


Text Summarization with NLTK in Python

By  Usman Malik (https://twitter.com/usman_malikk) • September 04, 2018 •
7 Comments (/text-summarization-with-nltk-in-python/#disqus_thread)



Introduction

As I write this article, 1,907,223,370 websites are active on the internet and 2,722,460 emails are being sent per second. This is an unbelievably huge amount of data. It is impossible for a user to get insights from such huge volumes of data. Furthermore, a large portion of this data is either redundant or doesn't contain much useful information. The most efficient way to get access to the most important parts of the data, without having to sift through redundant and insignificant data, is to summarize the data in a way that it contains non-redundant and useful information only. The data can be in any form such as audio, video, images, and text. In this article, we will see how we can use automatic text summarization techniques to summarize text data.

Text summarization is a subdomain of Natural Language Processing (</what-is-natural-language-processing/>) (NLP) that deals with extracting summaries from huge chunks of texts. There are two main types of techniques used for text summarization: NLP-based

techniques and deep learning-based techniques. In this article, we will see a simple NLP-based technique for text summarization. We will not use any machine learning library in this article. Rather we will simply use Python's NLTK library (<https://www.nltk.org/>) for summarizing Wikipedia articles.

Text Summarization Steps

I will explain the steps involved in text summarization using NLP techniques with the help of an example.

The following is a paragraph from one of the famous speeches by Denzel Washington at the 48th NAACP Image Awards:

So, keep working. Keep striving. Never give up. Fall down seven times, get up eight. Ease is a greater threat to progress than hardship. Ease is a greater threat to progress than hardship. So, keep moving, keep growing, keep learning. See you at work.

We can see from the paragraph above that he is basically motivating others to work hard and never give up. To summarize the above paragraph using NLP-based techniques we need to follow a set of steps, which will be described in the following sections.

Convert Paragraphs to Sentences

We first need to convert the whole paragraph into sentences. The most common way of converting paragraphs to sentences is to split the paragraph whenever a period is encountered. So if we split the paragraph under discussion into sentences, we get the following sentences:

1. So, keep working
2. Keep striving
3. Never give up
4. Fall down seven times, get up eight
5. Ease is a greater threat to progress than hardship
6. Ease is a greater threat to progress than hardship
7. So, keep moving, keep growing, keep learning
8. See you at work

Text Preprocessing

After converting paragraph to sentences, we need to remove all the special characters, stop words and numbers from all the sentences. After preprocessing, we get the following sentences:

1. keep working
2. keep striving
3. never give
4. fall seven time get eight
5. ease greater threat progress hardship
6. ease greater threat progress hardship
7. keep moving keep growing keep learning
8. see work

Tokenizing the Sentences

We need to tokenize all the sentences to get all the words that exist in the sentences. After tokenizing the sentences, we get list of following words:

```
['keep',  
'working',  
'keep',  
'striving',  
'never',  
'give',  
'fall',  
'seven',  
'time',  
'get',  
'eight',  
'ease',  
'greater',  
'threat',  
'progress',  
'hardship',  
'ease',  
'greater',  
'threat',  
'progress',  
'hardship',  
'keep',  
'moving',  
'keep',  
'growing',  
'keep',  
'learning',  
'see',  
'work']
```

Find Weighted Frequency of Occurrence

Next we need to find the weighted frequency of occurrences of all the words. We can find the weighted frequency of each word by dividing its frequency by the frequency of the most occurring word. The following table contains the weighted frequencies for each word:

Word	Frequency	Weighted Frequency
ease	2	0.40
eight	1	0.20
fall	1	0.20
get	1	0.20
give	1	0.20
greater	2	0.40
growing	1	0.20
hardship	2	0.40
keep	5	1.00
learning	1	0.20
moving	1	0.20
never	1	0.20
progress	2	0.40
see	1	0.20
seven	1	0.20
striving	1	0.20
threat	2	0.40
time	1	0.20
work	1	0.20
working	1	0.20

Since the word "keep" has the highest frequency of 5, therefore the weighted frequency of all the words have been calculated by dividing their number of occurrences by 5.

