(AI for Text Analytics)

Python 自然語言處理
(Python for Natural Language Processing)

Min-Yuh Day
Associate Professor

Institute of Information Management, National Taipei University

https://web.ntpu.edu.tw/~myday

2020-10-08
課程大綱 (Syllabus)

週次 (Week) 日期 (Date) 内容 (Subject/Topics)
1 2020/09/17 人工智慧文本分析課程介紹  (Course Orientation on Artificial Intelligence for Text Analytics)
2 2020/09/24 文本分析的基礎：自然語言處理 (Foundations of Text Analytics: Natural Language Processing; NLP)
3 2020/10/01 中秋節 (Mid-Autumn Festival) 放假一天 (Day off)
4 2020/10/08 Python自然語言處理 (Python for Natural Language Processing)
5 2020/10/15 處理和理解文本 (Processing and Understanding Text)
6 2020/10/22 文本表達特徵工程 (Feature Engineering for Text Representation)
課程大綱 (Syllabus)

週次 (Week) 日期 (Date) 內容 (Subject/Topics)
7 2020/10/29 人工智慧文本分析個案研究 I
(Case Study on Artificial Intelligence for Text Analytics I)
8 2020/11/05 文本分類
(Text Classification)
9 2020/11/12 文本摘要和主題模型
(Text Summarization and Topic Models)
10 2020/11/19 期中報告 (Midterm Project Report)
11 2020/11/26 文本相似度和分群
(Text Similarity and Clustering)
12 2020/12/03 語意分析和命名實體識別
(Semantic Analysis and Named Entity Recognition; NER)
課程大綱 (Syllabus)

週次 (Week)   日期 (Date)  內容 (Subject/Topics)
13 2020/12/10 情感分析
(Sentiment Analysis)
14 2020/12/17 人工智慧文本分析個案研究 II
(Case Study on Artificial Intelligence for Text Analytics II)
15 2020/12/24 深度學習和通用句子嵌入模型
(Deep Learning and Universal Sentence-Embedding Models)
16 2020/12/31 問答系統與對話系統
(Question Answering and Dialogue Systems)
17 2021/01/07 期末報告 I (Final Project Presentation I)
18 2021/01/14 期末報告 II (Final Project Presentation II)
Outline

• Python for Natural Language Processing
Python for Natural Language Processing
Connect Google Colab in Google Drive
Google Colab
Google Colab

Colaboratory
offered by https://colab.research.google.com
A data analysis tool that combines code, output, and descriptive text into one collaborative document.
Connect Colaboratory to Google Drive

Colaboratory was connected to Google Drive.

Make Colaboratory the default app for files it can open

OK
Google Colab
Google Colab
Run Jupyter Notebook
Python3 GPU
Google Colab
Google Colab Python Hello World

```
print('Hello World')
```
Natural Language Processing (NLP) and Text Mining

- Raw text
- Sentence Segmentation
- Tokenization
- Part-of-Speech (POS)
- Stop word removal
- Stemming / Lemmatization
- Dependency Parser
- String Metrics & Matching

Word’s stem: am → am having → hav
Word’s lemma: am → be having → have

Source: Nitin Hardeniya (2015), NLTK Essentials, Packt Publishing; Florian Leitner (2015), Text mining - from Bayes rule to dependency parsing
spaCy:
Natural Language Processing

Industrial-Strength Natural Language Processing

Get things done
spaCy is designed to help you do real work — to build real products, or gather real insights. The library respects your time, and tries to avoid wasting it. It’s easy to install, and its API is simple and productive. We like to think of spaCy as the Ruby on Rails of Natural Language Processing.

Blazing fast
spaCy excels at large-scale information extraction tasks. It's written from the ground up in carefully memory-managed Cython. Independent research in 2015 found spaCy to be the fastest in the world. If your application needs to process entire web dumps, spaCy is the library you want to be using.

Deep learning
spaCy is the best way to prepare text for deep learning. It interoperates seamlessly with TensorFlow, PyTorch, scikit-learn, Gensim and the rest of Python's awesome AI ecosystem. With spaCy, you can easily construct linguistically sophisticated statistical models for a variety of NLP problems.

https://spacy.io/
Python in Google Colab (Python101)

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

https://tinyurl.com/aintpuppython101
Python in Google Colab (Python101)

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

Text Analytics and Natural Language Processing (NLP)

Python for Natural Language Processing

- spaCy Chinese Model
- Open Chinese Convert (OpenCC, 開放中文轉換)
- Jieba 結巴中文分詞
- Natural Language Toolkit (NLTK)
- Stanza: A Python NLP Library for Many Human Languages

