RaRe-Technologies / **gensim**

Branch: **develop** **gensim** / gensim / corpora / **wikicorpus.py** **Find file** **Copy path**

**menshikh-iv** Add tox and pytest to gensim, integration with Travis and Appveyor. Fix 8766edc 25 days ago

**18** contributors

Executable File 398 lines (322 sloc) 16.3 KB

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
# Copyright (C) 2010 Radim Rehurek <radimrehurek@seznam.cz>
# Copyright (C) 2012 Lars Buitinck <larsmans@gmail.com>
# Licensed under the GNU LGPL v2.1 - http://www.gnu.org/licenses/lgpl.html


"""
Construct a corpus from a Wikipedia (or other MediaWiki-based) database dump.

If you have the `pattern` package installed, this module will use a fancy
lemmatization to get a lemma of each token (instead of plain alphabetic
tokenizer). The package is available at https://github.com/clips/pattern .

See scripts/process_wiki.py for a canned (example) script based on this
module.
"""


import bz2
import logging
```

```
import multiprocessing
import re
import signal
from xml.etree.cElementTree import \
    iterparse  # LXML isn't faster, so let's go with the built-in solution

from gensim import utils
# cannot import whole gensim.corpora, because that imports wikicorpus...
from gensim.corpora.dictionary import Dictionary
from gensim.corpora.textcorpus import TextCorpus

logger = logging.getLogger(__name__)

# ignore articles shorter than ARTICLE_MIN_WORDS characters (after full preprocessing)
ARTICLE_MIN_WORDS = 50

# default thresholds for lengths of individual tokens
TOKEN_MIN_LEN = 2
TOKEN_MAX_LEN = 15


RE_P0 = re.compile(r'<!--.*?-->', re.DOTALL | re.UNICODE)  # comments
RE_P1 = re.compile(r'<ref([> ].*?)(</ref>|/>)', re.DOTALL | re.UNICODE)  # footnotes
RE_P2 = re.compile(r'(\n\[\[[a-z][a-z][\w-]*:[^:\]]+\]\])+$', re.UNICODE)  # links to languages
RE_P3 = re.compile(r'{{([^{]*)}}', re.DOTALL | re.UNICODE)  # template
RE_P4 = re.compile(r'{{([^}]*)}}', re.DOTALL | re.UNICODE)  # template
RE_P5 = re.compile(r'\[(\w+):\/\/(.*?)(( (.*?))|())\]', re.UNICODE)  # remove URL, keep description
RE_P6 = re.compile(r'\[([^][]*)\|([^][]*)\]', re.DOTALL | re.UNICODE)  # simplify links, keep description
RE_P7 = re.compile(r'\n\[\[iI]mage(.*?)(\|.*?)*\|(.*?)\]\]', re.UNICODE)  # keep description of images
RE_P8 = re.compile(r'\n\[\[fF]ile(.*?)(\|.*?)*\|(.*?)\]\]', re.UNICODE)  # keep description of files
RE_P9 = re.compile(r'<nowiki([> ].*?)(</nowiki>|/>)', re.DOTALL | re.UNICODE)  # outside links
RE_P10 = re.compile(r'<math([> ].*?)(</math>|/>)', re.DOTALL | re.UNICODE)  # math content
RE_P11 = re.compile(r'<(.*?)>', re.DOTALL | re.UNICODE)  # all other tags
RE_P12 = re.compile(r'\n({\|)|(\|-)|(\|})(.*?)(?=\n)', re.UNICODE)  # table formatting
RE_P13 = re.compile(r'\n(\||\!)(.*?\|)*([^|]*?)', re.UNICODE)  # table cell formatting
```