Replace Words by Weighted Frequency in Original Sentences

The final step is to plug the weighted frequency in place of the corresponding words in original sentences and finding their sum. It is important to mention that weighted frequency for the words removed during preprocessing (stop words, punctuation, digits etc.) will be zero and therefore is not required to be added, as mentioned below:

Sentence	Sum of Weighted Frequencies
So, keep working	$1 + 0.20 = 1.20$
Keep striving	$1 + 0.20 = 1.20$
Never give up	$0.20 + 0.20 = 0.40$
Fall down seven times, get up eight	$0.20 + 0.20 + 0.20 + 0.20 + 0.20 = 1.0$
Ease is a greater threat to progress than hardship	$0.40 + 0.40 + 0.40 + 0.40 + 0.40 = 2.0$
Ease is a greater threat to progress than hardship	$0.40 + 0.40 + 0.40 + 0.40 + 0.40 = 2.0$
So, keep moving, keep growing, keep learning	$1 + 0.20 + 1 + 0.20 + 1 + 0.20 = 3.60$
See you at work	$0.20 + 0.20 = 0.40$

Sort Sentences in Descending Order of Sum

The final step is to sort the sentences in inverse order of their sum. The sentences with highest frequencies summarize the text. For instance, look at the sentence with the highest sum of weighted frequencies:

So, keep moving, keep growing, keep learning

You can easily judge that what the paragraph is all about. Similarly, you can add the sentence with the second highest sum of weighted frequencies to have a more informative summary. Take a look at the following sentences:

So, keep moving, keep growing, keep learning. Ease is a greater threat to progress than hardship.

These two sentences give a pretty good summarization of what was said in the paragraph.

Summarizing Wikipedia Articles

Now we know how the process of text summarization works using a very simple NLP technique. In this section, we will use Python's NLTK library to summarize a Wikipedia article.

Fetching Articles from Wikipedia

Before we could summarize Wikipedia articles, we need to fetch them from the web. To do so we will use a couple of libraries. The first library that we need to download is the beautiful soup (<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>) which is very useful Python utility for web scraping. Execute the following command at the command prompt to download the BeautifulSoup utility.

```
$ pip install beautifulsoup4
```

Another important library that we need to parse XML and HTML is the lxml (<https://lxml.de/>) library. Execute the following command at command prompt to download lxml:

```
$ pip install lxml
```

Now lets some Python code to scrape data from the web. The article we are going to scrape is the Wikipedia article on Artificial Intelligence (https://en.wikipedia.org/wiki/Artificial_intelligence). Execute the following script:

```
import bs4 as bs
import urllib.request
import re

scraped_data = urllib.request.urlopen('https://en.wikipedia.org/wiki/Artificial_intelligence')
article = scraped_data.read()

parsed_article = bs.BeautifulSoup(article,'lxml')

paragraphs = parsed_article.find_all('p')

article_text = ""

for p in paragraphs:
    article_text += p.text
```

In the script above we first import the important libraries required for scraping the data from the web. We then use the `urlopen` function from the `urllib.request` utility to scrape the data. Next, we need to call `read` function on the object returned by `urlopen` function in

order to read the data. To parse the data, we use `BeautifulSoup` object and pass it the scraped data object i.e. `article` and the `lxml` parser.

In Wikipedia articles, all the text for the article is enclosed inside the `<p>` tags. To retrieve the text we need to call `find_all` function on the object returned by the `BeautifulSoup`. The tag name is passed as a parameter to the function. The `find_all` function returns all the paragraphs in the article in the form of a list. All the paragraphs have been combined to recreate the article.

Once the article is scraped, we need to do some preprocessing.

Subscribe to our Newsletter

Get occasional tutorials, guides, and reviews in your inbox. No spam ever. Unsubscribe at any time.

Subscribe

Preprocessing

The first preprocessing step is to remove references from the article. Wikipedia, references are enclosed in square brackets. The following script removes the square brackets and replaces the resulting multiple spaces by a single space. Take a look at the script below:

```
# Removing Square Brackets and Extra Spaces
article_text = re.sub(r'\[[0-9]*\]', ' ', article_text)
article_text = re.sub(r'\s+', ' ', article_text)
```

The `article_text` object contains text without brackets. However, we do not want to remove anything else from the article since this is the original article. We will not remove other numbers, punctuation marks and special characters from this text since we will use this text to create summaries and weighted word frequencies will be replaced in this article.

To clean the text and calculate weighted frequencies, we will create another object. Take a look at the following script:

```
# Removing special characters and digits
formatted_article_text = re.sub('[^a-zA-Z]', ' ', article_text )
formatted_article_text = re.sub(r'\s+', ' ', formatted_article_text)
```

Now we have two objects `article_text` , which contains the original article and `formatted_article_text` which contains the formatted article. We will use `formatted_article_text` to create weighted frequency histograms for the words and will replace these weighted frequencies with the words in the `article_text` object.

