(Artificial Intelligence for Text Analytics: Foundations and Applications)

Time: 2020/05/22 (Fri) (9:10 -12:00)
Place: 國立臺北護理健康大學 (台北市明德路365號) G210
Host: 祝國忠 院長 (健康科技學院院長)

Min-Yuh Day
戴敏育
Associate Professor
副教授
Dept. of Information Management, Tamkang University

http://mail.tku.edu.tw/myday/
2020-05-22
Topics

1. 自然語言處理核心技術與文字探勘
   (Core Technologies of Natural Language Processing and Text Mining)

2. 人工智慧文本分析基礎與應用
   (Artificial Intelligence for Text Analytics: Foundations and Applications)

3. 文本表達特徵工程
   (Feature Engineering for Text Representation)

4. 語意分析和命名實體識別
   (Semantic Analysis and Named Entity Recognition; NER)

5. 深度學習和通用句子嵌入模型
   (Deep Learning and Universal Sentence-Embedding Models)

6. 問答系統與對話系統
   (Question Answering and Dialogue Systems)
Outline

• Python for Natural Language Processing

• Processing Text and Understanding Text
Text Analytics and Text Mining

Text Mining “Knowledge Discovery in Textual Data”

- Document Matching
- Link Analysis
- Search Engines
- Information Retrieval
- POS Tagging
- Lemmatization
- Word Disambiguation

Web Mining
- Web Content Mining
- Web Structure Mining
- Web Usage Mining

Data Mining
- Classification
- Clustering
- Association

Natural Language Processing

Text Analytics

Statistics
- Management Science
- Artificial Intelligence
- Computer Science
- Other Disciplines

Source: Ramesh Sharda, Dursun Delen, and Efraim Turban (2017), Business Intelligence, Analytics, and Data Science: A Managerial Perspective, 4th Edition, Pearson
NLP

Classical NLP

Documents → Language Detection → Pre-processing → Modeling → Output

Pre-processing:
- Tokenization (English)
- Tokenization (Spanish)
- Tokenization (Arabic)
- POS Tagging (English)
- POS Tagging (Spanish)
- POS Tagging (Arabic)
- Stopword Removal (EN)
- Stopword Removal (ES)
- Stopword Removal (AR)

Modeling:
- Feature Extraction (EN)
- Feature Extraction (ES)
- Feature Extraction (AR)
- Modeling (English)
- Modeling (Spanish)
- Modeling (Arabic)
- Inference (English)
- Inference (Spanish)
- Inference (Arabic)

Output:
- Sentiment
- Classification
- Entity Extraction
- Translation
- Topic Modelling

Deep Learning-based NLP

Documents → Preprocessing → Dense Embeddings → Hidden Layers → Output Units

Dense Embeddings:
- obtained via word2vec, doc2vec, GloVe, etc.

Output:
- Sentiment
- Classification
- Entity Extraction
- Translation
- Topic Modelling
Modern NLP Pipeline

Pre-processing

Documents → Language Detection → Tokenize → POS Tagging → ... → Token Filtering → Pre-processed Documents

Pre-processed Documents → Build Vocabulary → Word Embeddings

Classification → Sentiment Analysis → Entity Extraction → Topic Modeling → Similarity

Modern NLP Pipeline

Source: http://mattfortier.me/2017/01/31/nlp-intro-pt-1-overview/
Papers with Code: NLP

Natural Language Processing

- 500 leaderboards • 249 tasks • 100 datasets • 5219 papers with code

Representation Learning

- Representation Learning
  - 7 leaderboards
  - 548 papers with code

- Word Embeddings
  - 454 papers with code

- Graph Embedding
  - 116 papers with code

- Network Embedding
  - 62 papers with code

- Sentence Embeddings
  - 3 leaderboards
  - 52 papers with code

See all 17 tasks

Machine Translation

- Machine Translation
  - 45 leaderboards
  - 612 papers with code

- Transliteration
  - 17 papers with code

- Unsupervised Machine Translation
  - 9 leaderboards
  - 12 papers with code

- Low-Resource Neural Machine Translation
  - 8 papers with code

- Multimodal Machine Translation
  - 7 papers with code

See all 6 tasks

Question Answering

https://paperswithcode.com/area/natural-language-processing
# NLP Benchmark Datasets

