplets and should not be used independantly

---

generate_triplets(*output=None*, *sample=None*)
Generate all possible triplets for the whole task and the associated pairs

Generate the triplets and the pairs for an ABXpy.Task and store it in a h5db file.

Parameters **output** : filename, optional

The output file. If not specified, it will automatically create a new file with the same name as the input file.

sample : bool, optional

apply the function on a sample of the task

on_across_triplets(*by*, *on*, *across*, *on_across_block*, *on_across_by_values*, *with_regressors=True*)
Generate all possible triplets for a given by block.

Given an on_across_block of the database and the parameters of the task, this function will generate the complete set of triplets and the regressors.

Parameters **by** : int

The block index

on, across : int

The task attributes

on_across_block : list

the block

on_across_by_values : dict

the actual values

with_regressors : bool, optional

By default, true

Returns **triplets** : numpy.Array

the set of triplets generated

regressors : numpy.Array

the regressors generated

print_stats(*filename=None*, *summarized=True*)

print_stats_to_stream(*stream*, *summarized*)

ABXpy.task.**on_across_from_key**(*key*)

ABXpy.task.**verifydb**(*filename*, *features=None*)

1.5 Subpackages

1.5.1 database Package

database Package

Created on Sun Aug 18 22:31:59 2013

@author: thomas

This file only serves to signal that the content of the folder is a Python package.

database Module

ABXpy.database.database.**load**(*filename*, *features_info=False*)

ABXpy.database.database.**load_aux_dbs**(*basename*, *db*, *cols*, *mainfile*)

ABXpy.database.database.**read_table**(*filename*)

1.5.2 dbfun Package

dbfun Package

Created on Sun Aug 18 22:31:59 2013

@author: thomas

This file only serves to signal that the content of the folder is a Python package.

dbfun Module

Created on Mon Oct 14 16:59:27 2013

@author: Thomas Schatz

class ABXpy.dbfun.dbfun.**DBfun**(*input_names*)
Bases: object

evaluate(*inputs_dict*)

output_specs()

dbfun_column Module

Created on Fri Dec 20 13:36:52 2013

@author: Thomas Schatz

class ABXpy.dbfun.dbfun_column.**DBfun_Column**(*name*, *db=None*, *column=None*, *indexed=True*)
Bases: ABXpy.dbfun.dbfun.DBfun

evaluate(*context*)

output_specs()

dbfun_compute Module

Created on Mon Oct 14 16:59:27 2013

@author: Thomas Schatz

Class for defining and computing efficiently functions of the columns of a database. Implements the DBfun API

class ABXpy.dbfun.dbfun_compute.**DBfun_Compute**(*definition*, *columns*)
Bases: ABXpy.dbfun.dbfun.DBfun

evaluate(*context*)

output_specs()

parse()
first separate the script defining the function into various components (import statements, with 'h5file' statement, synopsis definition, main code)

process_with(*tree*)

class ABXpy.dbfun.dbfun_compute.**nameVisitor**(**args*, ***kwargs*)
Bases: ast.NodeVisitor

visit_Name(*node*)

dbfun_lookuptable Module

Created on Mon Nov 25 00:24:42 2013

@author: Thomas Schatz

Implements the DBfun API Basic implementation of database function in lookup tables. Allows on-the-fly computation by storing script for DBfun_compute alongside the table. Allows to store outputs with h5 compatible dtypes

either directly or under an indexed format Do not implement variable length outputs and requires that the entire lookup table fits in RAM memory.

class ABXpy.dbfun.dbfun_lookuptable.**DBfun_LookupTable**(*filename*, *synopsis=None*, *db=None*, *code=None*, *indexed=True*)

Bases: ABXpy.dbfun.dbfun.DBfun

compress_index(*indexed_output_name*)

evaluate(*context*)

fill(*data*, *append=False*, *iterate=False*)

get_keys(*input_values*)

initialize_output_dsets(*sample_data*)

load()

load_data()

load_metadata()

output_specs()

pack()

write(*data*)

ABXpy.dbfun.dbfun_lookuptable.**chunk_size**(*item_size=4*, *n_columns=1*, *size_in_mem=400*)