```
58  RE_P14 = re.compile(r'\[\[Category:[^][]*\]\]', re.UNICODE)  # categories
59  # Remove File and Image template
60  RE_P15 = re.compile(r'\[\[([fF]ile:|[iI]mage)[^]]*(\]\])', re.UNICODE)
61
62  # MediaWiki namespaces (https://www.mediawiki.org/wiki/Manual:Namespace) that
63  # ought to be ignored
64  IGNORED_NAMESPACES = [
65      'Wikipedia', 'Category', 'File', 'Portal', 'Template',
66      'MediaWiki', 'User', 'Help', 'Book', 'Draft', 'WikiProject',
67      'Special', 'Talk'
68  ]
69
70
71  def filter_wiki(raw):
72      """
73      Filter out wiki mark-up from `raw`, leaving only text. `raw` is either unicode
74      or utf-8 encoded string.
75      """
76      # parsing of the wiki markup is not perfect, but sufficient for our purposes
77      # contributions to improving this code are welcome :)
78      text = utils.to_unicode(raw, 'utf8', errors='ignore')
79      text = utils.decode_htmlentities(text)  # '&amp;nbsp;' --> '\xa0'
80      return remove_markup(text)
81
82
83  def remove_markup(text):
84      text = re.sub(RE_P2, '', text)  # remove the last list (=languages)
85      # the wiki markup is recursive (markup inside markup etc)
86      # instead of writing a recursive grammar, here we deal with that by removing
87      # markup in a loop, starting with inner-most expressions and working outwards,
88      # for as long as something changes.
89      text = remove_template(text)
90      text = remove_file(text)
91      iters = 0
92      while True:
```

```
        old, iters = text, iters + 1
        text = re.sub(RE_P0, '', text)  # remove comments
        text = re.sub(RE_P1, '', text)  # remove footnotes
        text = re.sub(RE_P9, '', text)  # remove outside links
        text = re.sub(RE_P10, '', text)  # remove math content
        text = re.sub(RE_P11, '', text)  # remove all remaining tags
        text = re.sub(RE_P14, '', text)  # remove categories
        text = re.sub(RE_P5, '\\3', text)  # remove urls, keep description
        text = re.sub(RE_P6, '\\2', text)  # simplify links, keep description only
        # remove table markup
        text = text.replace('||', '\n|')  # each table cell on a separate line
        text = re.sub(RE_P12, '\n', text)  # remove formatting lines
        text = re.sub(RE_P13, '\n\\3', text)  # leave only cell content
        # remove empty mark-up
        text = text.replace('[]', '')
        # stop if nothing changed between two iterations or after a fixed number of iterations
        if old == text or iters > 2:
            break

    # the following is needed to make the tokenizer see '[[socialist]]s' as a single word 'socialists'
    # TODO is this really desirable?
    text = text.replace('[', '').replace(']', '')  # promote all remaining markup to plain text
    return text


def remove_template(s):
    """Remove template wikimedia markup.

    Return a copy of `s` with all the wikimedia markup template removed. See
    http://meta.wikimedia.org/wiki/Help:Template for wikimedia templates
    details.

    Note: Since template can be nested, it is difficult remove them using
    regular expresssions.
    """
```

```
128
129     # Find the start and end position of each template by finding the opening
130     # '{{' and closing '}}'
131     n_open, n_close = 0, 0
132     starts, ends = [], []
133     in_template = False
134     prev_c = None
135     for i, c in enumerate(iter(s)):
136         if not in_template:
137             if c == '{' and c == prev_c:
138                 starts.append(i - 1)
139                 in_template = True
140                 n_open = 1
141         if in_template:
142             if c == '{':
143                 n_open += 1
144             elif c == '}':
145                 n_close += 1
146             if n_open == n_close:
147                 ends.append(i)
148                 in_template = False
149                 n_open, n_close = 0, 0
150         prev_c = c
151
152     # Remove all the templates
153     return ''.join([s[end + 1:start] for start, end in zip(starts + [None], [-1] + ends)])
154
155
156 def remove_file(s):
157     """Remove the 'File:' and 'Image:' markup, keeping the file caption.
158
159     Return a copy of `s` with all the 'File:' and 'Image:' markup replaced by
160     their corresponding captions. See http://www.mediawiki.org/wiki/Help:Images
161     for the markup details.
162     """
```

```
    # The regex RE_P15 match a File: or Image: markup
    for match in re.finditer(RE_P15, s):
        m = match.group(0)
        caption = m[:-2].split('|')[-1]
        s = s.replace(m, caption, 1)
    return s


def tokenize(content, token_min_len=TOKEN_MIN_LEN, token_max_len=TOKEN_MAX_LEN, lower=True):
    """
    Tokenize a piece of text from wikipedia. The input string `content` is assumed
    to be mark-up free (see `filter_wiki()`).

    Set `token_min_len`, `token_max_len` as character length (not bytes!) thresholds for individual tokens.

