文字探勘
(Text Mining)

處理和理解文本
(Processing and Understanding Text)

Chichang Jou
Associate Professor
周清江
副教授
cjou@mail.tku.edu.tw

Min-Yuh Day
Associate Professor
戴敏育
副教授
myday@mail.tku.edu.tw

Dept. of Information Management, Tamkang University
<table>
<thead>
<tr>
<th>週次 (Week)</th>
<th>日期 (Date)</th>
<th>內容 (Subject/Topics)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2020/03/02</td>
<td>文字探勘課程介紹 (Course Orientation on Text Mining)</td>
</tr>
<tr>
<td>2</td>
<td>2020/03/09</td>
<td>文字探勘基礎：自然語言處理 (Foundations of Text Mining: Natural Language Processing; NLP)</td>
</tr>
<tr>
<td>3</td>
<td>2020/03/16</td>
<td>Python自然語言處理 (Python for Natural Language Processing)</td>
</tr>
<tr>
<td>4</td>
<td>2020/03/23</td>
<td>處理和理解文本 (Processing and Understanding Text)</td>
</tr>
<tr>
<td>5</td>
<td>2020/03/30</td>
<td>文本表達特徵工程 (Feature Engineering for Text Representation)</td>
</tr>
<tr>
<td>6</td>
<td>2020/04/06</td>
<td>人工智慧文本分析個案研究 I (Case Study on Artificial Intelligence for Text Analytics I)</td>
</tr>
</tbody>
</table>
課程大綱 (Syllabus)

週次 (Week) 日期 (Date) 內容 (Subject/Topics)
7 2020/04/13 文本分類
   (Text Classification)
8 2020/04/20 文本摘要和主題模型
   (Text Summarization and Topic Models)
9 2020/04/27 期中報告 (Midterm Project Report)
10 2020/05/04 文本相似度和分群
    (Text Similarity and Clustering)
11 2020/05/11 語意分析和命名實體識別
    (Semantic Analysis and Named Entity Recognition; NER)
12 2020/05/18 情感分析
    (Sentiment Analysis)
<table>
<thead>
<tr>
<th>週次 (Week)</th>
<th>日期 (Date)</th>
<th>內容 (Subject/Topics)</th>
</tr>
</thead>
</table>
| 13         | 2020/05/25  | 人工智慧文本分析個案研究Ⅱ  
(Case Study on Artificial Intelligence for Text Analytics Ⅱ) |
| 14         | 2020/06/01  | 深度學習和通用句子嵌入模型  
(Deep Learning and Universal Sentence-Embedding Models) |
| 15         | 2020/06/08  | 問答系統與對話系統  
(Question Answering and Dialogue Systems) |
| 16         | 2020/06/15  | 期末報告Ⅰ (Final Project Presentation I) |
| 17         | 2020/06/22  | 期末報告Ⅱ (Final Project Presentation Ⅱ) |
| 18         | 2020/06/29  | 教師彈性補充教學 |
Outline

• Processing Text
• Understanding Text
Processing and Understanding Text
Free eBooks - Project Gutenberg

Some of the Latest eBooks

Welcome

New website available for testing. Visit https://dev.gutenberg.org (or http://dev.gutenberg.org) to test the site (it may have occasional outages, as improvements are made). There is a new website page that lists some known issues, and part of the motivation for the change. If you visit the new website, please consider providing your input and suggestions via an anonymous online survey afterwards.

Project Gutenberg is a library of over 60,000 free eBooks. Choose among free epub and Kindle eBooks, download them or read them online. You will find the world's great literature here, with focus on older works for which U.S. copyright has expired. Thousands of volunteers digitized and diligently proofread the eBooks, for enjoyment and education.