ABXpy.dbfun.dbfun_lookuptable.**get_dtype**(*data*)

lookuptable_connector Module

Created on Tue Nov 12 02:25:17 2013

@author: Thomas Schatz

class ABXpy.dbfun.lookuptable_connector.**LookupTableConnector**(*script*, *aux_functions*, *aliases*, **args*, ***kwargs*)

Bases: ast.NodeTransformer

bundle_call_info(*node*, *call_node*)

check_Call(*node*)

check_Subscript(*node*, *func_name*)

check_flatness(*node*)

generic_visit(*node*)

visit_Call(*node*)

visit_Subscript(*node*)

1.5.3 distances Package

distances Package

Created on Sun Aug 18 22:31:59 2013

@author: thomas

This file only serves to signal that the content of the folder is a Python package.

distances Module

Created on Thu May 8 04:07:43 2014

@author: Thomas Schatz

class ABXpy.distances.distances.**Features_Accessor**(*times*, *features*)
Bases: object

get_features_from_raw(*items*)

get_features_from_splitted(*items*)

ABXpy.distances.distances.**compute_distances**(*feature_file*, *feature_group*, *pair_file*, *distance_file*, *distance*, *n_cpu=None*, *mem=1000*, *feature_file_as_list=False*)

ABXpy.distances.distances.**create_distance_jobs**(*pair_file*, *distance_file*, *n_cpu*)

class ABXpy.distances.distances.**print_exception**(*fun*)
Bases: object

ABXpy.distances.distances.**run_distance_job**(*job_description*, *distance_file*, *distance*, *feature_files*, *feature_groups*, *splitted_features*, *job_id*, *distance_file_lock=None*)

Subpackages

metrics Package

metrics Package Created on Sun Aug 18 22:31:59 2013

@author: thomas

This file only serves to signal that the content of the folder is a Python package.

cosine Module Created on Wed Jan 22 01:47:42 2014

@author: Thomas Schatz

ABXpy.distances.metrics.cosine.**cosine_distance**(*x*, *y*)

ABXpy.distances.metrics.cosine.**normalize_cosine_distance**(*x*, *y*)

1.5.4 h5tools Package

h5tools Package

Created on Sun Aug 18 22:31:59 2013

@author: thomas

This file only serves to signal that the content of the folder is a Python package.

h52np Module

Created on Fri Oct 25 16:30:23 2013

@author: Thomas Schatz

class ABXpy.h5tools.h52np.**H52NP**(*h5file*)
Bases: object

add_dataset(*group*, *dataset*, *buf_size=100*, *minimum_occupied_portion=0.25*)

class ABXpy.h5tools.h52np.**H52NPbuffer**(*parent*, *group*, *dataset*, *buf_size*, *minimum_occupied_portion*)
Bases: object

buffer_empty()

current_tail()

dataset_empty()

isempty()

nb_lower_than(*x*)

read(*amount=None*)

refill_buffer()

h5_handler Module

Created on Tue Oct 15 09:48:31 2013

@author: Thomas Schatz

class ABXpy.h5tools.h5_handler.**H5Handler**(*h5file*, *keygroup*, *keyset*, *groups=None*, *datasets=None*)
Bases: object

extract_chunk(*i_start*, *i_end*, *chunk_id*)

sort(*buffer_size=1000*, *o_buffer_size=1000*)

class ABXpy.h5tools.h5_handler.**H5TMP**
Bases: object

ABXpy.h5tools.h5_handler.**test**()

h5io Module

Created on Sun Jan 19 17:06:15 2014

@author: Thomas Schatz

class ABXpy.h5tools.h5io.**H5IO**(*filename*, *datasets=None*, *indexes=None*, *fused=None*, *group='/'*)
Bases: object

find()

read()

sort()

write(*data*, *append=True*, *iterate=False*, *indexed=False*)

ABXpy.h5tools.h5io.**get_dtype**(*data*)

ABXpy.h5tools.h5io.**test_h5io**()

np2h5 Module

Created on Thu Sep 19 13:46:18 2013

@author: Thomas Schatz

Class for efficiently writing to disk (in a specified dataset of a HDF5 file) simple two-dimensional numpy arrays that are incrementally generated along the first dimension. It uses buffers to avoid small I/O.