<table>
<thead>
<tr>
<th>Task</th>
<th>Dataset</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text Summarization</td>
<td>CNN/DM</td>
<td><a href="https://cs.nyu.edu/~kcho/DMQA/">https://cs.nyu.edu/~kcho/DMQA/</a></td>
</tr>
<tr>
<td>Newsroom</td>
<td><a href="https://summari.es/">https://summari.es/</a></td>
<td></td>
</tr>
<tr>
<td>Gigaword</td>
<td><a href="https://catalog.ldc.upenn.edu/LDC2012T21">https://catalog.ldc.upenn.edu/LDC2012T21</a></td>
<td></td>
</tr>
<tr>
<td>Reading Comprehension</td>
<td>ARC</td>
<td><a href="http://data.allenai.org/arc/">http://data.allenai.org/arc/</a></td>
</tr>
<tr>
<td>CNN/DM</td>
<td><a href="https://cs.nyu.edu/~kcho/DMQA/">https://cs.nyu.edu/~kcho/DMQA/</a></td>
<td></td>
</tr>
<tr>
<td>NewsQA</td>
<td><a href="https://datasets.maluuba.com/NewsQA">https://datasets.maluuba.com/NewsQA</a></td>
<td></td>
</tr>
<tr>
<td>SQuAD</td>
<td><a href="https://rajpurkar.github.io/SQuAD-explorer/">https://rajpurkar.github.io/SQuAD-explorer/</a></td>
<td></td>
</tr>
<tr>
<td>NarrativeQA</td>
<td><a href="https://github.com/deepmind/narrativeqa">https://github.com/deepmind/narrativeqa</a></td>
<td></td>
</tr>
<tr>
<td>Quasar</td>
<td><a href="https://github.com/nyu-dl/SearchQA">https://github.com/nyu-dl/SearchQA</a></td>
<td></td>
</tr>
<tr>
<td>SearchQA</td>
<td><a href="https://github.com/nyu-dl/SearchQA">https://github.com/nyu-dl/SearchQA</a></td>
<td></td>
</tr>
<tr>
<td>Semantic Parsing</td>
<td>AMR parsing</td>
<td><a href="https://amr.isi.edu/index.html">https://amr.isi.edu/index.html</a></td>
</tr>
<tr>
<td>ATIS (SQL Parsing)</td>
<td><a href="https://github.com/jkkummerfeld/text2sql-data/tree/master/data">https://github.com/jkkummerfeld/text2sql-data/tree/master/data</a></td>
<td></td>
</tr>
<tr>
<td>WikiSQL (SQL Parsing)</td>
<td><a href="https://github.com/salesforce/WikiSQL">https://github.com/salesforce/WikiSQL</a></td>
<td></td>
</tr>
<tr>
<td>SST</td>
<td><a href="https://nlp.stanford.edu/sentiment/index.html">https://nlp.stanford.edu/sentiment/index.html</a></td>
<td></td>
</tr>
<tr>
<td>Yelp Reviews</td>
<td><a href="https://www.yelp.com/dataset/challenge">https://www.yelp.com/dataset/challenge</a></td>
<td></td>
</tr>
<tr>
<td>DBpedia</td>
<td><a href="https://wiki.dbpedia.org/Datasets">https://wiki.dbpedia.org/Datasets</a></td>
<td></td>
</tr>
<tr>
<td>TREC</td>
<td><a href="https://trec.nist.gov/data.html">https://trec.nist.gov/data.html</a></td>
<td></td>
</tr>
<tr>
<td>Natural Language Inference</td>
<td>SNLI Corpus</td>
<td><a href="https://nlp.stanford.edu/projects/snli/">https://nlp.stanford.edu/projects/snli/</a></td>
</tr>
<tr>
<td>MultiNLI</td>
<td><a href="https://www.nyu.edu/projects/bowman/multinli/">https://www.nyu.edu/projects/bowman/multinli/</a></td>
<td></td>
</tr>
<tr>
<td>SciTail</td>
<td><a href="http://data.allenai.org/scitail">http://data.allenai.org/scitail</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OneNotes</td>
<td><a href="https://catalog.ldc.upenn.edu/LDC2013T19">https://catalog.ldc.upenn.edu/LDC2013T19</a></td>
</tr>
</tbody>
</table>

