eWRT - Extensible Web Retrieval Toolkit Documentation

Release 0.1

Mister X

Jul 11, 2017

Contents

Knowledge capture in the age of massive Web data requires robust and scalable mechanisms to acquire, consolidate and pre-process large amounts of heterogeneous data. The Extensible Web Retrieval Toolkit (eWRT) is modular open-source Python API that addresses this requirement. It retrieves social data from Web sources such as Delicious, Flickr, Yahoo! and Wikipedia, including various helper classes for effective caching and data management. The toolkit provides components for content acquisition and caching, low-level natural language processing functionalities such as language detection, phonetic string similarity measures, and methods for string normalization.

eWRT has been jointly developed by researchers from MODUL University Vienna, webLyzard technology, the University of Applied Sciences Chur, and the Vienna University of Economics and Business. The library is currently being extended as part of the uComp Project, which investigates Embedded Human Computation for Knowledge Extraction and Evaluation.

Checkout: eWRT on gitweb.

Contents:

CHAPTER 1

Installation

todo: Finish this part

Access to the Repository

Clone the eWRT git repository::

```
git clone http://git.semanticlab.net/eWRT.git
```

Or directly install using pip:

```
pip install git+http://git.semanticlab.net/eWRT.git
```

Dependencies

- python 2.6 or higher
- python-nose >=0.11

CHAPTER 2

eWRT Package

eWRT Package

@package eWRT

The easy Web Retrieval Toolkit

Subpackages

access Package

db Module

file Module

Convenient methods for file access

@package eWRT.access.file Created on Dec 6, 2012

@author: albert

class eWRT.access.file.**CompressedFile**(*fname*, *mode='rb'*)

Bases: object

An intelligent file object that transparently opens compressed files.

COMPRESSION_EXT = ('bz2', 'gz')

classmethod get_extension_list(*fname*)

Parameters **fname** – the file name to analyze

Return type a list of file extensions of this file ignoring extensions indicating file compression.

e.g. 'x/y/test.awp.csv.bz2' -> ['csv', 'awp']

classmethod open (*fname*, *mode='rb'*)

class eWRT.access.file.**TestFile**
Bases: object

TESTSTRING = 'this is a test'

test_formats()

http Module

config Package

config Package

@package eWRT.config evaluates ~/.eWRT/siteconfig.py and publishes the values

input Package

input Package

@package eWRT.input

Modules supporting data input, conversion and cleanup.

Subpackages

clean Package

clean Package

@package eWRT.input.clean

data cleanup modules.

text Module

conv Package

conv Package

@package eWRT.input.conv

Classes for converting various input formats into each other.

doc Module

@package eWRT.input.conv.doc converts Microsoft Word documents into text

class eWRT.input.conv.doc.**HtmlToText**
Bases: object

converts HTML into text requires a converter

static getText(*word_document_content*)
@param[in] word_document_content the content of the html page to convert @param[in] encoding the document encoding @returns the text representation of the Web page

html Module

@package eWRT.input.conv.html converts HTML pages into text

class eWRT.input.conv.html.**HtmlToText**
Bases: object

converts HTML into text requires a converter

static getText(*html_content*, *encoding='utf8'*)
@param[in] html_content the content of the html page to convert @param[in] encoding the document encoding @returns the text representation of the Web page

class eWRT.input.conv.html.**TestHtmlToText**
Bases: object

testBorderCases()

testConversion()

pdf Module

@package eWRT.input.conv.pdf converts PDF documents into text

class eWRT.input.conv.pdf.**HtmlToText**
Bases: object

converts HTML into text requires a converter

static getText(*pdf_content*)
@param[in] pdf_content the content of the html page to convert @param[in] encoding the document encoding @returns the text representation of the Web page

Subpackages

cxl Package

cxl Package

corpus Package

Subpackages

bbc Package

bbc Package

@package DetectLanguage.reuters a generic method to retrieve text from the reuters corpus

class eWRT.input.corpus.bbc.**BBCGetCorpus**(*filePattern='*'*)
Bases: object

An iterator over all documents

static getTitle(*text*)
returns the title of a given text

next()

reuters Package

reuters Module

stock Package

stock Package

Subpackages

lib Package

lib Package

Result Module

class eWRT.lib.Result.**Result**(*id*, *name*)

getAttributes()
todo: return attributes of Result

getId()

getName()

ResultSet Module

class eWRT.lib.ResultSet.**ResultSet**(*id=None*, *name=None*, *content=None*)

addContent(*newContent*)

getId()

getName()

next()

static printRS(*resultSet*, *filler=0*)

refresh()

setContent(*content*)

Webservice Module

class eWRT.lib.Webservice.**Webservice**
Bases: object

login()
implement me

apihelber Module

eWRT.lib.apihelber.**info**(*object*, *spacing=10*, *collapse=1*)
Print methods and doc strings. Takes module, class, list, dictionary, or string.

