abstrctGenerat系统完整测试代码

V1.0

# 1.test\_keywords.py:

**import** unittest  
  
**from** summa.keywords **import** keywords  
**from** summa.preprocessing.textcleaner **import** deaccent  
**from** numpy **import** isclose  
**from** .utils **import** get\_text\_from\_test\_data  
  
  
**class** TestKeywords(unittest.TestCase):  
  
 **def** test\_text\_keywords(self):  
 text = get\_text\_from\_test\_data(**"mihalcea\_tarau.txt"**)  
  
 *# Calculate keywords* generated\_keywords = keywords(text, split=**True**)  
  
 *# To be compared to the reference.* reference\_keywords = get\_text\_from\_test\_data(**"mihalcea\_tarau.kw.txt"**).split(**"\n"**)  
  
 **return** self.assertEqual({str(x) **for** x **in** generated\_keywords}, {str(x) **for** x **in** reference\_keywords}), generated\_keywords  
  
  
 **def** test\_text\_keywords\_wempty\_stoplist(self):  
 text = get\_text\_from\_test\_data(**"mihalcea\_tarau.txt"**)  
 additional\_stoplist = []  
 generated\_keywords = keywords(text, split=**True**, additional\_stopwords=additional\_stoplist)  
 reference\_keywords = get\_text\_from\_test\_data(**"mihalcea\_tarau.kw.txt"**).split(**"\n"**)  
 **return** self.assertEqual({str(x) **for** x **in** generated\_keywords}, {str(x) **for** x **in** reference\_keywords}),generated\_keywords  
  
 **def** test\_text\_keywords\_wstoplist(self):  
 text = get\_text\_from\_test\_data(**"mihalcea\_tarau.txt"**)  
 additional\_stoplist = get\_text\_from\_test\_data(**"mihalcea\_tarau.sw.txt"**).strip().split(**","**)  
 generated\_keywords = keywords(text, split=**True**, additional\_stopwords=additional\_stoplist)  
 reference\_keywords = get\_text\_from\_test\_data(**"mihalcea\_tarau.swkw.txt"**).split(**"\n"**)  
 **return** self.assertEqual({str(x) **for** x **in** generated\_keywords}, {str(x) **for** x **in** reference\_keywords}),generated\_keywords  
  
 **def** test\_keywords\_few\_distinct\_words\_is\_empty\_string(self):  
 text = get\_text\_from\_test\_data(**"few\_distinct\_words.txt"**)  
 **return** self.assertEqual(keywords(text), **""**)  
  
 **def** test\_keywords\_few\_distinct\_words\_wempty\_stoplist\_is\_empty\_string(self):  
 text = get\_text\_from\_test\_data(**"few\_distinct\_words.txt"**)  
 self.assertEqual(keywords(text,additional\_stopwords=[]), **""**)  
  
 **def** test\_keywords\_few\_distinct\_words\_w\_stoplist\_is\_empty\_string(self):  
 text = get\_text\_from\_test\_data(**"few\_distinct\_words.txt"**)  
 additional\_stopwords = [**"here"**,**"there"**]  
 **return** self.assertEqual(keywords(text,additional\_stopwords=additional\_stopwords), **""**)  
  
 **def** test\_keywords\_few\_distinct\_words\_split\_is\_empty\_list(self):  
 text = get\_text\_from\_test\_data(**"few\_distinct\_words.txt"**)  
 **return** self.assertEqual(keywords(text, split=**True**), [])  
  
 **def** test\_keywords\_few\_distinct\_words\_wempty\_stoplist\_split\_is\_empty\_list(self):  
 text = get\_text\_from\_test\_data(**"few\_distinct\_words.txt"**)  
 self.assertEqual(keywords(text, split=**True**, additional\_stopwords=[]), [])  
  
 **def** test\_keywords\_few\_distinct\_words\_w\_stoplist\_split\_is\_empty\_list(self):  
 text = get\_text\_from\_test\_data(**"few\_distinct\_words.txt"**)  
 additional\_stopwords = [**"here"**,**"there"**]  
 self.assertEqual(keywords(text, split=**True**, additional\_stopwords=additional\_stopwords), [])  
  
 **def** test\_text\_summarization\_on\_short\_input\_text\_and\_split\_is\_not\_empty\_list(self):  
 text = get\_text\_from\_test\_data(**"unrelated.txt"**)  
  
 *# Keeps the first 8 sentences to make the text shorter.* text = **"\n"**.join(text.split(**'\n'**)[:8])  
  
