(AI for Text Analytics)

(Feature Engineering for Text Representation)

1091AITA05
MBA, IMTKU (M2455) (8418) (Fall 2020)
Thu 3, 4 (10:10-12:00) (B206)

Min-Yuh Day
Associate Professor

Institute of Information Management, National Taipei University

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

2020-10-22
課程大綱 (Syllabus)

週次 (Week)  日期 (Date)  內容 (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)
<table>
<thead>
<tr>
<th>週次 (Week)</th>
<th>日期 (Date)</th>
<th>內容 (Subject/Topics)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 2020/10/29</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>
<td></td>
</tr>
<tr>
<td>8 2020/11/05</td>
<td>文本分類 (Text Classification)</td>
<td></td>
</tr>
<tr>
<td>9 2020/11/12</td>
<td>文本摘要和主題模型 (Text Summarization and Topic Models)</td>
<td></td>
</tr>
<tr>
<td>10 2020/11/19</td>
<td>期中報告 (Midterm Project Report)</td>
<td></td>
</tr>
<tr>
<td>11 2020/11/26</td>
<td>文本相似度和分群 (Text Similarity and Clustering)</td>
<td></td>
</tr>
<tr>
<td>12 2020/12/03</td>
<td>語意分析和命名實體識別 (Semantic Analysis and Named Entity Recognition; NER)</td>
<td></td>
</tr>
</tbody>
</table>
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

• Traditional Feature Engineering for Text Data
  • Bag of Words Model
  • Bag of N-Grams Model
  • TF-IDF Model
• Advanced Word Embeddings with Deep Learning
  • Word2Vec Model
  • Robust Word2Vec Models with Gensim
  • GloVe Model
  • FastText Model
NLP

Classical NLP

Documents → Language Detection → Pre-processing (English, Spanish, Arabic) → Modeling (Feature Extraction, Modeling, Inference) → Output (Sentiment, Classification, Entity Extraction, Translation, Topic Modelling)

Deep Learning-based NLP

Documents → Preprocessing → Dense Embeddings (obtained via word2vec, doc2vec, GloVe, etc.) → Hidden Layers → Output Units → Output (Sentiment, Classification, Entity Extraction, Translation, Topic Modelling)

Source: http://blog.aylien.com/leveraging-deep-learning-for-multilingual/
Modern NLP Pipeline
Modern NLP Pipeline

Source: http://mattfortier.me/2017/01/31/nlp-intro-pt-1-overview/
Deep Learning NLP

Documents → Preprocessing → Dense Word Embeddings → Deep Neural Network

Task / Output
- Classification
- Sentiment Analysis
- Entity Extraction
- Topic Modeling
- Document Similarity

Pre-generated Lookup OR Generated in 1st level of NeuralNet

Source: http://mattfortier.me/2017/01/31/nlp-intro-pt-1-overview/
## Overview of Text Vectorization Methods

<table>
<thead>
<tr>
<th>Vectorization Method</th>
<th>Function</th>
<th>Good For</th>
<th>Considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>Counts term frequencies</td>
<td>Bayesian models</td>
<td>Most frequent words not always most informative</td>
</tr>
<tr>
<td>One-Hot Encoding</td>
<td>Binarizes term occurrence (0, 1)</td>
<td>Neural networks</td>
<td>All words equidistant, so normalization extra important</td>
</tr>
<tr>
<td>TF-IDF</td>
<td>Normalizes term frequencies across documents</td>
<td>General purpose</td>
<td>Moderately frequent terms may not be representative of document topics</td>
</tr>
<tr>
<td>Distributed Representations</td>
<td>Context-based, continuous term similarity encoding</td>
<td>Modeling more complex relationships</td>
<td>Performance intensive; difficult to scale without additional tools (e.g., Tensorflow)</td>
</tr>
</tbody>
</table>

Encoding Documents as Vectors

The elephant sneezed at the sight of potatoes.

Bats can see via echolocation. See the bat sight sneeze!

Wondering, she opened the door to the studio.

Token Frequency as Vector Encoding

The elephant sneezed at the sight of potatoes.

Bats can see via echolocation. See the bat sight sneeze!

Wondering, she opened the door to the studio.
One-hot Encoding

The elephant sneezed at the sight of potatoes.

Bats can see via echolocation. See the bat sight sneeze!

Wondering, she opened the door to the studio.

TF-IDF Encoding

The elephant sneezed at the sight of potatoes.

Bats can see via echolocation. See the bat sight sneeze!

Wondering, she opened the door to the studio.