It needs to be used within a 'with' statement, so as to handle buffer flushing and opening and closing of the underlying HDF5 file smoothly.

Buffer size should be chosen according to speed/memory trade-off. Due to cache issues there is probably an optimal size.

The size of the dataset to be written must be known in advance, excepted when overwriting an existing dataset. Not writing exactly the expected amount of data causes an Exception to be thrown excepted is the fixed_size option was set to False when adding the dataset.

class ABXpy.h5tools.np2h5.**NP2H5**(*h5file*)

Bases: object

add_dataset(*group*, *dataset*, *n_rows=0*, *n_columns=None*, *chunk_size=100*, *buf_size=100*, *item_type=<Mock id='139702446326992'>*, *overwrite=False*, *fixed_size=True*)

class ABXpy.h5tools.np2h5.**NP2H5buffer**(*parent*, *group*, *dataset*, *n_rows*, *n_columns*, *chunk_size*, *buf_size*, *item_type*, *overwrite*, *fixed_size*)

Bases: object

delete()

flush()

iscomplete()

write(*data*)

ABXpy.h5tools.np2h5.**nb_lines**(*item_size*, *n_columns*, *size_in_mem*)

1.5.5 misc Package

misc Package

Created on Sun Aug 18 22:31:59 2013

author thomas

This module contains several useful functions and classes that dont fit in any other modules.

progress_display Module

Created on Tue Jan 7 18:11:16 2014

author Thomas Schatz

This class is used to display the progress during the computing.

class ABXpy.misc.progress_display.**ProgressDisplay**
Bases: object

add(*name*, *message*, *total*)

display()

update(*name*, *amount*)

ABXpy.misc.progress_display.**testProgressDisplay**()

tinytree Module

class ABXpy.misc.tinytree.**Tree**(*children=None*)
Bases: object

A simple implementation of an ordered tree

addChild(*node*)
Add a child to this node.

:child A Tree object

addChildrenFromList(*children*)
Add children to this node.

:children A nested list specifying a tree of children

attrsToRoot(*attr*)
Traverses the path from this node to the root of the tree, and yields a value for each attribute. Nodes that do not have the attribute and attribute values that test false are ignored.

:attr A string attribute name

clear()
Clear all the children of this node. Return a list of the removed children.

count()
Number of nodes in this tree, including the root.

dump(*outf=<open file '<stdout>', mode 'w' at 0x7f785cc1a150>*)
Dump a formatted representation of this tree to the specified file descriptor.

:outf Output file descriptor.

findAttr(*attr*, *default=None*)
Traverses the path to the root of the tree, looking for the specified attribute. If it is found, return it, else return default.

:attr A string attribute name :default Arbitrary default return value

findBackwards(**func*, ***kwargs*)
Search backwards in a preOrder traversal of the whole tree (not this node's subnodes). Return None if object not found.

:func A list of selector functions, that accept a node, and return a boolean.

:kwargs A dictionary of attribute selectors. Checks that matching attributes exist, and that their values are equal to the specified values.

findChild(**func*, ***kwargs*)
Find the first child matching all specified selectors in a pre-order traversal of this node's subnodes. Return None if no matching object is found.

:func A list of selector functions, that accept a node, and return a boolean.

:kwargs A dictionary of attribute selectors. Checks that matching attributes exist, and that their values are equal to the specified values.

findForwards(**func*, ***kwargs*)
Search forwards in a preOrder traversal of the whole tree (not this node's subnodes). Return None if object not found.

:func A list of selector functions, that accept a node, and return a boolean.

:kwargs A dictionary of attribute selectors. Checks that matching attributes exist, and that their values are equal to the specified values.

findParent(**func*, ***kwargs*)
Find the first node matching func in a traversal to the root of the tree. Return None if no matching object is found.

:func A list of selector functions, that accept a node, and return a boolean.

:kwargs A dictionary of attribute selectors. Checks that matching attributes exist, and that their values are equal to the specified values.

getDepth()
Return the depth of this node, i.e. the number of nodes on the path to the root.

getNext()
Find the next node in the preOrder traversal of the tree.

getPrevious()
Find the previous node in the preOrder traversal of the tree.

getRoot()
Return the topmost node in the tree.

index()
Return the index of this node in the parent child list, based on object identity.

inject(*node*)
Inserts a node between the current node and its children. Returns the specified parent node.