Python for Natural Language Processing
Connect Google Colab in Google Drive
Google Colab
Google Colab
Connect Colaboratory to Google Drive
Google Colab
Run Jupyter Notebook
Python 3 GPU
Google Colab
Google Colab Python Hello World

```python
print('Hello World')
```
Python in Google Colab (Python101)

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

https://tinyurl.com/imtkupython101
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.

https://www.gutenberg.org/
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

https://www.gutenberg.org/files/11/11-h/11-h.htm
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/](https://www.nltk.org/book/)

```python
import nltk
nltk.download('gutenberg')
```

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

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

```python
print(nltk.corpus.gutenberg.fileids())
```

```
['austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt', 'bible-kjv.txt', 'blake-poems.txt', 'bryant-stories.txt', 'burgess-buster:...]
```

https://tinyurl.com/imtkupython101
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 her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, '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 how 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 Lon...)

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 wandered 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 lead would go through,' thought poor Alice, 'it would be of very little use way things had happened lately, that Alice had begun to think that very few thing certainly 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 i * * '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 altogether garden at once; but, alas, for poor Alice! when she got to the door, she fou

https://tinyurl.com/imtkupython101
alice.distribution_plot(['Alice', 'Rabbit', 'Hatter', 'Queen'])

```python
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt distributions.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():

# 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/imtkupython101
```python
nltk.download('stopwords')
stopwords = nltk.corpus.stopwords.words('english')
```

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

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

[Link to Python tutorial](https://tinyurl.com/imtku python101)
for word, freq in fdist.items()
if word not in stopwords and word.isalpha():

# import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
fdist_no_punc_no_stopwords = nltk.FreqDist(dict((word, freq) for word, freq in fdist.items() if word not in stopwords and word.isalpha()))
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/imtkupython101
BeautifulSoup

```python
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](https://tinyurl.com/imtkupython101)
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)
```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](https://tinyurl.com/imtkupython101)
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]]

https://tinyurl.com/imtkupython101
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()

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

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. 1. 1. 0. 0. 0. 0. 0. 0.]
 [0. 1. 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. 1.]]
```

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

'The mouse ran up the clock' =

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

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

[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

- down: 6

[1, 2, 3, 6]

[[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()])