Distributed Representation

The elephant sneezed at the sight of potatoes.

Bats can see via echolocation. See the bat sight sneeze!

Wondering, she opened the door to the studio.

-0.0225403  -0.0212964  0.02708783  0.0049877  0.0492694  -0.03268785  -0.0320941

Pipelines for Text Vectorization and Feature Extraction

Feature Unions for Branching Vectorization

Feature Extraction and Union

The elephant sneezed at the sight of potatoes.

... 

Teddy was terribly lost in the potato patch.

Corpus

Tokenization

Entity Extraction

Keyphrase Extraction

Vectorization

Modeling

Python in Google Colab (Python101)

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

Feature Engineering for Text Representation


Feature Engineering Text Data - Traditional Strategies

Import necessary dependencies and settings

```python
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
import re
import nltk
import matplotlib.pyplot as plt

pd.options.display.max_colwidth = 200
%matplotlib inline

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

https://tinyurl.com/aintpupython101
corpus = ['The sky is blue and beautiful.',
'Love this blue and beautiful sky!',
'The quick brown fox jumps over the lazy dog.',
'A king\'s breakfast has sausages, ham, bacon, eggs, toast and beans',
'I love green eggs, ham, sausages and bacon!',
'The brown fox is quick and the blue dog is lazy!',
'The sky is very blue and the sky is very beautiful today',
'The dog is lazy but the brown fox is quick!'
]
layers = ['weather', 'weather', 'animals', 'food', 'food', 'animals', 'weather', 'animals']

corpus = np.array(corpus)
corpus_df = pd.DataFrame({'Document': corpus, 'Category': layers})
corpus_df = corpus_df[['Document', 'Category']]
corpus_df

https://tinyurl.com/aintpupython101
corpus = np.array(corpus)
corpus_df = pd.DataFrame({"Document": corpus, "Category": labels})
corpus_df = corpus_df[["Document", "Category"]]
corpus_df

<table>
<thead>
<tr>
<th>Document</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>The sky is blue and beautiful.</td>
<td>weather</td>
</tr>
<tr>
<td>Love this blue and beautiful sky!</td>
<td>weather</td>
</tr>
<tr>
<td>The quick brown fox jumps over the lazy dog.</td>
<td>animals</td>
</tr>
<tr>
<td>A king's breakfast has sausages, ham, bacon, eggs, toast and beans</td>
<td>food</td>
</tr>
<tr>
<td>I love green eggs, ham, sausages and bacon!</td>
<td>food</td>
</tr>
<tr>
<td>The brown fox is quick and the blue dog is lazy!</td>
<td>animals</td>
</tr>
<tr>
<td>The sky is very blue and the sky is very beautiful today</td>
<td>weather</td>
</tr>
<tr>
<td>The dog is lazy but the brown fox is quick!</td>
<td>animals</td>
</tr>
</tbody>
</table>

https://tinyurl.com/aintpupython101
wpt = nltk.WordPunctTokenizer()
stop_words = nltk.corpus.stopwords.words('english')

def normalize_document(doc):
    # lower case and remove special characters\whitespaces
    doc = re.sub(r'[^a-zA-Z\s]', '', doc, re.I|re.A)
    doc = doc.lower()
    doc = doc.strip()
    # tokenize document
    tokens = wpt.tokenize(doc)
    # filter stopwords out of document
    filtered_tokens = [token for token in tokens if token not in stop_words]
    # re-create document from filtered tokens
    doc = ' '.join(filtered_tokens)
    return doc

normalize_corpus = np.vectorize(normalize_document)
norm_corpus = normalize_corpus(corpus)
norm_corpus

https://tinyurl.com/aintpupython101
from sklearn.feature_extraction.text import CountVectorizer
# get bag of words features in sparse format
cv = CountVectorizer(min_df=0., max_df=1.)
cv_matrix = cv.fit_transform(norm_corpus)
cv_matrix

# view non-zero feature positions in the sparse matrix
print(cv_matrix)

# view dense representation
# warning might give a memory error if data is too big
cv_matrix = cv_matrix.toarray()
cv_matrix

array([[0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
[0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0],
[1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0],
[1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0],
[0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0],
[0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])

https://tinyurl.com/aintpuppython101
# get all unique words in the corpus
vocab = cv.get_feature_names()
# show document feature vectors
pd.DataFrame(cv_matrix, columns=vocab)
# you can set the n-gram range to 1,2 to get unigrams as well as bigrams

