webtopics

Introduction

The webtopics module is used to extract topics from webpages. The use is to be able to get the important information out of the webpage to get its crux.

Created By

Akshay Bhagdikar

Structure:

```
webtopics
|
|___UrlToText
|
|___TextToTopic
|
|___main
```

Prerequisites

The module has been written using Python 3.7

The existing modules used in this module are:

requests, BeautifulSoup, re, nltk, collections, math

Out of these re, collections, math are python standard libraries and the remaining ones can be installed using pip if not installed

Class UrlToText()

This class is used to fetch the textual content from HTML document.

Attributes

url : str —> The URL of the webpage of which the topics need to be found

_status_codes: dictionary —> The dictionary of common HTTP status codes as the key and their brief description as values
_body: bs4.element.Tag —> The content under body tag of the HTML

Methods

1) _get_soup()

This function returns the BeautifulSoup object based on the URL supplied. The class attribute url is used by this method

raises Exception if the status code sent by the server is not of class 2xxx

raises ConnectionError if you are not connected to the internet or there is some other connection issue

raises Timeout Error if the connection is timed out

raises RequestException if some ambiguous error happens

returns

2) _get_meta_content()

This function Returns the keywords/title/description content under the meta tag (element) of the HTML document. If no meta tags are present then it prints out that "No meta information is present

3) _get_title_content()
This function Returns the textual content under the title tag of the HTML document. If no Title tag is present then it prints out "No title tag found

4) _set_body()
This function Sets the class variable _body to the bs4.element.tag object that represents the body element of the HTML document

Raises Exception if body element is not found in the HTML document.
_filter_tags(body)

5) _filter_tags(body)

This function removes the script and style tags from the HTML structure since they do not give any useful textual information. This is one of the strategies to reduce the clutter.

6) _find_filter_threshold(self):

This function calculates the threshold of text density to be used in _filter_content function. The threshold is set as the the text density of the body element. The text density for a particular node is found by (Ci)/(Ti) where Ci is the number of textual character for a node and Ti is the number of nodes present under the i-th node. More the text density more important the node is. This is another strategy to reduce the clutter

7) _filter_content(self,node,min_char,threshold):

Parameters
node : bs4.element.tag —> element from HTML markup
min_char: int —> threshold for minimum length of textual content
threshold: float. —> threshold for text density

This function removes the HTML nodes that have text density lower than the threshold density the nodes that have less textual characters for eg. <li> tag. The threshold is set as the the text density of the body element. The text density for a particular node is found by (Ci)/(Ti) where Ci is the number of textual character for a node and Ti is the number of nodes present under the i-th node. The number of textual characters that represent the hyperlink are subtrcted from the total characters because hyperlink would most of the time not convey meaningful information. More the text density more important the node is. This is another strategy to reduce the clutter.

8) _get_body_content(self):

This function returns the textual content for all the nodes under the body element of the HTML document. The nodes are first checked for text density. If it is lower than threshold density then those nodes are not considered and their textual content is not taken into account.

9) get_total_content(self):

This function returns all the textual content under the body, meta and the title tag combined

class **TextToTopics**

This class is used to extract topics from textual content of the HTML document

Attributes

content : list of strings. —> Every element in this list is textual content from different nodes of HTML document

Methods

1) _get_pos(token)

This function returns a character depending on the part of the speech of the token passed to it. It returns 'n' if the part of speech is of type noun, 'v' if it is verb, 'r' if it is adverb, 'a' if it is adjective, 's' if it is satelite adjective, else it returns 'n'

Parameters
token : str —> token(word) from a sentence

2) _lemmatize_content()

This method returns the lemmatized form of every statement in the content attribute. It tokenizes every statement and then finds its part of speech and then passes it to the lemmatizer function. NLTK lemmatizer is used to lemmatize the tokens after which they are joined to form a statement.