Converting Text To Sentences

At this point we have preprocessed the data. Next, we need to tokenize the article into sentences. We will use the `article_text` object for tokenizing the article to sentence since it contains full stops. The `formatted_article_text` does not contain any punctuation and therefore cannot be converted into sentences using the full stop as a parameter.

The following script performs sentence tokenization:

```
sentence_list = nltk.sent_tokenize(article_text)
```

Find Weighted Frequency of Occurrence

To find the frequency of occurrence of each word, we use the `formatted_article_text` variable. We used this variable to find the frequency of occurrence since it doesn't contain punctuation, digits, or other special characters. Take a look at the following script:

```
stopwords = nltk.corpus.stopwords.words('english')

word_frequencies = {}
for word in nltk.word_tokenize(formatted_article_text):
    if word not in stopwords:
        if word not in word_frequencies.keys():
            word_frequencies[word] = 1
        else:
            word_frequencies[word] += 1
```

In the script above, we first store all the English stop words from the `nltk` library into a `stopwords` variable. Next, we loop through all the sentences and then corresponding words to first check if they are stop words. If not, we proceed to check whether the words exist in `word_frequency` dictionary i.e. `word_frequencies` , or not. If the word is encountered for the first time, it is added to the dictionary as a key and its value is set to 1. Otherwise, if the word previously exists in the dictionary, its value is simply updated by 1.

Finally, to find the weighted frequency, we can simply divide the number of occurrences of all the words by the frequency of the most occurring word, as shown below:

```
maximum_frequency = max(word_frequencies.values())

for word in word_frequencies.keys():
    word_frequencies[word] = (word_frequencies[word]/maximum_frequency)
```

Calculating Sentence Scores

We have now calculated the weighted frequencies for all the words. Now is the time to calculate the scores for each sentence by adding weighted frequencies of the words that occur in that particular sentence. The following script calculates sentence scores:

```
sentence_scores = {}
for sent in sentence_list:
    for word in nltk.word_tokenize(sent.lower()):
        if word in word_frequencies.keys():
            if len(sent.split(' ')) < 30:
                if sent not in sentence_scores.keys():
                    sentence_scores[sent] = word_frequencies[word]
                else:
                    sentence_scores[sent] += word_frequencies[word]
```

In the script above, we first create an empty `sentence_scores` dictionary. The keys of this dictionary will be the sentences themselves and the values will be the corresponding scores of the sentences. Next, we loop through each sentence in the `sentence_list` and tokenize the sentence into words.

We then check if the word exists in the `word_frequencies` dictionary. This check is performed since we created the `sentence_list` list from the `article_text` object; on the other hand, the word frequencies were calculated using the `formatted_article_text` object, which doesn't contain any stop words, numbers, etc.

We do not want very long sentences in the summary, therefore, we calculate the score for only sentences with less than 30 words (although you can tweak this parameter for your own use-case). Next, we check whether the sentence exists in the `sentence_scores` dictionary or not. If the sentence doesn't exist, we add it to the `sentence_scores` dictionary as a key and assign it the weighted frequency of the first word in the sentence, as its value. On the contrary, if the sentence exists in the dictionary, we simply add the weighted frequency of the word to the existing value.

Getting the Summary

Now we have the `sentence_scores` dictionary that contains sentences with their corresponding score. To summarize the article, we can take top N sentences with the highest scores. The following script retrieves top 7 sentences and prints them on the screen.

```
import heapq
summary_sentences = heapq.nlargest(7, sentence_scores, key=sentence_scores.get)

summary = ' '.join(summary_sentences)
print(summary)
```

In the script above, we use the `heapq` library and call its `nlargest` function to retrieve the top 7 sentences with the highest scores.

The output summary looks like this:

Artificial intelligence (AI), sometimes called machine intelligence, is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans and other animals. Many tools are used in AI, including versions of search and mathematical optimization, artificial neural networks, and methods based on statistics, probability and economics. The traditional problems (or goals) of AI research include reasoning, knowledge representation, planning, learning, natural language processing, perception and the ability to move and manipulate objects. When access to digital computers became possible in the middle 1950s, AI research began to explore the possibility that human intelligence could be reduced to symbol manipulation. One proposal to deal with this is to ensure that the first generally intelligent AI is 'Friendly AI', and will then be able to control subsequently developed AIs. Nowadays, the vast majority of current AI researchers work instead on tractable "narrow AI" applications (such as medical diagnosis or automobile navigation). Machine learning, a fundamental concept of AI research since the field's inception, is the study of computer algorithms that improve automatically through experience.