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/1FEG6DnGvfwfUbeo4zJ1zTunjMqf2RkCrT
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)
```python
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:', 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), ('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. 1. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 1.]
 [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. 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/
BERT Sequence-level tasks

(a) Sentence Pair Classification Tasks:
MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks:
SST-2, CoLA

BERT Token-level tasks

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Sentiment Analysis: Single Sentence Classification

(b) Single Sentence Classification Tasks: SST-2, CoLA

A Visual Guide to Using BERT for the First Time
(Jay Alammar, 2019)

“a visually stunning rumination on love”
Reviewer #1

That’s a positive thing to say

“reassembled from the cutting room floor of any given daytime soap”
Reviewer #2

That’s negative

## Sentiment Classification: SST2

### Sentences from movie reviews

<table>
<thead>
<tr>
<th>sentence</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>a stirring, funny and finally transporting reimagining of beauty and the beast and 1930s horror films</td>
<td>1</td>
</tr>
<tr>
<td>apparently reassembled from the cutting room floor of any given daytime soap</td>
<td>0</td>
</tr>
<tr>
<td>they presume their audience won't sit still for a sociology lesson</td>
<td>0</td>
</tr>
<tr>
<td>this is a visually stunning rumination on love, memory, history and the war between art and commerce</td>
<td>1</td>
</tr>
<tr>
<td>jonathan parker 's bartleby should have been the be all end all of the modern office anomie films</td>
<td>1</td>
</tr>
</tbody>
</table>

Movie Review Sentiment Classifier

“a visually stunning rumination on love” -> Movie Review Sentiment Classifier -> positive

Movie Review Sentiment Classifier

“a visually stunning rumination on love”

Movie Review Sentiment Classifier

DistilBERT

Logistic
Regression

positive

Source: Jay Alammar (2019), A Visual Guide to Using BERT for the First Time,
http://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/
Movie Review Sentiment Classifier

Model Training

Movie Review Sentiment Classifier

DistilBERT

Already (pre-)trained

Logistic Regression

We will train in this tutorial

Step # 1 Use distilBERT to Generate Sentence Embeddings

Step #1: Use DistilBERT to embed all the sentences

Step #2: Test/Train Split for Model #2, Logistic Regression

Step #3 Train the logistic regression model using the training set

Tokenization

[CLS] a visually stunning rum ###ination on love [SEP] a visually stunning rumination on love

“a visually stunning rumination on love”
tokenizer.encode("a visually stunning rumination on love", add_special_tokens=True)
Tokenization for BERT Model

Flowing Through DistilBERT (768 features)

Model #1 Output Class vector as Model #2 Input

Fine-tuning BERT on Single Sentence Classification Tasks

Model #1 Output Class vector as Model #2 Input

Logistic Regression Model to classify Class vector

“a visually stunning rumination on love”

```python
df = pd.read_csv('https://github.com/clairett/pytorch-sentiment-classification/raw/master/data/SST2/train.tsv',
delimiter='\t', header=None)
df.head()
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a stirring, funny and finally transporting re...</td>
</tr>
<tr>
<td>1</td>
<td>apparently reassembled from the cutting room f...</td>
</tr>
<tr>
<td>2</td>
<td>they presume their audience wo n't sit still f...</td>
</tr>
<tr>
<td>3</td>
<td>this is a visually stunning rumination on love...</td>
</tr>
<tr>
<td>4</td>
<td>jonathan parker 's bartleby should have been t...</td>
</tr>
</tbody>
</table>
Tokenization

tokenized = df[0].apply((lambda x: tokenizer.encode(x, add_special_tokens=True)))

### BERT/_DISTILBERT Input Tensor

<table>
<thead>
<tr>
<th>Input sequences (reviews)</th>
<th>Tokens in each sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>101 1037 ... 0</td>
</tr>
<tr>
<td>1</td>
<td>101 2027 ... 0</td>
</tr>
<tr>
<td>...</td>
<td>... ... ...</td>
</tr>
<tr>
<td>1,999</td>
<td>101 1996 ... 0</td>
</tr>
</tbody>
</table>

Processing with DistilBERT

```python
input_ids = torch.tensor(np.array(padded))
last_hidden_states = model(input_ids)
```
Unpacking the BERT output tensor

last_hidden_states[0]

BERT Output Tensor/predictions

2,000 Output rows (one per sequence)

66 Tokens in each sequence

768 Number of hidden units

Sentence to `last_hidden_state[0]`

```
input_ids
0  1  ...  65
0  101  1037  ...  0
1
...  
1,999
```

```
last_hidden_states[0]
```

![DistilBERT processing a sentence](image)

```
Batch
Tokenize all 2,000 sentences
Put each sentence in its own row
```

```
101  137  1745  14726  19370  12758  2006  2291  102  ...  0
```

```
[CLS]  a  visually  stunning  run  #iniation  on  love  [SEP]  ...  PAD
```

"a visually stunning rumination on love"

Source: Jay Alammar (2019), A Visual Guide to Using BERT for the First Time, 
BERT’s output for the [CLS] tokens

# Slice the output for the first position for all the sequences, take all hidden unit outputs
features = last_hidden_states[0][::0,:].numpy()
The tensor sliced from BERT's output

**Sentence Embeddings**

Dataset for Logistic Regression (768 Features)