3) _handle_period(sentence)

This mehtod replaces all the periods "." in acronyms with empty string "" . The regex chceks for structures like M.B.A and changes it to MBA

Parameters
sentence : str —> sentence from content

4) _handle_punctuation(self,sentence)

This method returns the statement with punctuations removed. Most of the punctuations are removed except for "'" as it is contained in certain words such as don't.

Parameters
sentence : str —> sentence from content

5) _get_n_grams(n)

This method constructs n-grams from the statements in the content.It takes the content attribute of the class and then removes period from acronym like words. Then it separates individual statements and then tokenizes them. We only keep the tokens that have part of speech as either noun, verb, adverb,adjective. The stop words are automatically removed as most of the stop words have part of speech other than the above mentioned. Then contiguous sets of tokens are extracted. For eg. given a sentence "Sun rises in the east and sets in the west" would be first changed to "sun rise east set west" and then n-grams tokens are formed for eg. 2grams such as "sun rise","rise east","east set","set west". It returns a list of such tokens.

Parameters
n : int. —> n in n-grams. Specifies how many contiguous word tokens need to be formed.

6) pim_based_topics(n,number=None)

This method extracts the topic from the content based on the pointwise mutual information score.Pointwise mutual information (PMI) or point mutual information, is a measure of association used in information theory and statistics. It basically measures the conditional probability of the two events occuring together. For eg. for 2grams it is given by PMI = p(w1,w2)/(p(w1)*p(w2))
p(w1,w2) = probability of two words occuring together in the content.
p(w1) = probability of word 1 occuring at place 1 and place 2 can have any word
p(w2) = probability of word 2 occuring at place 2 and place 1 can have any word
Higher the PMI higher the association and hence higher the chance of the ngram being a topic. However the downside is that if two words are rare and say occur only once then PMI score for such pair will be high even though it does not really talk about the content. So I have set the threshold frequency to 2. Although this seems very low threshold it worked for me for the test urls.

Parameters
n : int. —> n in n-grams. Specifies how many contiguous word tokens need to be formed.

number : int —> The top n-grams to be displayed

7) pos_based_topics(n,number)

This method extracts the topic from the content based on part of speech tags of the contiguous tokens. English language has a structure and consecutive words to make sense generally belong to particular part of speech.
For valid 2-grams the strucure has to be (noun,noun) or (Adjective,noun).
For valid 3-grams the structue has to be (adjective/noun, anything, adjective/noun)
The n-grams are filtered and then based on their frequency the topics are chosen. higher the frequency more the chances of the n-grams to be a topic. This approach can only take 2grams or 3grams tokens. Other value of n would raise an exception.

Parameters
n : int. —> n in n-grams. Specifies how many contiguous word tokens need to be formed.
number : int. —> The top n-grams to be displayed

Raises
ValueError if n is given value other than 2 or 3

Running instructions

Open the terminal and go to the folder webtopics

Run the command python main.py <url> --n <3> --approach <pos> —number <5>

Url = url of webpage
—n = number of n-grams for topic (optional)
—approach = specifies the approach for finding the topics either pos or him (optional)
—number = specifies the top n topics (optional)

Eg:

```
python main.py https://www.rei.com/blog/camp/how-to-introduce-your-indoorsy-friend-to-the-outdoors --n 3 --approach pos --number 5
```

Or

```
python main.py https://www.rei.com/blog/camp/how-to-introduce-your-indoorsy-friend-to-the-outdoors
```

Or

```
python main.py https://www.rei.com/blog/camp/how-to-introduce-your-indoorsy-friend-to-the-outdoors —n 3
```

Or

```
python main.py https://www.rei.com/blog/camp/how-to-introduce-your-indoorsy-friend-to-the-outdoors —n 3 —approach phi
```

Or

```
python main.py https://www.rei.com/blog/camp/how-to-introduce-your-indoorsy-friend-to-the-outdoors —approach pmi —number 5
```

And so on.