:node A Tree object

isDescendantOf(*node*)
Returns true if this node lies on the path to the root from the specified node.

:node A Tree object

isSiblingOf(*node*)
Returns true if this node is a sibling of the specified node.

:node A Tree object

pathFromRoot()
Generator yielding all nodes on the path to this node from the root of the tree, including this node itself.

pathToRoot()
Generator yielding all objects on the path from this node to the root of the tree, including this node itself.

postOrder()
Return a list of the subnodes in PostOrder.

preOrder()
Return a list of subnodes in PreOrder.

register(*parent*)
Called after a node has been added to a parent.

:child A Tree object

remove()
Remove this node from its parent. Returns the index this node had in the parent child list.

reparent(*node*)
Inserts a node between the current node and its parent. Returns the specified parent node.

:node A Tree object

replace(**nodes*)
Replace this node with a sequence of other nodes. This is equivalent to deleting this node from the child list, and then inserting the specified sequence in its place.

:nodes A sequence of Tree objects

siblings()
Generator yielding all siblings of this node, including this node itself.

static **treeProp**(*name*)
Define a property whose value should be looked up on nodes between this node and the root, inclusive. Returns the first matching attribute. Raises ValueError if no matching attribute is found.

:name Property name

ABXpy.misc.tinytree.**constructFromList**(*lst*)
:lst a nested list of Tree objects

Returns a list consisting of the nodes at the base of each tree. Trees are constructed "bottom-up", so all parent nodes for a particular node are guaranteed to exist when "addChild" is run.

type_fitting Module

Created on Tue Oct 29 00:29:00 2013

author Thomas Schatz

ABXpy.misc.type_fitting.**fit_integer_type**(*n*, *is_signed=True*)
Determine the minimal space needed to store integers of maximal value n

1.5.6 sampling Package

sampling Package

This module implement an incremental sampler used to approximate the task and randomly select a portion of the triplets.

sampler Module

The sampler class implementing incremental sampling without replacement. Incremental meaning that you don't have to draw the whole sample at once, instead at any given time you can get a piece of the sample of a size you specify. This is useful for very large sample sizes.

class ABXpy.sampling.sampler.**IncrementalSampler**(*N*, *K*, *step=None*, *relative_indexing=True*, *dtype=<type 'numpy.int64'>*)
Bases: object

next()

sample(*n*, *dtype=<type 'numpy.int64'>*)
Fast implementation of the sampling function

Get all samples from the next n items in a way that avoid rejection sampling with too large samples, more precisely samples whose expected number of sampled items is larger than 10**5.

Parameters **n** : int

the size of the chunk

Returns

——-

sample : numpy.array

the indices to keep given relative to the current position in the sample or absolutely, depending on the value of relative_indexing specified when initialising the sampler (default value is True)

simple_sample(*n*)
get all samples from the next n items in a naive fashion

Parameters **n** : int

the size of the chunk

Returns

——-

sample : numpy.array

the indices to be kept relative to the current position in the sample

ABXpy.sampling.sampler.**Knuth_sampling**(*n*, *N*, *dtype=<type 'numpy.int64'>*)
This is the usual sampling function when n is comparable to N

ABXpy.sampling.sampler.**hypergeometric_sample**(*N*, *K*, *n*)
This function return the number of elements to sample from the next n items.

ABXpy.sampling.sampler.**rejection_sampling**(*n*, *N*, *dtype=<type 'numpy.int64'>*)
Using rejection sampling to keep a good performance if n << N

ABXpy.sampling.sampler.**sample_without_replacement**(*n*, *N*, *dtype=<type 'numpy.int64'>*)
Returns uniform samples in [0, N-1] without replacement. It will use Knuth sampling or rejection sampling depending on the parameters n and N.