The features are the output vectors of BERT for the [CLS] token (position #0)

<table>
<thead>
<tr>
<th>features</th>
<th>0</th>
<th>1</th>
<th>...</th>
<th>767</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,999</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

labels = df[1]
train_features, test_features, train_labels, test_labels =
train_test_split(features, labels)
Score Benchmarks
Logistic Regression Model on SST-2 Dataset

# Training
lr_clf = LogisticRegression()
lr_clf.fit(train_features, train_labels)

#Testing
lr_clf.score(test_features, test_labels)

# Accuracy: 81%
# Highest accuracy: 96.8%
# Fine-tuned DistilBERT: 90.7%
# Full size BERT model: 94.9%

<table>
<thead>
<tr>
<th>sentence</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>a stirring, funny and finally transporting re imagining of beauty and the beast and 1930s horror films</td>
<td>1</td>
</tr>
<tr>
<td>apparently reassembled from the cutting room floor of any given daytime soap</td>
<td>0</td>
</tr>
<tr>
<td>they presume their audience won't sit still for a sociology lesson</td>
<td>0</td>
</tr>
<tr>
<td>this is a visually stunning rumination on love, memory, history and the war between art and commerce</td>
<td>1</td>
</tr>
<tr>
<td>jonathan parker 's bartleby should have been the be all end all of the modern office anomie films</td>
<td>1</td>
</tr>
</tbody>
</table>

A Visual Notebook to Using BERT for the First Time

“a visually stunning rumination on love”
Reviewer #1

That’s a positive thing to say

“reassembled from the cutting room floor of any given daytime soap”
Reviewer #2

That’s negative

Text classification with preprocessed text: 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.

https://www.tensorflow.org/tutorials/keras/text_classification
Python in Google Colab (Python101)

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

Text Classification

- François Chollet (2017), Text classification with preprocessed text: Movie reviews, https://www.tensorflow.org/tutorials/keras/text_classification

Text Classification: IMDB Movie Reviews

Source: François Chollet (2017), Text classification with preprocessed text: Movie reviews, https://www.tensorflow.org/tutorials/keras/text_classification

```python
1!pip install tf-nightly
2!import tensorflow as tf
3!print(tf.__version__)
```

Collecting tf-nightly

Collecting tf-estimator-nightly

Requirement already satisfied: google-pasta>=0.1.8 in /usr/local/lib/python3.6/dist-packages (from tf-nightly)

https://tinyurl.com/imtkupyteron101
Python in Google Colab (Python101)

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

Sentiment Analysis


Sentiment Analysis - Unsupervised Lexical

```
  2 #!wget 'http://mail.tku.edu.tw/myday/data/example/movie_reviews.csv'
  3 !ls

[3]  1 import numpy as np
  2 import pandas as pd
  3 import tensorflow as tf
  4 import tensorflow_hub as hub
  5  6 df = pd.read_csv('http://mail.tku.edu.tw/myday/data/example/movie_reviews.csv')
  7 df.info()
```

https://tinyurl.com/imtkupython101
Summary

• Python for Natural Language Processing

• Processing Text and Understanding Text
References

- Gabe Ignatow and Rada F. Mihalcea (2017), An Introduction to Text Mining: Research Design, Data Collection, and Analysis, SAGE Publications.
- Rajesh Arumugam (2018), Hands-On Natural Language Processing with Python: A practical guide to applying deep learning architectures to your NLP applications, Packt.
- François Chollet (2017), Text classification with preprocessed text: Movie reviews, https://www.tensorflow.org/tutorials/keras/text_classification
人工智慧文本分析基礎與應用
(Artificial Intelligence for Text Analytics: Foundations and Applications)

Time: 2020/05/22 (Fri) (9:10 - 12:00)
Place: 國立臺北護理健康大學 (台北市明德路365號) G210
Host: 祝國忠 院長 (健康科技學院院長)

Min-Yuh Day
戴敏育
Associate Professor
副教授

Dept. of Information Management, Tamkang University
淡江大學 資訊管理學系

http://mail.tku.edu.tw/myday/