```python
bv = CountVectorizer(ngram_range=(2,2))
bv_matrix = bv.fit_transform(norm_corpus)

bv_matrix = bv_matrix.toarray()
vocab = bv.get_feature_names()
pd.DataFrame(bv_matrix, columns=vocab)
```

https://tinyurl.com/aintpuppython101
```python
from sklearn.feature_extraction.text import TfidfTransformer

tt = TfidfTransformer(norm='l2', use_idf=True, smooth_idf=True)
tt_matrix = tt.fit_transform(cv_matrix)

tt_matrix = tt_matrix.toarray()
vocab = cv.get_feature_names()
pd.DataFrame(np.round(tt_matrix, 2), columns=vocab)
```

<table>
<thead>
<tr>
<th>bacon</th>
<th>beans</th>
<th>beautiful</th>
<th>blue</th>
<th>breakfast</th>
<th>brown</th>
<th>dog</th>
<th>eggs</th>
<th>fox</th>
<th>green</th>
<th>ham</th>
<th>jumps</th>
<th>kings</th>
<th>lazy</th>
<th>love</th>
<th>quick</th>
<th>sausages</th>
<th>sky</th>
<th>toast</th>
<th>today</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.60</td>
<td>0.53</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1.00</td>
<td>0.00</td>
<td>0.49</td>
<td>0.43</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.57</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.38</td>
<td>0.38</td>
<td>0.38</td>
<td>0.00</td>
<td>0.00</td>
<td>0.53</td>
<td>0.00</td>
<td>0.38</td>
<td>0.00</td>
<td>0.38</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.32</td>
<td>0.38</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.32</td>
<td>0.00</td>
<td>0.00</td>
<td>0.38</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.32</td>
<td>0.00</td>
<td>0.00</td>
<td>0.38</td>
<td>0.00</td>
</tr>
<tr>
<td>0.39</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.39</td>
<td>0.00</td>
<td>0.47</td>
<td>0.39</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.39</td>
<td>0.00</td>
<td>0.00</td>
<td>0.39</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>0.42</td>
<td>0.42</td>
<td>0.00</td>
<td>0.42</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.42</td>
<td>0.00</td>
<td>0.42</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.36</td>
<td>0.32</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.72</td>
<td>0.00</td>
<td>0.5</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.45</td>
<td>0.45</td>
<td>0.00</td>
<td>0.45</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.45</td>
<td>0.00</td>
<td>0.45</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

[https://tinyurl.com/aintpuppython101](https://tinyurl.com/aintpuppython101)
from sklearn.feature_extraction.text import TfidfVectorizer

tv = TfidfVectorizer(min_df=0., max_df=1., norm='l2',
                     use_idf=True, smooth_idf=True)
tv_matrix = tv.fit_transform(norm_corpus)
tv_matrix = tv_matrix.toarray()

colnames = tv.get_feature_names()
pd.DataFrame(np.round(tv_matrix, 2), columns=colnames)
```python
from scipy.cluster.hierarchy import dendrogram, linkage

Z = linkage(similarity_matrix, 'ward')
pd.DataFrame(Z, columns=["Document Cluster 1", "Document Cluster 2", "Distance", "Cluster Size"], dtype='object')
```

<table>
<thead>
<tr>
<th>Document Cluster 1</th>
<th>Document Cluster 2</th>
<th>Distance</th>
<th>Cluster Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>7</td>
<td>0.253098</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>6</td>
<td>0.308539</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>8</td>
<td>0.386952</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>9</td>
<td>0.489845</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>0.732945</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>12</td>
<td>2.69565</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>13</td>
<td>3.45108</td>
</tr>
</tbody>
</table>

https://tinyurl.com/aintpupython101
plt.figure(figsize=(8, 3))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Data point')
plt.ylabel('Distance')
dendrogram(Z)
plt.axhline(y=1.0, c='k', ls='--', lw=0.5)

https://tinyurl.com/aintpupython101
```python
from scipy.cluster.hierarchy import fcluster