Note: the values 0.6 and 100 are based on empirical tests of the functions and would need to be changed if the functions are changed

---

1.5.7 sideop Package

sideop Package

Created on Sun Aug 18 22:31:59 2013

author thomas

This module contains the filter and regressor managers used by task to apply the filters and regressors. Both those classes use a side operation manager that implement the generic functions. This allow to apply the filters and regressors as early as possible during the triplet generation to optimise the performances.

filter_manager Module

Created on Mon Dec 16 05:00:10 2013

@author: Thomas Schatz

class ABXpy.sideop.filter_manager.**FilterManager**(*db_hierarchy*, *on*, *across*, *by*, *filters*)

Bases: ABXpy.sideop.side_operations_manager.SideOperationsManager

Manage the filters on attributes (on, across, by) or elements (A, B, X) for further processing

ABX_filter(*on_across_by_values*, *db*, *triplets*)

A_filter(*on_across_by_values*, *db*, *indices*)

B_filter(*on_across_by_values*, *db*, *indices*)

X_filter(*on_across_by_values*, *db*, *indices*)

by_filter(*by_values*)

classify_generic(*elements*, *db_fun*, *db_variables*)

generic_filter(*by_values*, *db*)

on_across_by_filter(*on_across_by_values*)

ABXpy.sideop.filter_manager.**singleton_filter**(*generator*)

ABXpy.sideop.filter_manager.**vectorial_filter**(*generator*, *indices*)

Note: To allow a lazy evaluation of the filter, the context is filtered explicitly which acts on the generator by a side-effect (dict being mutable in python)

---

regressor_manager Module

Created on Mon Dec 16 05:01:53 2013

author Thomas Schatz

class ABXpy.sideop.regressor_manager.**RegressorManager**(*db*, *db_hierarchy*, *on*, *across*, *by*, *regressors*)

Bases: ABXpy.sideop.side_operations_manager.SideOperationsManager

Manage the regressors on attributes (on, across, by) or elements (A, B, X) for further processing

classify_generic(*elements*, *db_fun*, *db_variables*)

fetch_regressor_info(*field*, *reg_id*)

get_regressor_info()

set_ABX_regressors(*on_across_by_values*, *db*, *triplets*)

set_A_regressors(*on_across_by_values*, *db*, *indices*)

set_B_regressors(*on_across_by_values*, *db*, *indices*)

set_X_regressors(*on_across_by_values*, *db*, *indices*)

set_by_regressors(*by_values*)

set_on_across_by_regressors(*on_across_by_values*)

side_operations_manager Module

Created on Tue Nov 12 06:39:40 2013

author Thomas Schatz

Class working closely with task.py providing services for it, specifically by:

- finding out the best point to execute side-operations (such as filtering and regressor generation) in the ABX task computation flow:
  - basically the more related a given side-operation is to the on/across/by structure of the ABX task, the earlier it can be executed and the lowest the computational cost is
- providing methods to actually carry out these side-operations at the point in the execution flow to which they were attributed

class ABXpy.sideop.side_operations_manager.**SideOperationsManager**(*db_hierarchy*, *on*, *across*, *by*)

Bases: object

add(*db_fun*, *name=None*)

check_extensions(*elements*)
Check that something with a AX, AB or 1, 2 extension is an on/across descendant and a correct one for AX, AB.

classify_ABX(*elements*, *db_fun*, *db_variables*)
the only left extensions are either not descendant of on/across/by or descendant of across and _X or descendant of on and _B (i.e. _2) we do not try to batch the _2 because we think they are potentially too small, instead if necessary we should batch several consecutive calls

classify_by(*elements*, *db_fun*, *db_variables*)
Detect operations that depend only on a variable that is used as a 'by' factor in the ABX task.

classify_generic(*elements*, *db_fun*, *db_variables*)
Detect operations that can be applied directly to the columns of the original database. This is subclass specific...

classify_on_across_by(*elements*, *db_fun*, *db_variables*)
Detect operations that can be applied at the level of an on/across/by block during the generation of the ABX triplets.

evaluate_A(**args*)

evaluate_A_B_X(*name*, *on_across_by_values*, *db*, *indices*, *context=None*)

evaluate_B(**args*)

evaluate_X(**args*)

evaluate_by(*by_values*)

evaluate_generic(*by_values*, *db*, *context=None*)

evaluate_on_across_by(*on_across_by_values*)

parse_extended_column(*column*)
Get radical and suffix part of a context_variable.

