AI for Text Analytics
(Processing and Understanding Text)

Min-Yuh Day
Associate Professor
Institute of Information Management, National Taipei University

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

Counts

Samples

https://tinyurl.com/aintpupython101
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'))
```

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

```
[ ] 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

[ ] 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>
```

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

['austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt', 'bible-jev.txt', 'blake-poems.txt', 'bryant-stories.txt', 'burgess-busters.txt',
```

https://tinyurl.com/aintpuppython101
alice.concordance("Alice")

Displaying 25 of 398 matches:

] CHAPTER I - Down the Rabbit-Hole Alice was beginning to get very tired of the 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 she fell past it. 'Well!' thought Alice to herself, 'after such a fall as down, I think --' (for, you see, Alice had not a moment to think about stop. 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 with 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 led 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 th 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 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 altog garden at once; but, alas for poor Alice! when she got to the door, she fou

https://tinyurl.com/aintpuppython101
```python
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
alice.dispersion_plot(['Alice', 'Rabbit', 'Hatter', 'Queen'])
```

https://tinyurl.com/aintpupython101
```python
# import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
fdist = nltk.FreqDist(alice)
fdist.plot(50)
```

[Graph showing frequency distribution of words in Alice's book, with counts decreasing as samples increase.]

https://tinyurl.com/aintpuppython101
for word, freq in fdist.items():
    if word.isalpha():
        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/aintpupython101
```python
import nltk
nltk.download('stopwords')
stopwords = nltk.corpus.stopwords.words('english')
```

https://tinyurl.com/aintpupython101
for word, freq in fdist.items()
if word not in stopwords and word.isalpha()
Alice Top 50 Tokens

50 most common tokens (no stopwords or punctuation)

https://tinyurl.com/aintpuppython101
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()
```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}
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/aintpuppython101
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]]

https://tinyurl.com/aintpuppython101
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

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_counts:', 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. 1. 1. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 1. 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/aintpupython101
One-hot encoding

'The mouse ran up the clock' =

| The   | 1 | [0, 1, 0, 0, 0, 0, 0, 0], |
| mouse | 2 | [0, 0, 1, 0, 0, 0, 0, 0], |
| ran   | 3 | [0, 0, 0, 1, 0, 0, 0, 0], |
| up    | 4 | [0, 0, 0, 0, 1, 0, 0, 0], |
| the   | 1 | [0, 1, 0, 0, 0, 0, 0, 0], |
| clock | 5 | [0, 0, 0, 0, 0, 1, 0, 0] ] |

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

- Male-Female
  - king
  - man
  - woman
  - queen

- Verb Tense
  - walking
  - walked
  - swam
  - swimming

- Country-Capital
  - Canada
  - Spain
  - Italy
  - Germany
  - Japan
  - China
  - Germany
  - Hanoi
  - Tokyo
  - Beijing
  - Moscow
  - Ankara
  - Russia
  - Turkey
  - Madrid
  - Rome

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>
<th>the</th>
<th>mouse</th>
<th>ran</th>
<th>up</th>
<th>clock</th>
<th>down</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</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]]

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

```
terms = ['the', 'mouse', 'ran', 'up', 'the', 'clock', 'the', 'mouse', 'ran', 'down']
sortedset = ['clock', 'down', 'mouse', 'ran', 'the', 'up']
```

https://tinyurl.com/aintpuppython101
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]

https://tinyurl.com/aintpupython101
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 {'the': 0, 'mouse': 1, 'ran': 2, 'up': 3, 'clock': 4, 'down': 5}

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://tinyurl.com/aintpuppython101
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)

https://tinyurl.com/aintpupython101
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)

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)])
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.]]
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/
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)
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 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>

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 Model Training

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

<table>
<thead>
<tr>
<th>Sentence</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>a stirring, funny and finallytransporting re imagining of beauty and the beast and 1930s...</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>...</td>
<td>...</td>
</tr>
<tr>
<td>the movie is undone by a filmmaking methodology that 's just experimental enough</td>
<td>1</td>
</tr>
</tbody>
</table>

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”
Tokenization

tokenizer.encode("a visually stunning rumination on love", add_special_tokens=True)

Tokenization for BERT Model

Flowing Through DistilBERT (768 features)

Model Inputs:

[CLS] 101 1037 17453 14726 19379 12758 2006 2293 102

[CLS] a visually stunning rum ##nation on love [SEP]

768 Number of hidden units

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

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

df.head()
```

|   | 0                   | 1
|---|---------------------|---
| 0 | a stirring, funny and finally transporting re... | 1
| 1 | apparently reassembled from the cutting room f... | 0
| 2 | they presume their audience wo n't sit still f... | 0
| 3 | this is a visually stunning rumination on love... | 1
| 4 | jonathan parker 's bartleby should have been t... | 1

Tokenization

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

<table>
<thead>
<tr>
<th>Raw Dataset</th>
<th>Sequences of Token IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>a stirring, funny and finally transporting re...</td>
<td>[101, 1037, 18385, 1010, 6057, 1998, 2633, 182...]</td>
</tr>
<tr>
<td>apparently reassembled from the cutting room f...</td>
<td>[101, 4593, 2128, 27241, 23931, 2013, 1996, 62...]</td>
</tr>
<tr>
<td>they presume their audience wo n't sit still f...</td>
<td>[101, 2027, 3653, 23545, 2037, 4378, 24185, 10...]</td>
</tr>
<tr>
<td>this is a visually stunning rumination on love...</td>
<td>[101, 2023, 2003, 1037, 17453, 14726, 19379, 1...]</td>
</tr>
<tr>
<td>jonathan parker 's bartleby should have been t...</td>
<td>[101, 5655, 6262, 1005, 1055, 12075, 2571, 376...]</td>
</tr>
</tbody>
</table>

# BERT Input Tensor

## 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

Sentence to last_hidden_state[0]

<table>
<thead>
<tr>
<th>input_ids</th>
<th>last_hidden_states[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0  1  ...  66</td>
<td></td>
</tr>
<tr>
<td>0  101  1037  ...  0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>1,999</td>
<td></td>
</tr>
</tbody>
</table>

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

```
101  137  1745  14726  19379  12758  2006  2291  102  ...  0
[CLS] a visually stunning run imitation on love [SEP] ... PAD
```

“a visually stunning rumination on love”

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/
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)

labels = df[1]
train_features, test_features, train_labels, test_labels =
train_test_split(features, labels)

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

## 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 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.
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

[25] 1!pip install tf-nightly
2 import tensorflow as tf
3 print(tf.__version__)

Collecting tf-nightly
  Downloading https://files.pythonhosted.org/packages/2a/a0/7381cd278a81a9235f032ea811af07be38f7d4ac9781f27tf-nightly-2.5.0-py3-none-any.whl (517.6MB in 24kB/s)
Collecting tf-estimator-nightly
  Downloading https://files.pythonhosted.org/packages/0f/fb/984408ab3aee0bde6c02e1364af75d8fd1e5c485e20tf-estimator-nightly-2.5.0-py3-none-any.whl (460kB in 40.2MB/s)

Requirement already satisfied: google-pasta>=0.1.8 in /usr/local/lib/python3.6/dist-packages (from tf-nightly)
Python in Google Colab (Python101)

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

Sentiment Analysis


Sentiment Analysis - Unsupervised Lexical

[2]

```
1 #!wget http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
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()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
# Column    Non-Null Count   Dtype
---        ------            -----  
0 review   50000 non-null object
1 sentiment 50000 non-null object
dtypes: object(2)
```

https://tinyurl.com/aintpupython101
Summary

• Processing Text
• 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