max_dist = 1.0

cluster_labels = fcluster(Z, max_dist, criterion='distance')
cluster_labels = pd.DataFrame(cluster_labels, columns=['ClusterLabel'])
pd.concat([corpus_df, cluster_labels], axis=1)
```

<table>
<thead>
<tr>
<th>Document</th>
<th>Category</th>
<th>ClusterLabel</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The sky is blue and beautiful.</td>
<td>weather</td>
</tr>
<tr>
<td>1</td>
<td>Love this blue and beautiful sky!</td>
<td>weather</td>
</tr>
<tr>
<td>2</td>
<td>The quick brown fox jumps over the lazy dog.</td>
<td>animals</td>
</tr>
<tr>
<td>3</td>
<td>A king's breakfast has sausages, ham, bacon, eggs, toast and beans</td>
<td>food</td>
</tr>
<tr>
<td>4</td>
<td>I love green eggs, ham, sausages and bacon!</td>
<td>food</td>
</tr>
<tr>
<td>5</td>
<td>The brown fox is quick and the blue dog is lazy!</td>
<td>animals</td>
</tr>
<tr>
<td>6</td>
<td>The sky is very blue and the sky is very beautiful today</td>
<td>weather</td>
</tr>
<tr>
<td>7</td>
<td>The dog is lazy but the brown fox is quick!</td>
<td>animals</td>
</tr>
</tbody>
</table>

[https://tinyurl.com/aintpuppython101](https://tinyurl.com/aintpuppython101)
```python
from sklearn.decomposition import LatentDirichletAllocation
lda = LatentDirichletAllocation(n_components=3, max_iter=10000, random_state=0)
#lda = LatentDirichletAllocation(n_topics=3, max_iter=10000, random_state=0)
dt_matrix = lda.fit_transform(cv_matrix)
features = pd.DataFrame(dt_matrix, columns=['T1', 'T2', 'T3'])
features
```

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.832191</td>
<td>0.083480</td>
<td>0.084329</td>
</tr>
<tr>
<td>1</td>
<td>0.863554</td>
<td>0.069100</td>
<td>0.067346</td>
</tr>
<tr>
<td>2</td>
<td>0.047794</td>
<td>0.047776</td>
<td>0.904430</td>
</tr>
<tr>
<td>3</td>
<td>0.037243</td>
<td>0.925559</td>
<td>0.037198</td>
</tr>
<tr>
<td>4</td>
<td>0.049121</td>
<td>0.903076</td>
<td>0.047802</td>
</tr>
<tr>
<td>5</td>
<td>0.054902</td>
<td>0.047778</td>
<td>0.897321</td>
</tr>
<tr>
<td>6</td>
<td>0.888287</td>
<td>0.055697</td>
<td>0.056016</td>
</tr>
<tr>
<td>7</td>
<td>0.055704</td>
<td>0.055689</td>
<td>0.888607</td>
</tr>
</tbody>
</table>

https://tinyurl.com/aintpupython101
tt_matrix = lda.components_
for topic_weights in tt_matrix:
    topic = [(token, weight) for token, weight in zip(vocab, topic_weights)]
    topic = sorted(topic, key=lambda x: -x[1])
    topic = [item for item in topic if item[1] > 0.6]
    print(topic)
print()
```python
from sklearn.cluster import KMeans