parse_extended_columns(*columns*)
Get radical and suffix part for every context_variable, returns the set of the encountered couples.

set_ABX_context(*context*, *db*, *triplets*)

set_A_B_X_context(*context_field*, *context*, *stage*, *db*, *indices*)

set_by_context(*context*, *stage*, *by_values*)

set_generic_context(*context*, *stage*, *db*)

set_on_across_context(*context*, *stage*, *on_across_values*)

ABXpy.sideop.side_operations_manager.**result_generator**(*db_funs*, *context*)

ABXpy.sideop.side_operations_manager.**singleton_result_generator**(*db_funs*, *context*)

CHAPTER 2

Files format

This package uses several types of files, this section describe them all.

2.1 *Dataset*

Extension: .item

This file indexes the database on which the ABX task is executed. It is a regular text file and should have the following structure:

#source	onset	offset	#label 1	label 2	label 3
file 1	start 1	stop 1	value 1	value 1	value 1
file 2	start 2	stop 2	value 2	value 1	value 1
file 3	start 3	stop 3	value 3	value 1	value 1

- **#source** is the name of the file minus the extension. Note that the '#' at the begining is mandatory.
- **onset** is the instant when the sound start.
- **offset** is the instant when the sound end.
- the **label** columns are various regressors relevant to the discrimination task. Note that the first column must start with a **'#'**.

2.2 *Features file*

Extension: .features

This file contains the features and the center time of each window in the h5features format. This is a special hdf5 file with the following attributes:

- **features** a 2D arrays with the 'feature' dimension along the columns and the 'time' dimension along the lines.
- **times** a 1D array with the center time of each window.
- **files** the basename of the files from which the features are extracted. Note that it does not contain the full absolute path nor the relative path of the files, each file must have a unique name.

2.3 *Task file*

Extension: .abx

This file can be generated by the task module. It is a hdf5 file. It contains all the triplets and the resulting pairs. The elements are grouped by their 'by' attribute (all the elements with the same by attributes belong to the same block)

The structure is as follow:

data.abx

- **triplets**
  - by0: (3 x ?)-array referencing all the possible triplets sharing a 'by' value of by0
  - by1
  - etc.
- **unique_pairs (All the pairs AX and BX, useful to calculate the distances. Note that a pair is designated by a single numbe**
  - by0: 1D-array referencing all the pairs sharing a 'by' value of by0. Note that this is only 1D instead of 2D due to a special encoding of the pairs. Let 'n' be the number of items in the block, 'a' be the index of the first item of the pair and 'b' the index of the second item: the index of the pair 'p' = n*a + b
  - etc.
- regressors (infos of the item file in a computer efficient format)
- feat_dbs (infos of the item file in a computer efficient format)

2.4 *Distance file*

Extension: .distance

This file contains the distances between the two members of each unique pair. The distances are store by 'by' block and in the same order as the unique_pairs in the Task file.

- **distances**
  - by0: 1D-array containing the distances between the two members of each pair.
  - by1
  - etc.

2.5 *Score file*

Extension: .score

This file contains the score of each triplets. The score is 1 when X is closer to A and -1 when X is closer to B. The score are stored by 'by' block and in the same order as the triplets in the Task file.

- **scores**
  - by0: 1D-array of integers containing the score of each triplet.
  - by1

  - etc.

2.6 *Analyse file*

Extension: .csv

The output file of the ABX baseline, in a human readable format. Contains the average results collapsed over triplets sharing the same on, across and by attributes. It uses a score of 1 when X is closer to A and 0 when X is closer to B.

The extensions _1 and _2 to the labels name follow the following convention:

A	B	X
on_1 ac_1	on_2 ac_1	on_1 ac_2

Example: For a task on 'on', across 'ac' and by 'by'

on_1	ac_1	ac_2	on_2	by	score	n
v0 v1	v0 v1	v1 v0	v1 v0	v0 v0	0.2 0.7	5 3

- **on_1** value of 'on' label for A and X
- **on_2** value of 'on' label for B
- **ac_1** value of 'ac' label for A and B
- **ac_2** value of 'ac' label for X
- **by** value of 'by' label for A, B and X
- **score** average score for those triplets
- **n** number of triplets

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*

a