 **return** self.assertNotEqual(keywords(text, split=**True**), [])  
  
 **def** test\_text\_summarization\_on\_short\_input\_text\_is\_not\_empty\_string(self):  
 text = get\_text\_from\_test\_data(**"unrelated.txt"**)  
  
 *# Keeps the first 8 sentences to make the text shorter.* text = **"\n"**.join(text.split(**'\n'**)[:8])  
  
 self.assertNotEqual(keywords(text, split=**True**), **""**)  
  
 **def** test\_keywords\_ratio(self):  
 text = get\_text\_from\_test\_data(**"mihalcea\_tarau.txt"**)  
  
 *# Check ratio parameter is well behaved.  
 # Because length is taken on tokenized clean text we just check that  
 # ratio 40% is twice as long as ratio 20%* selected\_docs\_20 = keywords(text, ratio=0.2, split=**True**)  
 selected\_docs\_40 = keywords(text, ratio=0.4, split=**True**)  
  
 **return** self.assertAlmostEqual(float(len(selected\_docs\_40)) / len(selected\_docs\_20), 0.4 / 0.2, places=1)  
  
 **def** test\_keywords\_ratio\_wstopwords(self):  
 text = get\_text\_from\_test\_data(**"mihalcea\_tarau.txt"**)  
 additional\_stoplist = get\_text\_from\_test\_data(**"mihalcea\_tarau.sw.txt"**).strip().split(**","**)  
 *# Check ratio parameter is well behaved.  
 # Because length is taken on tokenized clean text we just check that  
 # ratio 40% is twice as long as ratio 20%* selected\_docs\_20 = keywords(text, ratio=0.2, split=**True**, additional\_stopwords=additional\_stoplist)  
 selected\_docs\_40 = keywords(text, ratio=0.4, split=**True**, additional\_stopwords=additional\_stoplist)  
  
 actual\_ratio = float(len(selected\_docs\_40)) / len(selected\_docs\_20)  
 expected\_ratio = 0.4 / 0.2  
 *# Expect the same ratio with a relative tolerance of 5%.* **return** self.assertTrue(isclose(actual\_ratio, expected\_ratio, rtol=0.5), **"Ratio between number of keywords should be 2."**)  
  
 **def** test\_keywords\_consecutive\_keywords(self):  
 text = **"Rabbit populations known to be plentiful, large, and diverse \  
 in the area. \  
 Adjacent to the site, a number number well over a thousand. \  
 The number of these rabbit populations has diminished in recent \  
 years, and perhaps we have become number to a number of their \  
 numbers numbering fewer."** *# Should not raise an exception.* self.assertIsNotNone(keywords(text, words=10))  
  
 **def** test\_repeated\_keywords(self):  
 text = get\_text\_from\_test\_data(**"repeated\_keywords.txt"**)  
 kwds = keywords(text)  
 print(len(kwds.splitlines())) *#len = 12* **return** self.assertTrue(len(kwds.splitlines())),text  
  
 **def** test\_repeated\_keywords\_wstopwords(self):  
 text = get\_text\_from\_test\_data(**"repeated\_keywords.txt"**)  
 additional\_stoplist = [**"sage"**,**"user"**]  
 kwds = keywords(text,additional\_stopwords=additional\_stoplist)  
 **return** self.assertTrue(len(kwds.splitlines()))  
  
 **def** test\_spanish\_without\_accents(self):  
 *# Test the keyword extraction with accented characters.* text = get\_text\_from\_test\_data(**"spanish.txt"**)  
 kwds = keywords(text, language=**"spanish"**, deaccent=**True**, split=**True**)  
 *# Verifies that all words are retrieved without accents.* **return** self.assertTrue(all(deaccent(keyword) == keyword **for** keyword **in** kwds))  
  
 **def** test\_spanish\_with\_accents(self):  
 *# Test the keyword extraction with accented characters.* text = get\_text\_from\_test\_data(**"spanish.txt"**)  
 kwds = keywords(text, language=**"spanish"**, deaccent=**False**, split=**True**)  
 *# Verifies that there are some keywords are retrieved with accents.* **return** self.assertTrue(any(deaccent(keyword) != keyword **for** keyword **in** kwds))  
  
 **def** test\_text\_as\_bytes\_raises\_exception(self):  
 *# Test the keyword extraction for a text that is not a unicode object  
 # (Python 3 str).* text = get\_text\_from\_test\_data(**"spanish.txt"**)  
 bytes = text.encode(encoding=**"utf-8"**)  
 **with** self.assertRaises(ValueError):  
 keywords(bytes, language=**"spanish"**)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 unittest.main()