Text Processing and Understanding

- NLTK (Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit)
- NLP Zero to Hero
  - Natural Language Processing - Tokenization (NLP Zero to Hero, part 1)
  - Natural Language Processing - Sequencing - Turning sentence into data (NLP Zero to Hero, part 2)
  - Natural Language Processing - Training a model to recognize sentiment in text (NLP Zero to Hero, part 3)

Python for Natural Language Processing

**spaCy**

- spaCy: Industrial-Strength Natural Language Processing in Python
- Source: [https://spacy.io/usage/spacy-101](https://spacy.io/usage/spacy-101)

```
[1] 1 !python -m spacy download en_core_web_sm

[3] 1 import spacy
  2 nlp = spacy.load("en_core_web_sm")
  3 doc = nlp("Apple is looking at buying U.K. startup for $1 billion")
  4 for token in doc:
      5     print(token.text, token.pos_, token.dep_)
```

Apple PROPN nsubj
is AUX aux
looking VERB ROOT
at ADP prep
buying VERB pcomp
U.K. PROPN compound
startup NOUN dobj
for ADP prep
$ SYM quantmod
1 NUM compound
billion NUM pobj

https://tinyurl.com/aintpuppython101
```python
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("Apple is looking at buying U.K. startup for $1 billion")
import pandas as pd
cols = ("text", "lemma", "POS", "explain", "stopword")
rows = []
for t in doc:
    row = [t.text, t.lemma_, t.pos_, spacy.explain(t.pos_), t.is_stop]
    rows.append(row)
df = pd.DataFrame(rows, columns=cols)
df
```

<table>
<thead>
<tr>
<th>text</th>
<th>lemma</th>
<th>POS</th>
<th>explain</th>
<th>stopword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple</td>
<td>Apple</td>
<td>PROPN</td>
<td>proper noun</td>
<td>False</td>
</tr>
<tr>
<td>is</td>
<td>be</td>
<td>VERB</td>
<td>verb</td>
<td>True</td>
</tr>
<tr>
<td>looking</td>
<td>look</td>
<td>VERB</td>
<td>verb</td>
<td>False</td>
</tr>
<tr>
<td>at</td>
<td>at</td>
<td>ADP</td>
<td>adposition</td>
<td>True</td>
</tr>
<tr>
<td>buying</td>
<td>buy</td>
<td>VERB</td>
<td>verb</td>
<td>False</td>
</tr>
<tr>
<td>U.K.</td>
<td>U.K.</td>
<td>PROPN</td>
<td>proper noun</td>
<td>False</td>
</tr>
<tr>
<td>startup</td>
<td>startup</td>
<td>NOUN</td>
<td>noun</td>
<td>False</td>
</tr>
<tr>
<td>for</td>
<td>for</td>
<td>ADP</td>
<td>adposition</td>
<td>True</td>
</tr>
<tr>
<td>$</td>
<td>$</td>
<td>SYM</td>
<td>symbol</td>
<td>False</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>NUM</td>
<td>numeral</td>
<td>False</td>
</tr>
<tr>
<td>billion</td>
<td>billion</td>
<td>NUM</td>
<td>numeral</td>
<td>False</td>
</tr>
</tbody>
</table>
```python
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("Stanford University is located in California. It is a great university.")
import pandas as pd
cols = ("text", "lemma", "POS", "explain", "stopword")
rows = []
for t in doc:
    row = [t.text, t.lemma_, t.pos_, spacy.explain(t.pos_), t.is_stop]
    rows.append(row)
df = pd.DataFrame(rows, columns=cols)
df
```