    Return list of tokens as utf8 bytestrings.
    """
    # TODO maybe ignore tokens with non-latin characters? (no chinese, arabic, russian etc.)
    return [
        utils.to_unicode(token) for token in utils.tokenize(content, lower=lower, errors='ignore')
        if token_min_len <= len(token) <= token_max_len and not token.startswith('_')
    ]


def get_namespace(tag):
    """Returns the namespace of tag."""
    m = re.match("^{(.*?)}", tag)
    namespace = m.group(1) if m else ""
    if not namespace.startswith("http://www.mediawiki.org/xml/export-"):
        raise ValueError("%s not recognized as MediaWiki dump namespace" % namespace)
    return namespace


_get_namespace = get_namespace

```

```
198
199 def extract_pages(f, filter_namespaces=False):
200     """
201     Extract pages from a MediaWiki database dump = open file-like object `f`.
202
203     Return an iterable over (str, str, str) which generates (title, content, pageid) triplets.
204
205     """
206     elems = (elem for _, elem in iterparse(f, events=("end",)))
207
208     # We can't rely on the namespace for database dumps, since it's changed
209     # it every time a small modification to the format is made. So, determine
210     # those from the first element we find, which will be part of the metadata,
211     # and construct element paths.
212     elem = next(elems)
213     namespace = get_namespace(elem.tag)
214     ns_mapping = {"ns": namespace}
215     page_tag = "{%(ns)s}page" % ns_mapping
216     text_path = "./{%(ns)s}revision/{%(ns)s}text" % ns_mapping
217     title_path = "./{%(ns)s}title" % ns_mapping
218     ns_path = "./{%(ns)s}ns" % ns_mapping
219     pageid_path = "./{%(ns)s}id" % ns_mapping
220
221     for elem in elems:
222         if elem.tag == page_tag:
223             title = elem.find(title_path).text
224             text = elem.find(text_path).text
225
226             if filter_namespaces:
227                 ns = elem.find(ns_path).text
228                 if ns not in filter_namespaces:
229                     text = None
230
231             pageid = elem.find(pageid_path).text
232             yield title, text or "", pageid     # empty page will yield None
```

```
233
234             # Prune the element tree, as per
235             # http://www.ibm.com/developerworks/xml/library/x-hiperfparse/
236             # except that we don't need to prune backlinks from the parent
237             # because we don't use LXML.
238             # We do this only for <page>s, since we need to inspect the
239             # ./revision/text element. The pages comprise the bulk of the
240             # file, so in practice we prune away enough.
241             elem.clear()
242
243
244 _extract_pages = extract_pages  # for backward compatibility
245
246
247 def process_article(args, tokenizer_func=tokenize, token_min_len=TOKEN_MIN_LEN,
248                     token_max_len=TOKEN_MAX_LEN, lower=True):
249     """
250     Parse a wikipedia article, returning its content as a list of tokens
251     (utf8-encoded strings).
252
253     Set `tokenizer_func` (defaults to `tokenize`) parameter for languages like japanese or thai to perform better
254     tokenization. The `tokenizer_func` needs to take 4 parameters: (text, token_min_len, token_max_len, lower).
255     """
256     text, lemmatize, title, pageid = args
257     text = filter_wiki(text)
258     if lemmatize:
259         result = utils.lemmatize(text)
260     else:
261         result = tokenizer_func(text, token_min_len, token_max_len, lower)
262     return result, title, pageid
263
264
265 def init_to_ignore_interrupt():
266     """Should only be used when master is prepared to handle termination of child processes."""
267     signal.signal(signal.SIGINT, signal.SIG_IGN)
```

```
268
269
270 def _process_article(args):
271     """Should not be called explicitly. Use `process_article` instead."""
272
273     tokenizer_func, token_min_len, token_max_len, lower = args[-1]
274     args = args[:-1]
275
276     return process_article(
277         args, tokenizer_func=tokenizer_func, token_min_len=token_min_len,
278         token_max_len=token_max_len, lower=lower
279     )
280
281
282 class WikiCorpus(TextCorpus):
283     """
284     Treat a wikipedia articles dump (<LANG>wiki-<YYYYMMDD>-pages-articles.xml.bz2
285     or <LANG>wiki-latest-pages-articles.xml.bz2) as a (read-only) corpus.
286
287     The documents are extracted on-the-fly, so that the whole (massive) dump
288     can stay compressed on disk.
289
290     **Note:** "multistream" archives are *not* supported in Python 2 due to
291     `limitations in the core bz2 library
292     <https://docs.python.org/2/library/bz2.html#de-compression-of-files>`_.
293
294     >>> wiki = WikiCorpus('enwiki-20100622-pages-articles.xml.bz2') # create word->word_id mapping, takes almost 8h
295     >>> MmCorpus.serialize('wiki_en_vocab200k.mm', wiki) # another 8h, creates a file in MatrixMarket format and mapping
296
297     """
298     def __init__(self, fname, processes=None, lemmatize=utils.has_pattern(), dictionary=None,
299                  filter_namespaces=('0',), tokenizer_func=tokenize, article_min_tokens=ARTICLE_MIN_WORDS,
300                  token_min_len=TOKEN_MIN_LEN, token_max_len=TOKEN_MAX_LEN, lower=True):
301         """
302         Initialize the corpus. Unless a dictionary is provided, this scans the
```