# 2.test\_summarizer.py

**import** unittest  
  
**from** summa.summarizer **import** summarize  
**from** .utils **import** get\_text\_from\_test\_data  
  
  
**class** TestSummarizer(unittest.TestCase):  
  
 **def** test\_reference\_text\_summarization(self):  
 text = get\_text\_from\_test\_data(**"mihalcea\_tarau.txt"**)  
  
 *# Makes a summary of the text.* generated\_summary = summarize(text)  
  
 *# To be compared to the method reference.与相关的方法进行比较* summary = get\_text\_from\_test\_data(**"mihalcea\_tarau.summ.txt"**)  
  
 **return** self.assertEqual(generated\_summary, summary)  
  
 **def** test\_reference\_text\_summarization\_wstopwords(self):  
 text = get\_text\_from\_test\_data(**"mihalcea\_tarau.txt"**)  
 *# Identify non-index words 规定非索引词* additional\_stoplist = get\_text\_from\_test\_data(**"mihalcea\_tarau.sw.txt"**).strip().split(**","**)  
 *# Makes a summary of the text.* generated\_summary = summarize(text,additional\_stopwords=additional\_stoplist)  
  
 *# To be compared to the method reference.* summary = get\_text\_from\_test\_data(**"mihalcea\_tarau.summ.txt"**)  
  
 **return** self.assertEqual(generated\_summary, summary, msg=**'生成文本摘要失败'**),generated\_summary  
  
 **def** test\_reference\_text\_summarization\_with\_split(self):  
 text = get\_text\_from\_test\_data(**"mihalcea\_tarau.txt"**)  
  
 *# Makes a summary of the text as a list.* generated\_summary = summarize(text, split=**True**)  
  
 *# To be compared to the method reference.* summary = get\_text\_from\_test\_data(**"mihalcea\_tarau.summ.txt"**)  
 summary = summary.split(**"\n"**)  
  
 self.assertSequenceEqual(generated\_summary, summary)  
  
 **def** test\_reference\_text\_summarization\_wstopwords\_with\_split(self):  
 text = get\_text\_from\_test\_data(**"mihalcea\_tarau.txt"**)  
 additional\_stoplist = get\_text\_from\_test\_data(**"mihalcea\_tarau.sw.txt"**).strip().split(**","**)  
  
 *# Makes a summary of the text as a list.* generated\_summary = summarize(text, split=**True**, additional\_stopwords=additional\_stoplist)  
  
 *# To be compared to the method reference.* summary = get\_text\_from\_test\_data(**"mihalcea\_tarau.summ.txt"**)  
 summary = summary.split(**"\n"**)  
  
 self.assertSequenceEqual(generated\_summary, summary)  
  
 **def** test\_few\_distinct\_words\_summarization\_is\_empty\_string(self):  
 text = get\_text\_from\_test\_data(**"few\_distinct\_words.txt"**)  
 self.assertEqual(summarize(text), **""**)  
 **return** text  
  
 **def** test\_few\_distinct\_words\_summarization\_with\_split\_is\_empty\_list(self):  
 text = get\_text\_from\_test\_data(**"few\_distinct\_words.txt"**)  
 self.assertEqual(summarize(text, split=**True**), [])  
 **return** text  
  
 **def** test\_few\_distinct\_words\_summarization\_wstopwords\_is\_empty\_string(self):  
 text = get\_text\_from\_test\_data(**"few\_distinct\_words.txt"**)  
 additional\_stoplist = [**"here"**,**"there"**]  
 self.assertEqual(summarize(text, additional\_stopwords=additional\_stoplist), **""**)  
  
 **def** test\_few\_distinct\_words\_summarization\_wstopwords\_with\_split\_is\_empty\_list(self):  
 text = get\_text\_from\_test\_data(**"few\_distinct\_words.txt"**)  
 additional\_stoplist = [**"here"**,**"there"**]  
 self.assertEqual(summarize(text, split=**True**, additional\_stopwords=additional\_stoplist), [])  
  
 **def** test\_summary\_from\_unrelated\_sentences\_is\_not\_empty\_string(self):  
 *# Tests that the summarization of a text with unrelated sentences is not empty string.* text = get\_text\_from\_test\_data(**"unrelated.txt"**)  
 self.assertNotEqual(summarize(text), **""**)  
 **return** text  
  
 **def** test\_summary\_from\_unrelated\_sentences\_and\_split\_is\_not\_empty\_list(self):  
 *# Tests that the summarization of a text with unrelated sentences is not empty string.* text = get\_text\_from\_test\_data(**"unrelated.txt"**)  
 self.assertNotEqual(summarize(text, split=**True**), [])  
  