km = KMeans(n_clusters=3, random_state=0)
km.fit_transform(features)
cluster_labels = km.labels_
cluster_labels = pd.DataFrame(cluster_labels, columns=["ClusterLabel"])
pd.concat([corpus_df, cluster_labels], axis=1)
```

<table>
<thead>
<tr>
<th>Document</th>
<th>Category</th>
<th>ClusterLabel</th>
</tr>
</thead>
<tbody>
<tr>
<td>The sky is blue and beautiful.</td>
<td>weather</td>
<td>1</td>
</tr>
<tr>
<td>Love this blue and beautiful sky!</td>
<td>weather</td>
<td>1</td>
</tr>
<tr>
<td>The quick brown fox jumps over the lazy dog.</td>
<td>animals</td>
<td>2</td>
</tr>
<tr>
<td>A king's breakfast has sausages, ham, bacon, eggs, toast and beans</td>
<td>food</td>
<td>0</td>
</tr>
<tr>
<td>I love green eggs, ham, sausages and bacon!</td>
<td>food</td>
<td>0</td>
</tr>
<tr>
<td>The brown fox is quick and the blue dog is lazy!</td>
<td>animals</td>
<td>2</td>
</tr>
<tr>
<td>The sky is very blue and the sky is very beautiful today</td>
<td>weather</td>
<td>1</td>
</tr>
<tr>
<td>The dog is lazy but the brown fox is quick!</td>
<td>animals</td>
<td>2</td>
</tr>
</tbody>
</table>

[https://tinyurl.com/aintpuppython101](https://tinyurl.com/aintpuppython101)
from gensim.models import word2vec

# tokenize sentences in corpus
wpt = nltk.WordPunctTokenizer()
tokenized_corpus = [wpt.tokenize(document) for document in norm_bible]

# Set values for various parameters
feature_size = 100  # Word vector dimensionality
window_context = 30  # Context window size
min_word_count = 1  # Minimum word count
sample = 1e-3  # Downsample setting for frequent words

w2v_model = word2vec.Word2Vec(tokenized_corpus, size=feature_size,
window=window_context, min_count=min_word_count,
sample=sample, iter=50)

# view similar words based on gensim's model
similar_words = {search_term: [item[0] for item in w2v_model.wv.most_similar([search_term], topn=5)]
for search_term in ['god', 'jesus', 'noah', 'egypt', 'john', 'gospel',
'moses', 'famine']}
similar_words

https://tinyurl.com/aintpupython101
w2v_model.wv.most_similar([search_term], topn=5)
from sklearn.decomposition import PCA

pca = PCA(n_components=2, random_state=0)
pcs = pca.fit_transform(w2v_feature_array)
labels = ap.labels_
categories = list(corpus_df["Category"])
plt.figure(figsize=(8, 6))

for i in range(len(labels)):
    label = labels[i]
    color = 'orange' if label == 0 else 'blue' if label == 1 else 'green'
    annotation_label = categories[i]
x, y = pcs[i]
plt.scatter(x, y, c=color, edgecolors='k')
plt.annotate(annotation_label, xy=(x+1e-4, y+1e-3), xytext=(0, 0), textcoords='offset points')

https://tinyurl.com/aintpupython101
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin    Ming-Wei Chang    Kenton Lee    Kristina Toutanova
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

BERT uses a bidirectional Transformer.
OpenAI GPT uses a left-to-right Transformer.
ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTM to generate features for downstream tasks.
Among three, only BERT representations are jointly conditioned on both left and right context in all layers.

Pre-training model architectures

The input embeddings is the sum of the token embeddings, the segmentation embeddings and the position embeddings.
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

### General Language Understanding Evaluation (GLUE) benchmark

#### GLUE Test results

<table>
<thead>
<tr>
<th>System</th>
<th>MNLI-(m/mm)</th>
<th>QQP</th>
<th>QNLI</th>
<th>SST-2</th>
<th>CoLA</th>
<th>STS-B</th>
<th>MRPC</th>
<th>RTE</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>392k</td>
<td>363k</td>
<td>108k</td>
<td>67k</td>
<td>8.5k</td>
<td>5.7k</td>
<td>3.5k</td>
<td>2.5k</td>
<td></td>
</tr>
<tr>
<td>Pre-OpenAI SOTA</td>
<td>80.6/80.1</td>
<td>66.1</td>
<td>82.3</td>
<td>93.2</td>
<td>35.0</td>
<td>81.0</td>
<td>86.0</td>
<td>61.7</td>
<td>74.0</td>
</tr>
<tr>
<td>BiLSTM+ELMo+Attn</td>
<td>76.4/76.1</td>
<td>64.8</td>
<td>79.9</td>
<td>90.4</td>
<td>36.0</td>
<td>73.3</td>
<td>84.9</td>
<td>56.8</td>
<td>71.0</td>
</tr>
<tr>
<td>OpenAI GPT</td>
<td>82.1/81.4</td>
<td>70.3</td>
<td>88.1</td>
<td>91.3</td>
<td>45.4</td>
<td>80.0</td>
<td>82.3</td>
<td>56.0</td>
<td>75.2</td>
</tr>
<tr>
<td>BERT\text{BASE}</td>
<td>84.6/83.4</td>
<td>71.2</td>
<td>90.1</td>
<td>93.5</td>
<td>52.1</td>
<td>85.8</td>
<td>88.9</td>
<td>66.4</td>
<td>79.6</td>
</tr>
<tr>
<td>BERT\text{LARGE}</td>
<td>86.7/85.9</td>
<td>72.1</td>
<td>91.1</td>
<td>94.9</td>
<td>60.5</td>
<td>86.5</td>
<td>89.3</td>
<td>70.1</td>
<td>81.9</td>
</tr>
</tbody>
</table>

- **MNLI**: Multi-Genre Natural Language Inference