Subpackages

thirdparty Package

Subpackages

advas Package

advas Package

source code from the AdvaS Advanced Search project version: 0.2.3

phonetics Module

eWRT.lib.thirdparty.advas.phonetics.**caverphone**(*term*)
returns the language key using the caverphone algorithm 2.0

eWRT.lib.thirdparty.advas.phonetics.**metaphone**(*term*)
returns metaphone code for a given string

eWRT.lib.thirdparty.advas.phonetics.**nysiis**(*term*)
returns New York State Identification and Intelligence Algorithm (NYSIIS) code for the given term

eWRT.lib.thirdparty.advas.phonetics.**soundex**(*term*)
Return the soundex value to a string argument.

ontology Package

ontology Package

@package eWRT.ontology.eval Evaluates ontologies based on their _internal_ characteristics such as term coherence, internal integrity, ...

For evaluations against a reference ontology @see eWRT.ontology.compare

Subpackages

compare Package

Subpackages

relations Package

`relations` Package

relationtypes Package

`relationtypes` Package

eval Package

Subpackages

terminology Package

`terminology` Package

visualize Package

`visualize` Package

output Package

Subpackages

plot Package

`plot` Package

`radar_chart` Module

stat Package

`stat` Package

@package eWRT.stat

Subpackages

coherence Package

coherence Package

@package eWRT.ws.stat.coherence Determines how strongly two terms are connected to each other

class eWRT.stat.coherence.**Coherence**(*dataSource*, *cache=True*)
Bases: object

@class Coherence abstract class for computing the coherence between terms

static getCoherence(*nx*, *ny*, *nt*)
@param[in] nx counts of term1 @param[in] ny counts of term2 @param[in] nt counts of term1 together with term2 @returns the coherence

getTermCoherence(*t1*, *t2*)
@param[in] t1 term1 @param[in] t2 term2 @returns the coherence between these two terms

class eWRT.stat.coherence.**DiceCoherence**(*dataSource*, *cache=True*)
Bases: *eWRT.stat.coherence.Coherence*

@class DiceCoherence computes the dice coherence for the given terms

static getCoherence(*nx*, *ny*, *nt*)
@param[in] nx counts of term1 @param[in] ny counts of term2 @param[in] nt counts of term1 together with term2 @returns the coherence

class eWRT.stat.coherence.**PMICoherence**(*dataSource*, *cache=True*)
Bases: *eWRT.stat.coherence.Coherence*

@class PMICoherence computes the coherence based on the pointwise mutual information (PMI)

static getCoherence(*nx*, *ny*, *nt*)
@param[in] nx counts of term1 @param[in] ny counts of term2 @param[in] nt counts of term1 together with term2 @returns the coherence

class eWRT.stat.coherence.**TestCoherence**(*methodName='runTest'*)
Bases: unittest.case.TestCase

testDice()
tests the computation of the dice coefficient based on the example in

http://en.wikipedia.org/wiki/Dice's_coefficient

testPMI()
tests the computation of the PMI based on the results from wilson's paper

testPMIZero()
tests the handling of PMI values of no counts are found

eval Package

metrics Module

@package eWRT.ws.stat.eval.metrics Standard IR evaluation metrics such as

- precision
- recall
- F1 measure

class eWRT.stat.eval.metrics.**TestEvaluationMetrics**
Bases: object

tests the evaluation metrics

a = set([8, 1, 3, 9])

b = set([1, 10, 3, 12])

c = set([1, 3])

d = set([10, 11])

testFMeasure()

testPrecision()

testRecall()

eWRT.stat.eval.metrics.**fMeasure**(*p*, *r*, *beta=1.0*)
returns the F-measure for the given precision and recall @param[in] p precision @param[in] r recall @param[in] beta weight used to compute the f mesure @returns the F-Measure