No fee or registration! Everything from Project Gutenberg is gratis, libre, and completely without cost to readers. If you find Project Gutenberg useful, please consider a small donation, to help Project Gutenberg digitize more books, maintain our online presence, and improve Project Gutenberg programs and offerings. Other ways to help include digitizing, proofreading and formatting, recording audio books, or reporting errors.
The Project Gutenberg Ebook of Alice’s Adventures in Wonderland, by Lewis Carroll

This ebook is for the use of anyone anywhere at no cost and with almost no restrictions whatsoever. You may copy it, give it away or re-use it under the terms of the Project Gutenberg License included with this ebook or online at www.gutenberg.org

Title: Alice’s Adventures in Wonderland

Author: Lewis Carroll

Release Date: June 25, 2008 [EBook #11]
Last Updated: February 22, 2020

Language: English

Character set encoding: UTF-8

*** START OF THIS PROJECT GUTENBERG EBOOK ALICE’S ADVENTURES IN WONDERLAND ***

Produced by Arthur DiBianca and David Widger
Alice Top 50 Tokens

50 most common tokens (no stopwords or punctuation)

https://tinyurl.com/imtkupython101
Python in Google Colab (Python101)

https://colab.research.google.com/drive/1FEG6DnGvwfUbeo4zJ1zTunjMqf2RkCrT

```python
nltk.download('gutenberg')
alice = Text(nltk.corpus.gutenberg.words('carroll-alice.txt'))
```

Text Processing and Understanding

- NLTK (Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit) Book: https://www.nltk.org/book/

```python
[ ] 1!pip install nltk
2 import nltk
3 nltk.download('gutenberg')

Requirement already satisfied: nltk in /usr/local/lib/python3.6/dist-packages (3.2.5)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from nltk) (1.12.0)
[nltk_data] Downloading package gutenberg to /root/nltk_data...
True
```

```python
[ ] 1 from nltk.text import Text
2 alice = Text(nltk.corpus.gutenberg.words('carroll-alice.txt'))
3 alice

<Text: Alice ’s Adventures in Wonderland by Lewis Carroll 1865>
```

```python
[ ] 1 print(nltk.corpus.gutenberg.fileids())

Displaying 25 of 398 matches:

Alice's Adventures in Wonderland by Lewis Carroll

[ CHAPTER I. ] Down the Rabbit-Hole

Alice was beginning to get very tired of sitting by the fire and \_ what is the use of a book,\_ thought Alice \_ without pictures or conversation? \_ so very remarkable in that; nor did Alice think it so very much out of the way looked at it, and then hurried on; Alice started to her feet, for it flashed hedge. In another moment down went Alice after it, never once considering ho ped suddenly down, so suddenly that Alice had not a moment to think about stop she fell past it. 'Well!' thought Alice to herself, 'after such a fall as down, I think --' (for, you see, Alice had learnt several things of this so tude or Longitude I've got to?' (Alice had no idea what Latitude was, or L. There was nothing else to do, so Alice soon began talking again. 'Dinah, cats eat bats, I wonder?' And here Alice began to get rather sleepy, and wen dry leaves, and the fall was over. Alice was not a bit hurt, and she jumped not a moment to be lost: away went Alice like the wind, and was just in time but they were all locked; and when Alice had been all the way down one side a on it except a tiny golden key, and Alice's first thought was that it might and to her great delight it fitted! Alice opened the door and found that it le ead would go through,' thought poor Alice, 'it would be of very little use w ay things had happened lately, that Alice had begun to think that very few thi ertainly was not here before,' said Alice,) and round the neck of the bottle ay 'Drink me,' but the wise little Alice was not going to do THAT in a hurry bottle was NOT marked 'poison,' so Alice ventured to taste it, and finding it * * 'What a curious feeling!' said Alice; 'I must be shutting up like a tel for it might end, you know,' said Alice to herself, 'in my going out altog garden at once; but, alas for poor Alice! when she got to the door, she fou

https://tinyurl.com/imtkupython101
```python
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
alice.dispersion_plot(['Alice', 'Rabbit', 'Hatter', 'Queen'])
```
```python
# import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
fdist = nltk.FreqDist(alice)
fdist.plot(50)
```

https://tinyurl.com/imtkupython101
for word, freq in fdist.items()
if word.isalpha()

```python
# import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
fdist_no_punc = nltk.FreqDist(dict((word, freq) for word, freq in fdist.items() if word.isalpha()))
fdist_no_punc.plot(50, cumulative=False, title="50 most common tokens (no punctuation)")
```

https://tinyurl.com/imtkupyter101
```python
import nltk
nltk.download('stopwords')
stopwords = nltk.corpus.stopwords.words('english')