Remember, since Wikipedia articles are updated frequently, you might get different results depending upon the time of execution of the script.

Conclusion

This article explains the process of text summarization with the help of the Python NLTK library. The process of scraping articles using the BeautifulSoup library has also been briefly covered in the article. I will recommend you to scrape any other article from Wikipedia and see whether you can get a good summary of the article or not.

📁 [python \(/tag/python/\)](/tag/python/), [nlp \(/tag/nlp/\)](/tag/nlp/), [nltk \(/tag/nltk/\)](/tag/nltk/)

[https://twitter.com/share?](https://twitter.com/share?ext%20Summarization%20with%20NLTK%20in%20Python&url=https://stackabuse.com/text-azarization-with-nltk-in-python/)

[ext%20Summarization%20with%20NLTK%20in%20Python&url=https://stackabuse.com/text-azarization-with-nltk-in-python/](https://twitter.com/share?ext%20Summarization%20with%20NLTK%20in%20Python&url=https://stackabuse.com/text-azarization-with-nltk-in-python/)

<https://www.facebook.com/sharer/sharer.php?u=https://stackabuse.com/text-summarization-ltk-in-python/>

<https://plus.google.com/share?url=https://stackabuse.com/text-summarization-with-nltk-in-iv/>

[https://www.linkedin.com/shareArticle?mini=true%26url=https://stackabuse.com/text-arization-with-nltk-in-python/%26source=https://stackabuse.com\)](https://www.linkedin.com/shareArticle?mini=true%26url=https://stackabuse.com/text-arization-with-nltk-in-python/%26source=https://stackabuse.com))



[\(/author/usman/\)](/author/usman/)

About Usman Malik [\(/author/usman/\)](/author/usman/)

🏠 Paris (France) 🐦 Twitter (https://twitter.com/usman_malikk)

Programmer | Blogger | Data Science Enthusiast | PhD To Be | Arsenal FC for Life

Subscribe to our Newsletter

Get occasional tutorials, guides, and reviews in your inbox. No spam ever. Unsubscribe at any time.

Enter your email...

Subscribe

Recommend 3 Tweet Share

Sort by Best ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name



Sajid Hassan • 8 months ago

Hi Usman Malik,

I just copy and paste above complete code into a file but i could not get the summary.

I am facing following error

Please help me to get rid of this error.

```
<http.client.httpresponse object="" at="" 0x7fb4c7629940="">
Traceback (most recent call last):
File "/sajid/python_working_stuff/simple-summarizer/summarizer/summarizer-2.py", line 11, in <module>
  parsed_article = bs.BeautifulSoup(article, 'lxml')
File "/sajid/python_working_stuff/simple-summarizer/venv/lib/python3.6/site-packages/bs4/__init__.py", line 196, in __init__
  % ", ".join(features))
bs4.FeatureNotFound: Couldn't find a tree builder with the features you requested:
lxml. Do you need to install a parser library?
```

^ | ▾ • Reply • Share ▾



Educational psycho • 10 months ago

But I have already installed beautifulsoup4.

Package.<https://uploads.disquscdn.com/image...>

View

^ | ▾ • Reply • Share ▾



Educational psycho • 10 months ago

Hey, I have written this code but it shows an error something like this..... Can anyone suggest me that how I can resolve this problem. View –

uploads.disquscdn.com

^ | ▾ • Reply • Share ▾



Scott Mod → **Educational psycho** • 10 months ago

It looks like you don't have the beautifulsoup4 (which is imported as bs4) package installed on our system. Use pip to download it:

```
pip install beautifulsoup4
```

1 ^ | ▾ 1 • Reply • Share ▾



Educational psycho • 10 months ago

I have made a

.Py file and written this code but it shows syntax error in line- sentence_list = nltk.sent_tokenize(article_text) . Can anyone help me.