eWRT.stat.eval.metrics.**precision**(*relevant*, *retrieved*)
returns the precision of the given sets @param[in] relevant set of relevant terms @param[in] retrieved set of retrieved terms @returns the precision

eWRT.stat.eval.metrics.**recall**(*relevant*, *retrieved*)
returns the recall of the given sets @param[in] relevant set of relevant terms @param[in] retrieved set of retrieved terms @returns the recall

language Package

language Package

@package eWRT.stat.language

language detection

class eWRT.stat.language.**DetectLanguageTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_detect_language()
tests the language detection based on examples

test_exceptions()
results for empyt strings

eWRT.stat.language.**detect_language**(*text*)
detects the most probable language for the given text

eWRT.stat.language.**get_lang_name**(*fname*)

eWRT.stat.language.**read_wordlist**(*fname*)
reads a language wordlist from a file

string Package

string Package

spelling Module

util Package

advLogging Module

class eWRT.util.advLogging.**SNMPHandler**(*moduleName*)
Bases: logging.Handler

Logging handler for sending SNMP traps

emit(*record*)
sends the message

class eWRT.util.advLogging.**TestHandler**(*methodName='runTest'*)
Bases: unittest.case.TestCase

testHandler()
tests the handler

eWRT.util.advLogging.**sendSNMPTrap**(*message*, *module*, *level*)
sends a SNMP message @param message: String that should be sent @param level: SNMP levels ('ok', 'warning', 'critical', 'unknown')

assert Module

@package eWRT.util.assert Assertion based counters

Examples: see unittests

class eWRT.util.assert.**AssertReturnValue**(*evalExpression*, *counterNameTrue*, *counterNameFalse*)
Bases: object

decorator class used to time functions

class eWRT.util.assert.**TestAssertReturnValue**
Bases: object

testAssertCounter()
verifies the assertion counters

async Module

@package eWRT.util.async asynchronous procedure calls

@warning this library is still a draft and might change considerable

class eWRT.util.async.**Async**(*cache_dir*, *cache_nesting_level=0*, *cache_file_suffix=''*, *max_processes=8*, *debug_dir=None*)
Bases: *eWRT.util.cache.DiskCache*

Asynchronous Call Handling

fetch(*cache_file*)

getPostHashfile(*cmd*)
returns an identifier representing the object which is compatible to the identifiers returned by the eWRT.util.cache.* classes.

has_processes_limit_reached()
closes finished processes and verifies whether we have already reached the maximum number of processes

post(*cmd*)
checks whether the given command is already cached and calls the command otherwise. @param[in] cmdline command to call @returns the hash required to fetch this object

class eWRT.util.async.**TestAsync**
Bases: object

unittests covering the class async

TEST_CACHE_DIR = './.test-async'

setUp()

tearDown()

testDebugMode()
tests the debug mode

testMaxProcessLimit()
tests the max process limit

cache Module

@package eWRT.util.cache caches arbitrary objects

class eWRT.util.cache.**Cache**(*fn=None*)
Bases: object

An abstract class for caching functions

fetch(*fetch_function*, **args*, ***kargs*)
Fetches a object from the cache or computes it by calling the fetch_function. The objectId is computed based on the function arguments

fetchObjectId(*key*, *fetch_function*, **args*, ***kargs*)
Fetches a object from the cache or computes it by calling the fetch_function. The key helps to determine whether the object is already in the cache or not.

static getKey(**args*, ***kargs*)
returns the key for a set of function parameters

static getObjectId(*obj*)
returns an identifier representing the object

class eWRT.util.cache.**DiskCache**(*cache_dir*, *cache_nesting_level=0*, *cache_file_suffix=''*, *fn=None*)
Bases: *eWRT.util.cache.Cache*

@class DiskCache Caches abitrary functions based on the function's arguments (fetch) or on a user defined key (fetchObjectId)

@remarks This version of DiskCached is threadsafe

fetch(*fetch_function*, **args*, ***kargs*)

fetches the object with the given id, querying

1. the cache and
2. the fetch_function

if the fetch_function is called, the functions result is saved in the cache

::param fetch_function: function to call if the result is not in the cache ::param args: arguments ::param kargs: optional keyword arguments