'same',
'so',
'than',
'too',
'very',
's',
't',
'can',
'will',
'just',
'don',
'don\'t',
'should',
'should\'ve',
'now',
```

https://tinyurl.com/imtkupython101
for word, freq in fdist.items():
    if word not in stopwords and word.isalpha():

1 import matplotlib.pyplot as plt
2 plt.figure(figsize=(10, 6))
3 fdist_no_punc_no_stopwords = nltk.FreqDist(dict((word, freq) for word, freq in fdist.items() if word not in stopwords and word.isalpha()))
4 fdist_no_punc_no_stopwords.plot(50, cumulative=False, title="50 most common tokens (no stopwords or punctuation)")

https://tinyurl.com/imtkupython101
Alice Top 50 Tokens

50 most common tokens (no stopwords or punctuation)

https://tinyurl.com/imtkuppython101
import requests
from bs4 import BeautifulSoup

url = 'https://www.gutenberg.org/files/11/11-h/11-h.htm'
reqs = requests.get(url)
html_doc = reqs.text

soup = BeautifulSoup(html_doc, 'html.parser')
text = soup.get_text()

https://tinyurl.com/imtkupython101
```python
from tensorflow.keras.preprocessing.text import Tokenizer

sentences = [
    'i love my dog',
    'I, love my cat',
    'You love my dog!'
]

tokenizer = Tokenizer(num_words=100)
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index
print('sentences:', sentences)
print('word index:', word_index)
```

```
sentences: ['i love my dog', 'I, love my cat', 'You love my dog!']
word index: {'love': 1, 'my': 2, 'i': 3, 'dog': 4, 'cat': 5, 'you': 6}
```
```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

sentences = [
'I love my dog',
'I love my cat',
'You love my dog!',
'Do you think my dog is amazing?'
]

tokenizer = Tokenizer(num_words=100, oov_token='<OOV>')
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(sentences)
padded = pad_sequences(sequences, maxlen=5)
print("sentences = ", sentences)
print("Word Index = ", word_index)
print("Sequences = ", sequences)
print("Padded Sequences:")
print(padded)
```

https://tinyurl.com/imtkupython101
import tensorflow.keras.preprocessing.sequence
import pad_sequences

sentences = ['I love my dog', 'I love my cat', 'You love my dog!', 'Do you think my dog is amazing?']

Word Index = {'<OOV>': 1, 'my': 2, 'love': 3, 'dog': 4, 'i': 5, 'you': 6, 'cat': 7, 'do': 8, 'think': 9, 'is': 10, 'amazing': 11}

Sequences = [[5, 3, 2, 4], [5, 3, 2, 7], [6, 3, 2, 4], [8, 6, 9, 2, 4, 10, 11]]

Padded Sequences: [[ 0 5 3 2 4] [ 0 5 3 2 7] [ 0 6 3 2 4] [ 9 2 4 10 11]]
Python in Google Colab

https://colab.research.google.com/drive/1FEG6DnGvwfUbeo4zJ1zTunjMqf2RkCrT

Keras preprocessing text