 **def** test\_text\_summarization\_on\_short\_input\_text\_is\_not\_empty\_string(self):  
 text = get\_text\_from\_test\_data(**"unrelated.txt"**)  
  
 *# Keeps the first 8 sentences to make the text shorter.* text = **"\n"**.join(text.split(**'\n'**)[:8])  
  
 self.assertNotEqual(summarize(text), **""**)  
  
 **def** test\_text\_summarization\_on\_short\_input\_text\_with\_split\_is\_not\_empty\_list(self):  
 text = get\_text\_from\_test\_data(**"unrelated.txt"**)  
  
 *# Keeps the first 8 sentences to make the text shorter.* text = **"\n"**.join(text.split(**'\n'**)[:8])  
  
 self.assertNotEqual(summarize(text, split=**True**), [])  
  
 **def** test\_text\_summarization\_on\_single\_input\_sentence\_is\_empty\_string(self):  
 text = get\_text\_from\_test\_data(**"unrelated.txt"**)  
  
 *# Keeps the first sentence only.* text = text.split(**'\n'**)[0]  
  
 self.assertEqual(summarize(text), **""**)  
  
 **def** test\_text\_summarization\_on\_single\_input\_sentence\_with\_split\_is\_empty\_list(self):  
 text = get\_text\_from\_test\_data(**"unrelated.txt"**)  
  
 *# Keeps the first sentence only.* text = text.split(**'\n'**)[0]  
  
 self.assertEqual(summarize(text, split=**True**), [])  
  
 **def** test\_empty\_text\_summarization\_is\_empty\_string(self):  
 self.assertEqual(summarize(**""**), **""**)  
  
 **def** test\_empty\_text\_summarization\_with\_split\_is\_empty\_list(self):  
 self.assertEqual(summarize(**""**, split=**True**), [])  
  
 **def** test\_corpus\_summarization\_ratio(self):  
 text = get\_text\_from\_test\_data(**"mihalcea\_tarau.txt"**)  
  
 sentences = text.split(**'\n'**)  
  
 *# Makes summaries of the text using different ratio parameters.* **for** x **in** range(1, 10):  
 ratio = x / float(10)  
 selected\_sentences = summarize(text, ratio=ratio, split=**True**)  
 expected\_summary\_length = int(len(sentences) \* ratio)  
  
 self.assertEqual(len(selected\_sentences), expected\_summary\_length)  
  
 **def** test\_spanish(self):  
 *# Test the summarization module with accented characters.* text = get\_text\_from\_test\_data(**"spanish.txt"**)  
 self.assertIsNotNone(summarize(text, language=**"spanish"**))  
 **return** text  
  
 **def** test\_polish(self):  
 *# Test the summarization module for Polish language.* text = get\_text\_from\_test\_data(**"polish.txt"**)  
 self.assertIsNotNone(summarize(text, language=**"polish"**))  
 **return** text  
  
 **def** test\_text\_as\_bytes\_raises\_exception(self):  
 *# Test the keyword extraction for a text that is not a unicode object  
 # (Python 3 str).* text = get\_text\_from\_test\_data(**"spanish.txt"**)  
 bytes = text.encode(encoding=**"utf-8"**)  
 **with** self.assertRaises(ValueError):  
 summarize(bytes, language=**"spanish"**)  
 **return** text  
   
 **def** test\_arabic(self):  
 *# Test the summarization module for arabic language.* text = get\_text\_from\_test\_data(**"arabic.txt"**)  
 self.assertIsNotNone(summarize(text, language=**'arabic'**))  
 **return** text  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 unittest.main()

# 3.test\_polish.py

**import** unittest  
**from** summa.summarizer **import** summarize  
**from** test.test\_summarizer **import** TestSummarizer  
  
t = TestSummarizer();  
text = t.test\_polish();  
print(summarize(text, words=30));

# 4.test\_arabic.py

**import** unittest  
**from** summa.summarizer **import** summarize  
**from** test.test\_summarizer **import** TestSummarizer  
  
t = TestSummarizer();  
text = t.test\_arabic();  
print(summarize(text, words=30));

# 5.test\_spanish.py

**import** unittest  
**from** summa.summarizer **import** summarize  
**from** test.test\_summarizer **import** TestSummarizer  
  
t = TestSummarizer();  
text = t.test\_spanish();  
print(summarize(text, words=30));

# 6.test\_unrelated.py

**import** unittest  
**from** summa.summarizer **import** summarize  
**from** test.test\_summarizer **import** TestSummarizer  
  
t = TestSummarizer();  
text = t.test\_summary\_from\_unrelated\_sentences\_is\_not\_empty\_string();  
print(summarize(text));