^ | ▾ • Reply • Share ▾



Stian ↗ Educational psycho • 10 months ago • edited

you have to add `import nltk` and probably you will also need

```
nltk.download('stopwords')
```

^ | v • Reply • Share ›



Bintah Thawood • a year ago • edited

Hi Malik,

Do you have any idea on how to understand mathematical word problems on Sets theory and identifying the universal set, sub sets with its value (in python)

^ | v • Reply • Share ›

◀ Previous Post : Beginner's Tutorial on the Pandas Python Library (/beginners-tutorial-on-the-pandas-python-library/)

Next Post : File Handling in Python > (/file-handling-in-python/)

Ad

Follow Us

🐦 Twitter (<https://twitter.com/StackAbuse>)

📘 Facebook (<https://www.facebook.com/stackabuse>)

📡 RSS (<https://stackabuse.com/rss/>)

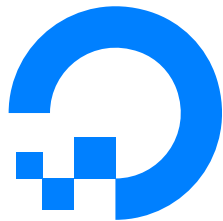
Newsletter

Subscribe to our newsletter! Get occasional tutorials, guides, and reviews in your inbox.

No spam ever. Unsubscribe at any time.

Ad

Our Sponsors



DigitalOcean

(<https://stackabu.se/digitalocean>)

The simplest cloud platform for developers and teams.

Learn More (<https://stackabu.se/digitalocean>)

Want a remote job?

Senior Python Engineer

ReCharge Payments a day ago (<https://hiredremote.io/remote-job/0821-senior-python-engineer-at-recharge-payments>)

(<https://hiredremote.io/remote-job/0821-senior-python-engineer-at-recharge-payments>) python

(<https://hiredremote.io/remote-python-jobs>) senior (<https://hiredremote.io/remote-senior-jobs>) java

(<https://hiredremote.io/remote-java-jobs>) ruby (<https://hiredremote.io/remote-ruby-jobs>)

Senior Software Engineer

Servosity, Inc. 2 days ago ([https://hiredremote.io/remote-job/0813-senior-software-engineer-at-servosity-inc.](https://hiredremote.io/remote-job/0813-senior-software-engineer-at-servosity-inc))

([https://hiredremote.io/remote-job/0813-senior-software-engineer-at-servosity-inc.](https://hiredremote.io/remote-job/0813-senior-software-engineer-at-servosity-inc)) [c](#)

(<https://hiredremote.io/remote-c-jobs>) [cpp](#) (<https://hiredremote.io/remote-cpp-jobs>) [drivers](#)

(<https://hiredremote.io/remote-drivers-jobs>) [operating-system](#) (<https://hiredremote.io/remote-operating-system-jobs>)

Senior React Developer

KNØX 2 days ago (<https://hiredremote.io/remote-job/0814-senior-react-developer-at-knox>)

(<https://hiredremote.io/remote-job/0814-senior-react-developer-at-knox>) [react-js](#)

(<https://hiredremote.io/remote-react-js-jobs>) [typescript](#) (<https://hiredremote.io/remote-typescript-jobs>)

[javascript](#) (<https://hiredremote.io/remote-javascript-jobs>) [senior](#) (<https://hiredremote.io/remote-senior-jobs>)

[🔗 More jobs \(<https://hiredremote.io>\)](#)

Jobs via HireRemote.io (<https://hiredremote.io>)

Recent Posts

Deploying Node.js Apps to AWS EC2 with Docker (</deploying-node-js-apps-to-aws-ec2-with-docker/>)

Executing Shell Commands with Node.js (</executing-shell-commands-with-node-js/>)

Creational Design Patterns in Python (</creational-design-patterns-in-python/>)

Tags

[ai \(/tag/ai/\)](/tag/ai/)

[algorithms \(/tag/algorithms/\)](/tag/algorithms/)

[amqp \(/tag/amqp/\)](/tag/amqp/)

[angular \(/tag/angular/\)](/tag/angular/)

[announcements \(/tag/announcements/\)](/tag/announcements/)

[apache \(/tag/apache/\)](/tag/apache/)

[api \(/tag/api/\)](/tag/api/)

[arduino \(/tag/arduino/\)](/tag/arduino/)

[artificial intelligence \(/tag/artificial-intelligence/\)](/tag/artificial-intelligence/)

[asynchronous \(/tag/asynchronous/\)](/tag/asynchronous/)

Follow Us

[Twitter \(https://twitter.com/StackAbuse\)](https://twitter.com/StackAbuse)

[Facebook \(https://www.facebook.com/stackabuse\)](https://www.facebook.com/stackabuse)

[RSS \(https://stackabuse.com/rss/\)](https://stackabuse.com/rss/)

Copyright © 2019, Stack Abuse (<https://stackabuse.com>). All Rights Reserved.

[Disclosure \(/disclosure\)](/disclosure) • [Privacy Policy \(/privacy-policy\)](/privacy-policy) • [Terms of Service \(/terms-of-service\)](/terms-of-service)