```python
# keras.preprocessing.text Tokenizer
from keras.preprocessing.text import Tokenizer
# define 5 documents
docs = ['Well done!', 'Good work', 'Great effort', 'nice work', 'Excellent!']
# create the tokenizer
t = Tokenizer(num_words=10)
# fit the tokenizer on the documents
t.fit_on_texts(docs)
print('docs:', docs)
print('word_counts:', t.word_counts)
print('document_count:', t.document_count)
print('word_index:', t.word_index)
print('word_docs:', t.word_docs)
# integer encode documents
texts_to_matrix = t.texts_to_matrix(docs, mode='count')
print('texts_to_matrix:')
print(texts_to_matrix)
```

Using TensorFlow backend.
docs: ['Well done!', 'Good work', 'Great effort', 'nice work', 'Excellent!']
word_counts: OrderedDict([('well', 1), ('done', 1), ('good', 1), ('work', 2), ('great', 1), ('effort', 1), ('nice', 1), ('excellent', 1)]
document_count: 5
word_index: {'work': 1, 'well': 2, 'done': 3, 'good': 4, 'great': 5, 'effort': 6, 'nice': 7, 'excellent': 8}
word_docs: [('done', 1), ('well', 1), ('work', 2), ('good', 1), ('great', 1), ('effort', 1), ('nice', 1), ('excellent', 1)]
texts_to_matrix:
[[0.0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1.]]

https://tinyurl.com/imtkupython101
# One-hot encoding

The mouse ran up the clock' =

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>[0, 1, 0, 0, 0, 0, 0, 0],</th>
<th>[0, 0, 1, 0, 0, 0, 0, 0],</th>
<th>[0, 0, 0, 1, 0, 0, 0, 0],</th>
<th>[0, 0, 0, 0, 1, 0, 0, 0],</th>
<th>[0, 0, 0, 0, 0, 1, 0, 0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>The</td>
<td>1</td>
<td>[0, 1, 0, 0, 0, 0, 0, 0]</td>
<td>[0, 0, 1, 0, 0, 0, 0, 0]</td>
<td>[0, 0, 0, 1, 0, 0, 0, 0]</td>
<td>[0, 0, 0, 0, 1, 0, 0, 0]</td>
<td>[0, 0, 0, 0, 0, 1, 0, 0]</td>
</tr>
<tr>
<td>mouse</td>
<td>2</td>
<td>[0, 0, 1, 0, 0, 0, 0, 0]</td>
<td>[0, 0, 0, 1, 0, 0, 0, 0]</td>
<td>[0, 0, 0, 0, 1, 0, 0, 0]</td>
<td>[0, 0, 0, 0, 0, 1, 0, 0]</td>
<td>[0, 0, 0, 0, 0, 0, 1, 0]</td>
</tr>
<tr>
<td>ran</td>
<td>3</td>
<td>[0, 0, 0, 1, 0, 0, 0, 0]</td>
<td>[0, 0, 0, 0, 1, 0, 0, 0]</td>
<td>[0, 0, 0, 0, 0, 1, 0, 0]</td>
<td>[0, 0, 0, 0, 0, 0, 1, 0]</td>
<td>[0, 0, 0, 0, 0, 0, 0, 1]</td>
</tr>
<tr>
<td>up</td>
<td>4</td>
<td>[0, 0, 0, 0, 1, 0, 0, 0]</td>
<td>[0, 0, 0, 0, 0, 1, 0, 0]</td>
<td>[0, 0, 0, 0, 0, 0, 1, 0]</td>
<td>[0, 0, 0, 0, 0, 0, 0, 1]</td>
<td>[0, 0, 0, 0, 0, 0, 0, 0]</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
<td>[0, 1, 0, 0, 0, 0, 0, 0]</td>
<td>[0, 0, 1, 0, 0, 0, 0, 0]</td>
<td>[0, 0, 0, 1, 0, 0, 0, 0]</td>
<td>[0, 0, 0, 0, 1, 0, 0, 0]</td>
<td>[0, 0, 0, 0, 0, 1, 0, 0]</td>
</tr>
<tr>
<td>clock</td>
<td>5</td>
<td>[0, 0, 0, 0, 0, 0, 0, 1]</td>
<td>[0, 0, 0, 0, 0, 0, 0, 1]</td>
<td>[0, 0, 0, 0, 0, 0, 0, 1]</td>
<td>[0, 0, 0, 0, 0, 0, 0, 1]</td>
<td>[0, 0, 0, 0, 0, 0, 0, 1]</td>
</tr>
</tbody>
</table>

[0, 1, 2, 3, 4, 5, 6]
Word embeddings

Male-Female

Verb Tense

Country-Capital

Source: https://developers.google.com/machine-learning/guides/text-classification/step-3
Word embeddings

The mouse ran up the clock
- the: 1
- mouse: 2
- ran: 3
- up: 4
- clock: 5

The mouse ran down
- mouse: 2
- ran: 3
- down: 6

[1, 2, 3, 4, 1, 5]

Embedding layer (output dim = 4)

[[0.236, -0.141, 0.000, 0.045],
 [0.006, 0.652, 0.270, -0.556],
 [0.305, 0.569, -0.028, 0.496],
 [0.421, 0.195, -0.058, 0.477],
 [0.236, -0.141, 0.000, 0.045],
 [0.844, -0.001, 0.763, 0.201]]
t1 = 'The mouse ran up the clock'
t2 = 'The mouse ran down'
s1 = t1.lower().split(' ')
s2 = t2.lower().split(' ')
terms = s1 + s2
sortedset = sorted(set(terms))
print('terms =', terms)
print('sortedset =', sortedset)
```python
t1 = 'The mouse ran up the clock'
t2 = 'The mouse ran down'
s1 = t1.lower().split(' ')
s2 = t2.lower().split(' ')
terms = s1 + s2
print(terms)