::returns: the object (retrieved from the cache or computed)

fetchObjectId(*key*, *fetch_function*, **args*, ***kargs*)

fetches the object with the given id, querying

- the cache and
- the fetch_function

if the fetch_function is called, the functions result is saved in the cache

::param key: key to fetch ::param fetch_function: function to call if the result is not in the cache ::param args: arguments ::param kargs: optional keyword arguments

::returns: the object (retrieved from the cache or computed)

getCacheStatistics()
returns statistics regarding the cache's hit/miss ratio

class eWRT.util.cache.**DiskCached**(*cache_dir*, *cache_nesting_level=0*, *cache_file_suffix=''*)
Bases: object

Decorator based on Cache for caching arbitrary function calls usage:

@DiskCached("./cache/myfunction") def myfunction(*args):

@remarks This version of DiskCached is threadsafe

cache

class eWRT.util.cache.**IterableCache**(*cache_dir*, *cache_nesting_level=0*, *cache_file_suffix=''*, *fn=None*)
Bases: *eWRT.util.cache.DiskCache*

caches arbitrary iterable content identified by an identifier

fetchObjectId(*key*, *function*, **args*, ***kargs*)

fetches the object with the given id, querying

1. the cache and
2. the function

if the function is called, the functions result is saved in the cache

::param key: key to fetch ::param function: function to call if the result is not in the cache ::param args: arguments ::param kargs: optional keyword arguments

::returns: the object (retrieved from the cache or computed)

next()

class eWRT.util.cache.**MemoryCache**(*max_cache_size=0*, *fn=None*)
Bases: *eWRT.util.cache.Cache*

@class MemoryCached

Caches abitrary functions based on the function's arguments (fetch) or on a user defined key (fetchObjectId)

fetch(*fetch_function*, **args*, ***kargs*)

fetchObjectId(*key*, *fetch_function*, **args*, ***kargs*)

garbage_collect_cache()
removes the object which have not been in use for the longest time

max_cache_size

class eWRT.util.cache.**MemoryCached**(*arg*)
Bases: *eWRT.util.cache.MemoryCache*

Decorator based on MemoryCache for caching arbitrary function calls usage:

@MemoryCached or @MemoryCached(max_cache_size) def myfunction(*args): ...

class eWRT.util.cache.**RedisCache**(*max_cache_size=0*, *fn=None*, *host='localhost'*, *port=6379*, *db=0*)
Bases: *eWRT.util.cache.Cache*

fetch(*fetch_function*, **args*, ***kargs*)

fetchObjectId(*key*, *fetch_function*, **args*, ***kargs*)

garbage_collect_cache()
removes the object which have not been in use for the longest time

class eWRT.util.cache.**RedisCached**(*arg*)
Bases: *eWRT.util.cache.RedisCache*

Decorator based on MemoryCache for caching arbitrary function calls usage:

@MemoryCached or @MemoryCached(max_cache_size) def myfunction(*args): ...

eWRT.util.cache.**get_unique_temp_file**(*fname*)

exception Module

@package eWRT.util.exception

exception eWRT.util.exception.**SNMPException**(*module_name*, *msg*, *level='warning'*)
Bases: exceptions.Exception

reports an exception to snmp

class eWRT.util.exception.**TestSNMPException**
Bases: object

testQuoting()

testSNMPException()
tests an SNMP exception

execute Module

@package eWRT.util.execute Helpers for executing third party modules

class eWRT.util.execute.**TestExecute**
Bases: object

test_pipe_content()

eWRT.util.execute.**pipe_content**(*cmd*, *stdin=None*)
Pipes the content through the given command. @param[in] cmd: command to be executed @param[in] stdin: standard input @return: (exit_status, stdout)

monitoring Module

The class NSCA-Service helps the enables to send test-results over the NSCA service to Nagios. This is a more reliable way of sending messages directly from programs than with SNMP. On the one hand because they are not only associated to single service, but also as the configuration is easier

== Configuration ==

- Install the package nsca

  aptitude install nsca

- On the monitoring system:

** edit the file ''/etc/nsca.cfg'': ** set a password and an appropriate encryption method

- On the host:

** edit the file ''/etc/send_nsca.cfg" ** enter the above password and encryption method

class eWRT.util.monitoring.**NSCA**
Bases: object

class NSCA Service helps sending test results to Nagios

static send(*message*, *status*, *service_name*, *src_host=None*, *performance=[]*, *monitoringServer="'tor.wu.ac.at"'*)
sends the

class eWRT.util.monitoring.**Performance**(*label*, *value*, *unit=''*, *warn=''*, *critical=''*, *min=''*, *max=''*)
Bases: object

pickleIterator Module

pickelIterator

class eWRT.util.pickleIterator.**AbstractIterator**(*fname*, *file_mode=None*)
Bases: object