```
        corpus once, to determine its vocabulary.

        If `pattern` package is installed, use fancier shallow parsing to get
        token lemmas. Otherwise, use simple regexp tokenization. You can override
        this automatic logic by forcing the `lemmatize` parameter explicitly.
        self.metadata if set to true will ensure that serialize will write out article titles to a pickle file.

        Set `article_min_tokens` as a min threshold for article token count (defaults to 50). Any article below this is
        ignored.

        Set `tokenizer_func` (defaults to `tokenize`) with a custom function reference to control tokenization else use
        the default regexp tokenization. Set this parameter for languages like japanese or thai to perform better
        tokenization. The `tokenizer_func` needs to take 4 parameters: (text, token_min_len, token_max_len, lower). The
        parameter values are as configured on the class instance by default.

        Set `lower` to control if everything should be converted to lowercase or not (default True).

        Set `token_min_len`, `token_max_len` as thresholds for token lengths that are returned (default to 2 and 15).

        """
        self.fname = fname
        self.filter_namespaces = filter_namespaces
        self.metadata = False
        if processes is None:
            processes = max(1, multiprocessing.cpu_count() - 1)
        self.processes = processes
        self.lemmatize = lemmatize
        self.tokenizer_func = tokenizer_func
        self.article_min_tokens = article_min_tokens
        self.token_min_len = token_min_len
        self.token_max_len = token_max_len
        self.lower = lower

        if dictionary is None:
            self.dictionary = Dictionary(self.get_texts())
```

```
        else:
            self.dictionary = dictionary

    def get_texts(self):
        """
        Iterate over the dump, returning text version of each article as a list
        of tokens.

        Only articles of sufficient length are returned (short articles & redirects
        etc are ignored). This is control by `article_min_tokens` on the class instance.

        Note that this iterates over the **texts**; if you want vectors, just use
        the standard corpus interface instead of this function::

        >>> for vec in wiki_corpus:
        >>>     print(vec)
        """

        articles, articles_all = 0, 0
        positions, positions_all = 0, 0

        tokenization_params = (self.tokenizer_func, self.token_min_len, self.token_max_len, self.lower)
        texts = \
            ((text, self.lemmatize, title, pageid, tokenization_params)
             for title, text, pageid
             in extract_pages(bz2.BZ2File(self.fname), self.filter_namespaces))
        pool = multiprocessing.Pool(self.processes, init_to_ignore_interrupt)

        try:
            # process the corpus in smaller chunks of docs, because multiprocessing.Pool
            # is dumb and would load the entire input into RAM at once...
            for group in utils.chunkize(texts, chunksize=10 * self.processes, maxsize=1):
                for tokens, title, pageid in pool.imap(_process_article, group):
                    articles_all += 1
                    positions_all += len(tokens)
```

```
                    # article redirects and short stubs are pruned here
                    if len(tokens) < self.article_min_tokens or \
                            any(title.startswith(ignore + ':') for ignore in IGNORED_NAMESPACES):
                        continue
                    articles += 1
                    positions += len(tokens)
                    if self.metadata:
                        yield (tokens, (pageid, title))
                    else:
                        yield tokens
        except KeyboardInterrupt:
            logger.warn(
                "user terminated iteration over Wikipedia corpus after %i documents with %i positions "
                "(total %i articles, %i positions before pruning articles shorter than %i words)",
                articles, positions, articles_all, positions_all, ARTICLE_MIN_WORDS
            )
        else:
            logger.info(
                "finished iterating over Wikipedia corpus of %i documents with %i positions "
                "(total %i articles, %i positions before pruning articles shorter than %i words)",
                articles, positions, articles_all, positions_all, ARTICLE_MIN_WORDS
            )
            self.length = articles  # cache corpus length
        finally:
            pool.terminate()
```