tfdict = {}
for term in terms:
    if term not in tfdict:
        tfdict[term] = 1
    else:
        tfdict[term] += 1

a = []
for k,v in tfdict.items():
    a.append('{} = {}'.format(k,v))
print(a)
```

```
['the', 'mouse', 'ran', 'up', 'the', 'clock', 'the', 'mouse', 'ran', 'down']
['the', 3, 'mouse', 2, 'ran', 2, 'up', 1, 'clock', 1, 'down', 1]
```
sorted_by_value_reverse = sorted(tfdict.items(), key=lambda kv: kv[1], reverse=True)

sorted_by_value_reverse_dict = dict(sorted_by_value_reverse)

id2word = {id: word for id, word in enumerate(sorted_by_value_reverse_dict)}

word2id = dict([(v, k) for (k, v) in id2word.items()])

sorted_by_value: [('up', 1), ('clock', 1), ('down', 1), ('mouse', 2), ('ran', 2), ('the', 3)]
sorted_by_value2: ['the', 'mouse', 'ran', 'up', 'clock', 'down']
sorted_by_value_reverse: [('the', 3), ('mouse', 2), ('ran', 2), ('up', 1), ('clock', 1), ('down', 1)]
sorted_by_value_reverse_dict {'the': 3, 'mouse': 2, 'ran': 2, 'up': 1, 'clock': 1, 'down': 1}
id2word {0: 'the', 1: 'mouse', 2: 'ran', 3: 'up', 4: 'clock', 5: 'down'}
word2id {'the': 0, 'mouse': 1, 'ran': 2, 'up': 3, 'clock': 4, 'down': 5}
len_words: 6
sorted_by_key: [('clock', 1), ('down', 1), ('mouse', 2), ('ran', 2), ('the', 3), ('up', 1)]
the, 3
mouse, 2
ran, 2
up, 1
clock, 1
down, 1

https://colab.research.google.com/drive/1FEG6DnGvwfUbeo4zJ1zTunjMqf2RkCrT
sorted_by_value = sorted(tfdict.items(), key=lambda kv: kv[1])
print('sorted_by_value: ', sorted_by_value)
sorted_by_value2 = sorted(tfdict, key=tfdict.get, reverse=True)
print('sorted_by_value2: ', sorted_by_value2)
sorted_by_value_reverse = sorted(tfdict.items(), key=lambda kv: kv[1], reverse=True)
print('sorted_by_value_reverse: ', sorted_by_value_reverse)
sorted_by_value_reverse_dict = dict(sorted_by_value_reverse)
print('sorted_by_value_reverse_dict', sorted_by_value_reverse_dict)
id2word = {id: word for id, word in enumerate(sorted_by_value_reverse_dict)}
print('id2word', id2word)
word2id = dict([(v, k) for (k, v) in id2word.items()])
print('word2id', word2id)
print('len_words', len(word2id))
sorted_by_key = sorted(tfdict.items(), key=lambda kv: kv[0])
print('sorted_by_key: ', sorted_by_key)

tfstring = \n'.join(a)
print(tfstring)
tf = tfdict.get('mouse')
print(tf)

sorted_by_value: [{'up', 1}, {'clock', 1}, {'down', 1}, {'mouse', 2}, {'ran', 2}, {'the', 3}]
sorted_by_value2: [{'the', 'mouse', 'ran', 'up', 'clock', 'down'}]
sorted_by_value_reverse: [{'the', 3}, {'mouse', 2}, {'ran', 2}, {'up', 1}, {'clock', 1}, {'down', 1}]
sorted_by_value_reverse_dict {'the': 3, 'mouse': 2, 'ran': 2, 'up': 1, 'clock': 1, 'down': 1}
id2word {0: 'the', 1: 'mouse', 2: 'ran', 3: 'up', 4: 'clock', 5: 'down'}
word2id {'the': 0, 'mouse': 1, 'ran': 2, 'up': 3, 'clock': 4, 'down': 5}
len_words: 6
sorted_by_key: [{'clock', 1}, {'down', 1}, {'mouse', 2}, {'ran', 2}, {'the', 3}, {'up', 1}]
the, 3
mouse, 2
ran, 2
up, 1
clock, 1
down, 1

https://colab.research.google.com/drive/1FEG6DnGvUwFUBeo4zJ1zTunjMqf2RkCrT
from keras.preprocessing.text import Tokenizer

define 5 documents