Abstract Iterator class used to implement ReadPickleIterator and WritePickleIterator

close()

classmethod get_filename(*fname*)

next()
Python 2 compatibility

class eWRT.util.pickleIterator.**ReadPickleIterator**(*fname*)
Bases: *eWRT.util.pickleIterator.AbstractIterator*

provides an iterator over pickeled elements

class eWRT.util.pickleIterator.**WritePickleIterator**(*fname*)
Bases: *eWRT.util.pickleIterator.AbstractIterator*

writes pickeled elements (available as iterator) to a file

dump(*obj*)
dumps the following object to the pickle file

profile Module

@package eWRT.util.profile google like profiling :)

@warning this library is still a draft and might change considerable

eWRT.util.profile.**profile**(*fn*, *logfile='profile.awi'*)
profile function

eWRT.util.profile.**profileFunction**()
function to profile

eWRT.util.profile.**testProfile**()
test the profiling

timing Module

@package eWRT.util.timing timing of abitrary method calls

Examples: see unittests

class eWRT.util.timing.**Timed**(*f*)
Bases: object

decorator class used to time functions

clear()

class eWRT.util.timing.**TimedTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

testCallStatistics()

testClearCallStatistics()

testReturnValue()

ws Package

ws Package

@package eWRT.ws Web Service access.

class eWRT.ws.**AbstractIterableWebSource**
Bases: *eWRT.ws.AbstractWebSource*

web source implementing several calls to the API iterating over search terms and over API-specific maximal number of results restriction

DEFAULT_COMMAND = None

DEFAULT_FORMAT = None

DEFAULT_MAX_RESULTS = None

DEFAULT_START_INDEX = None

RESULT_PATH = None

invoke_iterator(*search_terms*, *max_results*, *from_date=None*, *to_date=None*, *command=None*, *output_format=None*)
iterator: iterates over search terms and API requests

process_output (*results*, *path*)
results' post-processor: iterates over the API responses and calls the output convertor

request (*search_term*, *current_index*, *max_results*, *from_date=None*, *to_date=None*, *command=None*, *output_format=None*)
calls the web source's API

search_documents (*search_terms*, *max_results=None*, *from_date=None*, *to_date=None*, ***kwargs*)
calls iterator and results' post-processor

class eWRT.ws.**AbstractWebSource**
Bases: object

a raw Web Source object

MAPPING = None

NAME = None

SUPPORTED_PARAMS = None

post_search (*search_params*)
override this function to perform post-run tasks

pre_search (*search_params*)
override this function to perform pre-run tasks

search_documents (*search_terms*, *max_results=None*, *from_date=None*, *to_date=None*, ***kwargs*)
runs the actual search / calls the webservice / API ...

TagInfoService Module

class eWRT.ws.TagInfoService.**TagInfoService**
Bases: object

Class for fetching assigned tags

getRelatedTags (*tags*, *retrieveTagInfo=False*)
returns a the count of related tags @param list/tuple of tags @param retrieveTagInfo determines whether we will retrieve the tagInfo for the related tags @returns list of related tags with a count of their occurence

getTagInfo (*tags*)
returns the count for the given tags @param list/tuple of tags @returns number of counts

WebDataSource Module

class eWRT.ws.WebDataSource.**WebDataSource**
Bases: object

A WebDataSource performs search queries against Web Sources such as Youtube, Twitter, ...

search (*search_terms*)
returns the count for the given tags @param search_terms: a list of search terms @returns a list of matching documents

Subpackages

amazon Package

initialize Package

create-distance-db Module

wot Package

wot Package

yahoo Package

yahoo Package

term_extractor Module

youtube Package

youtube Package

CHAPTER 3

Features

eWRT provides the following features:

- content acquisition components for the Web (see `eWRT.access.http`) and different social media sources (see *eWRT.ws*)
- low-level natural language processing, e.g.
  - language detection (*eWRT.stat.language*)
  -
- content caching (see *eWRT.util.cache*)

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

Index

F

G

H

I

L

M

N

O

P

R

S

T

W