# 7.test\_few\_distinct\_words.py

**import** unittest  
**from** summa.summarizer **import** summarize  
**from** test.test\_summarizer **import** TestSummarizer  
  
t = TestSummarizer();  
text = t.test\_few\_distinct\_words\_summarization\_with\_split\_is\_empty\_list();  
print(summarize(text));

# 8.test\_mihalcea\_wsl.py

**import** unittest  
**from** summa.summarizer **import** summarize  
**from** test.test\_summarizer **import** TestSummarizer  
  
t = TestSummarizer();  
**'''  
receive two results returned by function t.test\_reference\_text\_summarization\_wstopwords;  
the first one represents whether the results is correct, and wherever, the second will return  
the abstract gerenated.  
'''**res = t.test\_reference\_text\_summarization\_wstopwords();  
print(res)

# 9.test\_kw\_wtsplit.py

**import** unittest  
  
**from** summa.keywords **import** keywords  
**from** summa.preprocessing.textcleaner **import** deaccent  
**from** test.test\_keywords **import** TestKeywords  
**'''  
test generate keywords function with split = True  
'''**t = TestKeywords();  
res = t.test\_text\_keywords()  
print(res)

# 10.test\_kw\_without\_stoplist.py

**import** unittest  
**from** summa.keywords **import** keywords  
**from** summa.preprocessing.textcleaner **import** deaccent  
**from** test.test\_keywords **import** TestKeywords  
**'''  
test generate keywords function without split = True  
'''**t = TestKeywords();  
res = t.test\_text\_keywords\_wempty\_stoplist()  
print(res)

# 11.test\_kw\_with\_stoplist.py

**import** unittest  
**from** summa.keywords **import** keywords  
**from** summa.preprocessing.textcleaner **import** deaccent  
**from** test.test\_keywords **import** TestKeywords  
**'''  
test few distinct words with stoplist=['here', 'there'] and split = True  
'''**t = TestKeywords();  
res = t.test\_keywords\_few\_distinct\_words\_w\_stoplist\_is\_empty\_string()  
print(res)

# 12.test\_kw\_few\_distinct\_words\_with\_splitn.py

**import** unittest  
**from** summa.keywords **import** keywords  
**from** summa.preprocessing.textcleaner **import** deaccent  
**from** test.test\_keywords **import** TestKeywords  
**'''  
test few distinct words with split = True  
'''**t = TestKeywords();  
res = t.test\_keywords\_few\_distinct\_words\_split\_is\_empty\_list()  
print(res)

# 13.test\_kw\_ratio\_with\_stoplist.py

**import** unittest  
**from** summa.keywords **import** keywords  
**from** summa.preprocessing.textcleaner **import** deaccent  
**from** test.test\_keywords **import** TestKeywords  
**'''  
test the ratio of generating keywords with stoplist  
'''**t = TestKeywords();  
res = t.test\_keywords\_ratio\_wstopwords()  
print(res)

# 14.test\_kw\_repeated\_words.py

**import** unittest  
**from** summa.keywords **import** keywords  
**from** summa.preprocessing.textcleaner **import** deaccent  
**from** test.test\_keywords **import** TestKeywords  
**'''  
test repeated words  
'''**t = TestKeywords();  
res = t.test\_repeated\_keywords()  
print(res)

# 15.test\_kw\_repeated\_words\_with\_stoplist.py

**import** unittest  
**from** summa.keywords **import** keywords  
**from** summa.preprocessing.textcleaner **import** deaccent  
**from** test.test\_keywords **import** TestKeywords  
**'''  
test repeated words with stoplist  
'''**t = TestKeywords();  
res = t.test\_repeated\_keywords\_wstopwords()  
print(res)

# 16.test\_kw\_spanish.py

**import** unittest  
**from** summa.keywords **import** keywords  
**from** summa.preprocessing.textcleaner **import** deaccent  
**from** test.test\_keywords **import** TestKeywords  
**'''  
test keywords in language = 'spanish' without accents  
'''**t = TestKeywords();  
res = t.test\_spanish\_without\_accents()  
print(res)

# 17.test\_kw\_spanish\_with\_accents.py

**import** unittest  
**from** summa.keywords **import** keywords  
**from** summa.preprocessing.textcleaner **import** deaccent  
**from** test.test\_keywords **import** TestKeywords  
**'''  
test keywords in language = 'spanish' with accents  
'''**t = TestKeywords();  
res = t.test\_spanish\_with\_accents()  
print(res)