<table>
<thead>
<tr>
<th></th>
<th>text</th>
<th>lemma</th>
<th>POS</th>
<th>explain</th>
<th>stopword</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Stanford</td>
<td>Stanford</td>
<td>PROPN</td>
<td>proper noun</td>
<td>False</td>
</tr>
<tr>
<td>1</td>
<td>University</td>
<td>University</td>
<td>PROPN</td>
<td>proper noun</td>
<td>False</td>
</tr>
<tr>
<td>2</td>
<td>is</td>
<td>be</td>
<td>VERB</td>
<td>verb</td>
<td>True</td>
</tr>
<tr>
<td>3</td>
<td>located</td>
<td>locate</td>
<td>VERB</td>
<td>verb</td>
<td>False</td>
</tr>
<tr>
<td>4</td>
<td>in</td>
<td>in</td>
<td>ADP</td>
<td>adposition</td>
<td>True</td>
</tr>
<tr>
<td>5</td>
<td>California</td>
<td>California</td>
<td>PROPN</td>
<td>proper noun</td>
<td>False</td>
</tr>
<tr>
<td>6</td>
<td>.</td>
<td>.</td>
<td>PUNCT</td>
<td>punctuation</td>
<td>False</td>
</tr>
<tr>
<td>7</td>
<td>It</td>
<td>-PRON-</td>
<td>PRON</td>
<td>pronoun</td>
<td>True</td>
</tr>
<tr>
<td>8</td>
<td>is</td>
<td>be</td>
<td>VERB</td>
<td>verb</td>
<td>True</td>
</tr>
<tr>
<td>9</td>
<td>a</td>
<td>a</td>
<td>DET</td>
<td>determiner</td>
<td>True</td>
</tr>
<tr>
<td>10</td>
<td>great</td>
<td>great</td>
<td>ADJ</td>
<td>adjective</td>
<td>False</td>
</tr>
<tr>
<td>11</td>
<td>university</td>
<td>university</td>
<td>NOUN</td>
<td>noun</td>
<td>False</td>
</tr>
<tr>
<td>12</td>
<td>.</td>
<td>.</td>
<td>PUNCT</td>
<td>punctuation</td>
<td>False</td>
</tr>
</tbody>
</table>
Python in Google Colab (Python101)

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

```python
import spacy
nlp = spacy.load("en_core_web_sm")
text = "Stanford University is located in California. It is a great university."
doc = nlp(text)
for ent in doc.ents:
    print(ent.text, ent.label_)
```

Stanford University ORG
California GPE

```python
from spacy import displacy
text = "Stanford University is located in California. It is a great university."
doc = nlp(text)
displacy.render(doc, style="ent", jupyter=True)
```

Stanford University ORG is located in California GPE. It is a great university.

https://tinyurl.com/aintpupython101
from spacy import displacy
text = "Stanford University is located in California. It is a great university."
doc = nlp(text)
displacy.render(doc, style="ent", jupyter=True)
displacy.render(doc, style="dep", jupyter=True)
```python
import spacy
nlp = spacy.load("en_core_web_sm")
text = "Stanford University is located in California. It is a great university."
doc = nlp(text)
for ent in doc.ents:
    print(ent.text, ent.label_)
```

Stanford University ORG
California GPE

```python
from spacy import displacy
text = "Stanford University is located in California. It is a great university."
doc = nlp(text)
displacy.render(doc, style="ent", jupyter=True)
```

Stanford University ORG is located in California GPE. It is a great university.

https://tinyurl.com/aintpupython101
Python for Natural Language Processing

- spaCy Chinese Model
- Open Chinese Convert (OpenCC, 開放中文轉換)
- Jieba 結巴中文分詞

Natural Language Toolkit (NLTK)
- Stanza: A Python NLP Library for Many Human Languages

Text Processing and Understanding
- NLTK (Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit)
- NLP Zero to Hero
- Natural Language Processing - Tokenization (NLP Zero to Hero, part 1)
- Natural Language Processing - Sequencing - Turning sentence into data (NLP Zero to Hero, part 2)
- Natural Language Processing - Training a model to recognize sentiment in text (NLP Zero to Hero, part 3)

Keras preprocessing text

JSON File

https://tinyurl.com/aintpupytpython101
MONPA 囧拍：
正體中文斷詞、詞性標註以及命名實體辨識的多任務模型

```
# MONPA 囧拍：正體中文斷詞、詞性標註以及命名實體辨識的多任務模型
# Source: https://github.com/monpa-team/monpa
!pip install monpa

import monpa
sentence = "銀行產業正在改變，金融機構欲挖角科技人才"
words = monpa.cut(sentence)
print(sentence)
print(" ".join(words))
result_pseg = monpa.pseg(sentence)
for item in result_pseg:
    print(item)
```

銀行產業正在改變，金融機構欲挖角科技人才
銀行 產業 正在 改變 ， 金融 機構 欲 挖角 科技 人才
('銀行', 'ORG')
('產業', 'Na')
('正在', 'D')
('改變', 'VC')
('，', 'COMMACATEGORY')
('金融', 'Na')
('機構', 'Nc')
('欲', 'VK')
('挖角', 'VA')
('科技', 'Na')
('人才', 'Na')

https://tinyurl.com/aintpuppython101
jieba
c
words = jieba.cut(sentence)

```python
1 import jieba
2 import jieba.posseg as pseg
3 sentence = "銀行產業正在改變，金融機構欲挖角科技人才"
4 words = jieba.cut(sentence)
5 print(sentence)
6 print(" ".join(words))
7 wordspos = pseg.cut(sentence)
8 result = ""
9 for word, pos in wordspos:
  10     print(word + ' (' + pos + ')')
  11     result = result + ' ' + word + ' (' + pos + ')'
12 print(result.strip())
```