docs = ['Well done!', 'Good work', 'Great effort', 'nice work', 'Excellent!']
# create the tokenizer
t = Tokenizer()
# fit the tokenizer on the documents
t.fit_on_texts(docs)
print('docs:', docs)
print('word_counts:', t.word_counts)
print('document_count:', t.document_count)
print('word_index:', t.word_index)
print('word_docs:', t.word_docs)

# integer encode documents
texts_to_matrix = t.texts_to_matrix(docs, mode='count')
print('texts_to_matrix:')
print(texts_to_matrix)
from keras.preprocessing.text import Tokenizer

# define 5 documents
docs = ['Well done!', 'Good work', 'Great effort', 'nice work', 'Excellent!']
# create the tokenizer
t = Tokenizer()
# fit the tokenizer on the documents
t.fit_on_texts(docs)
print('docs:', docs)
print('word_counts:', t.word_counts)
print('document_count:', t.document_count)
print('word_index:', t.word_index)
print('word_docs:', t.word_docs)
# integer encode documents
texts_to_matrix = t.texts_to_matrix(docs, mode='count')
print('texts_to_matrix:')
print(texts_to_matrix)
```python
texts_to_matrix = t.texts_to_matrix(docs, mode='count')
docs: ['Well done!', 'Good work', 'Great effort', 'nice work', 'Excellent!']
word_counts: OrderedDict([('well', 1), ('done', 1), ('good', 1), ('work', 2), ('great', 1), ('effort', 1), ('nice', 1), ('excellent', 1)])
document_count: 5
word_index: {'work': 1, 'well': 2, 'done': 3, 'good': 4, 'great': 5, 'effort': 6, 'nice': 7, 'excellent': 8}
word_docs: {'done': 1, 'well': 1, 'work': 2, 'good': 1, 'great': 1, 'effort': 1, 'nice': 1, 'excellent': 1}
texts_to_matrix:
[[0. 0. 1. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 1. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1.]]
```

Source: https://machinelearningmastery.com/prepare-text-data-deep-learning-keras/
from keras.preprocessing.text import Tokenizer

# define 5 documents
docs = ['Well done!', 'Good work', 'Great effort', 'nice work', 'Excellent!']

# create the tokenizer

t = Tokenizer()

# fit the tokenizer on the documents

t.fit_on_texts(docs)

print('docs:', docs)

print('word_counts:', t.word_counts)

print('document_count:', t.document_count)

print('word_index:', t.word_index)

print('word_docs:', t.word_docs)

# integer encode documents

texts_to_matrix = t.texts_to_matrix(docs, mode='tfidf')

print('texts_to_matrix: ')

print(texts_to_matrix)

Source: https://machinelearningmastery.com/prepare-text-data-deep-learning-keras/
Summary

• Processing Text
• Understanding Text
References

• NLTK (Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit) Book: https://www.nltk.org/book/
• Laurence Moroney (2020), Natural Language Processing - Tokenization (NLP Zero to Hero, part 1), https://www.youtube.com/watch?v=fNxaJsNG3-s
• Laurence Moroney (2020), Natural Language Processing - Sequencing - Turning sentence into data (NLP Zero to Hero, part 2), https://www.youtube.com/watch?v=r9QjkdSJZ2g
• Laurence Moroney (2020), Natural Language Processing - Training a model to recognize sentiment in text (NLP Zero to Hero, part 3), https://www.youtube.com/watch?v=Y_hzMnRXjhI