銀行產業正在改變，金融機構欲挖角科技人才
銀行 產業 正在 改變 ， 金融 機構 欲 挖角 科技人才
銀行 (n)
產業 (n)
正在 (t)
改變 (v)
， (x)
金融 (n)
機構 (n)
欲 (d)
挖角 (n)
科技人才 (n)
銀行 (n) 產業 (n) 正在 (t) 改變 (v) ， (x) 金融 (n) 機構 (n) 欲 (d) 挖角 (n) 科技人才 (n)

https://tinyurl.com/aintpupython101
Python Jieba “结巴”中文分词

- https://github.com/fxsjy/jieba
- jieba.set_dictionary('data/dict.txt.big')
  - #/anaconda/lib/python3.5/site-packages/jieba
  - dict.txt (5.4MB)(349,046)
  - dict.txt.big.txt (8.6MB)(584,429)
  - dict.txt.small.txt (1.6MB)(109,750)
  - dict.tw.txt (4.2MB)(308,431)
- https://github.com/Ldkrsi/jieba-zh_TW
  - 结巴中文斷詞台灣繁體版本
Python in Google Colab

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

```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()

# fit the tokenizer on the documents
t.fit_on_texts(docs)

t.fit_on_texts(docs)

t.print('docs:', docs)

t.print('word_counts:', t.word_counts)

t.print('document_counts', t.document_count)

t.print('word_index:', t.word_index)

t.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:
[[0.0.0.1.1.0.0.0.0.]
[0.0.0.1.0.0.0.0.0.]
[0.0.0.0.1.1.0.0.0.]
[0.1.0.0.0.0.1.0.0.0.]
[0.0.0.0.0.0.0.0.1.0.]]
Text Classification

Source: https://developers.google.com/machine-learning/guides/text-classification/
Text Classification Workflow

• Step 1: Gather Data
• Step 2: Explore Your Data
• Step 2.5: Choose a Model*
• Step 3: Prepare Your Data
• Step 4: Build, Train, and Evaluate Your Model
• Step 5: Tune Hyperparameters
• Step 6: Deploy Your Model

Source: https://developers.google.com/machine-learning/guides/text-classification/
Text Classification Flowchart

Text Classification S/W<1500: N-gram

Prepare data

- N-gram
  - N-gram range
    - unigram
    - bigram
    - trigram

- Count mode
  - binary
  - tf-idf
  - count

- Scoring method
  - none
  - f_classif
  - chi2

- Select top_k features [score]
  - min(top: 1K, 2K, ..., 15K, 20K, 25K, ..., 90K, all)

- Normalization mode
  - samplewise
  - None
  - featurewise

- Build model
  - SVM
  - MLP
  - GBDT

Text Classification S/W>=1500: Sequence

Select top_k features [freq]

min(top_1K, 2K, ..., 15K, 20K, 25K, ..., 90K, all)

Normalization mode

samplewise None featurewise

Embeddings

Yes

S/W < 15K

Fine-tuned pre-trained embedding

Frozen pre-trained embedding

Embbedings learned from scratch

No

Build model

RNN

stacked RNN

CNN-RNN

sepCNN

CNN

Hyperparameter tuning

Step 2.5: Choose a Model

Samples/Words < 1500

150,000/100 = 1500

IMDb review dataset,
the samples/words-per-sample ratio is ~ 144

Step 2.5: Choose a Model

Samples/Words < 15,000

1,500,000/100 = 15,000

Step 3: Prepare Your Data

Texts:
T1: 'The mouse ran up the clock'
T2: 'The mouse ran down'

Token Index:
{'the': 1, 'mouse': 2, 'ran': 3, 'up': 4, 'clock': 5, 'down': 6}.
NOTE: 'the' occurs most frequently,
so the index value of 1 is assigned to it.
Some libraries reserve index 0 for unknown tokens,
as is the case here.

Sequence of token indexes:
T1: 'The mouse ran up the clock' =
[1, 2, 3, 4, 1, 5]
T1: 'The mouse ran down' =
[1, 2, 3, 6]
One-hot encoding

'The mouse ran up the clock' =

<table>
<thead>
<tr>
<th>Word</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>The</td>
<td>1</td>
</tr>
<tr>
<td>mouse</td>
<td>2</td>
</tr>
<tr>
<td>ran</td>
<td>3</td>
</tr>
<tr>
<td>up</td>
<td>4</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
</tr>
<tr>
<td>clock</td>
<td>5</td>
</tr>
</tbody>
</table>

[0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 0] ]

[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

<table>
<thead>
<tr>
<th>The mouse ran up the clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
</tr>
<tr>
<td>mouse</td>
</tr>
<tr>
<td>ran</td>
</tr>
<tr>
<td>up</td>
</tr>
<tr>
<td>clock</td>
</tr>
<tr>
<td>down</td>
</tr>
</tbody>
</table>

[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]]

The mouse ran down

<table>
<thead>
<tr>
<th>The mouse ran down</th>
</tr>
</thead>
<tbody>
<tr>
<td>down</td>
</tr>
</tbody>
</table>

[1, 2, 3, 6]

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],
 [0.466, -0.326, 0.884, 0.007]]

Source: https://developers.google.com/machine-learning/guides/text-classification/step-3
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)
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()])

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 = '
'.join(a)
print(tfstring)
tf = tfdict.get('mouse')
print(tf)
from keras.preprocessing.text import Tokenizer

docs = ['Well done!', 'Good work', 'Great effort', 'nice work', 'Excellent!']

t = Tokenizer()
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)

texts_to_matrix = t.texts_to_matrix(docs, mode='count')
print('texts_to_matrix:')
print(texts_to_matrix)

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), ('ni
word_index: {'work': 1, 'well': 2, 'done': 3, 'good': 4, 'great': 5, 'effort': 6, 'nice': 7, 'excellent': 8}
texts_to_matrix:
[[0. 0. 1. 1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 1. 0.]
 [0. 1. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1.]]
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)
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.]
 [0. 1. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 1. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]

Source: https://machinelearningmastery.com/prepare-text-data-deep-learning-keras/
```python
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)
```

texts_to_matrix:

```
[[0. 0. 1. 25276297 1. 25276297 0. 0. 0. 0. 0. 0. ]
 [0. 0. 0. 0. 98082925 0. 0. 1. 25276297 0. 0. 0. 0. ]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 25276297 1. 25276297 0. 0. ]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 25276297 0. ]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 25276297]]
```
Natural Language Toolkit

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational linguistics, plus comprehensive API documentation, NLTK is suitable for linguists, engineers, students, educators, researchers, and industry users alike. NLTK is available for Windows, Mac OS X, and Linux. Best of all, NLTK is a free, open source, community-driven project.

NLTK has been called “a wonderful tool for teaching, and working in, computational linguistics using Python,” and “an amazing library to play with natural language.”

Natural Language Processing with Python provides a practical introduction to programming for language processing. Written by the creators of NLTK, it guides the reader through the fundamentals of writing Python programs, working with corpora, categorizing text, analyzing linguistic structure, and more. The book is being updated for Python 3 and NLTK 3. (The original Python 2 version is still available at http://nltk.org/book_1ed.)

Some simple things you can do with NLTK

Tokenize and tag some text:

```python
>>> import nltk
http://www.nltk.org/
```
TensorFlow NLP Examples

• Basic Text Classification (Text Classification) (46 Seconds)

• NMT with Attention (20-30 minutes)
Text classification with movie reviews

This notebook classifies movie reviews as positive or negative using the text of the review. This is an example of binary—or two-class—classification, an important and widely applicable kind of machine learning problem.

We’ll use the IMDB dataset that contains the text of 50,000 movie reviews from the Internet Movie Database. These are split into 25,000 reviews for training and 25,000 reviews for testing. The training and testing sets are balanced, meaning they contain an equal number of positive and negative reviews.

This notebook uses tf.keras, a high-level API to build and train models in TensorFlow. For a more advanced text classification tutorial using tf.keras, see the MLCC Text Classification Guide.
Summary

• Python for Natural Language Processing
References

• Ramesh Sharda, Dursun Delen, and Efraim Turban (2017), Business Intelligence, Analytics, and Data Science: A Managerial Perspective, 4th Edition, Pearson.
• Rajesh Arumugam (2018), Hands-On Natural Language Processing with Python: A practical guide to applying deep learning architectures to your NLP applications, Packt.
• Christopher D. Manning and Hinrich Schütze (1999), Foundations of Statistical Natural Language Processing, The MIT Press.
• Nitin Hardeniya (2015), NLTK Essentials, Packt.
• Bing Liu (2009), Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data, Springer.