- **QQP**: Quora Question Pairs
- **QNLI**: Question Natural Language Inference
- **SST-2**: The Stanford Sentiment Treebank
- **CoLA**: The Corpus of Linguistic Acceptability
- **STS-B**: The Semantic Textual Similarity Benchmark
- **MRPC**: Microsoft Research Paraphrase Corpus
- **RTE**: Recognizing Textual Entailment

Pre-trained word vectors
Word2Vec
wiki.zh.vec (861MB)
332647 word
300 vec

Pre-trained word vectors for 90 languages, trained on Wikipedia using fastText.

These vectors in dimension 300 were obtained using the skip-gram model with default parameters.

https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md

Facebook Research FastText

Word2Vec: wiki.zh.vec

(861MB) (332647 word 300 vec)

https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md
Word Embeddings in LSTM RNN

Time Expanded LSTM Network

LSTM Internal States

Word Embeddings

Input Question: Is this person dancing? 

Fixed length question vector encoded by the LSTM

Source: https://avisingh599.github.io/deeplearning/visual-qa/
Transformer (Attention is All You Need)  
(Vaswani et al., 2017)
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

BERT (Bidirectional Encoder Representations from Transformers)

Overall pre-training and fine-tuning procedures for BERT

Pre-training

Fine-Tuning

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

BERT (Bidirectional Encoder Representations from Transformers)

BERT input representation

BERT, OpenAI GPT, ELMo

Fine-tuning BERT on Different Tasks

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

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

(c) Question Answering Tasks: SQuAD v1.1

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

Pre-trained Language Model (PLM)
Turing Natural Language Generation (T-NLG)

Transformers

State-of-the-art Natural Language Processing for TensorFlow 2.0 and PyTorch

• Transformers
  – pytorch-transformers
  – pytorch-pretrained-bert
• provides state-of-the-art general-purpose architectures
  – (BERT, GPT-2, RoBERTa, XLM, DistilBert, XLNet, CTRL...)
  – for Natural Language Understanding (NLU) and Natural Language Generation (NLG)
    with over 32+ pretrained models
    in 100+ languages
    and deep interoperability between TensorFlow 2.0 and PyTorch.

Source: https://github.com/huggingface/transformers
Transfer Learning in Natural Language Processing

# NLP Benchmark Datasets

<table>
<thead>
<tr>
<th>Task</th>
<th>Dataset</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WMT 2014 EN-FR</td>
<td></td>
</tr>
<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></td>
<td>Newsroom</td>
<td><a href="https://summari.es/">https://summari.es/</a></td>
</tr>
<tr>
<td></td>
<td>Gigaword</td>
<td><a href="https://catalog.ldc.upenn.edu/LDC2012T21">https://catalog.ldc.upenn.edu/LDC2012T21</a></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>Question Generation</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></td>
<td>NewsQA</td>
<td><a href="https://datasets.maluuba.com/NewsQA">https://datasets.maluuba.com/NewsQA</a></td>
</tr>
<tr>
<td></td>
<td>SQuAD</td>
<td><a href="https://rajpurkar.github.io/SQuAD-explorer/">https://rajpurkar.github.io/SQuAD-explorer/</a></td>
</tr>
<tr>
<td></td>
<td>NarrativeQA</td>
<td><a href="https://github.com/deepmind/narrativeqa">https://github.com/deepmind/narrativeqa</a></td>
</tr>
<tr>
<td></td>
<td>Quasar</td>
<td><a href="https://github.com/nyu-dl/Quasar">https://github.com/nyu-dl/Quasar</a></td>
</tr>
<tr>
<td></td>
<td>SearchQA</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></td>
<td>ATIS (SQL Parsing)</td>
<td><a href="https://github.com/jkummerfeld/text2sql-data/tree/master/data">https://github.com/jkummerfeld/text2sql-data/tree/master/data</a></td>
</tr>
<tr>
<td></td>
<td>WikiSQL (SQL Parsing)</td>
<td><a href="https://github.com/salesforce/WikiSQL">https://github.com/salesforce/WikiSQL</a></td>
</tr>
<tr>
<td></td>
<td>SST</td>
<td><a href="https://nlp.stanford.edu/sentiment/index.html">https://nlp.stanford.edu/sentiment/index.html</a></td>
</tr>
<tr>
<td></td>
<td>Yelp Reviews</td>
<td><a href="https://www.yelp.com/dataset/challenge">https://www.yelp.com/dataset/challenge</a></td>
</tr>
<tr>
<td></td>
<td>DBpedia</td>
<td><a href="https://wiki.dbpedia.org/Datasets">https://wiki.dbpedia.org/Datasets</a></td>
</tr>
<tr>
<td></td>
<td>TREC</td>
<td><a href="https://trec.nist.gov/data.html">https://trec.nist.gov/data.html</a></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></td>
<td>MultiNLI</td>
<td><a href="https://www.nyu.edu/projects/bowman/multinli/">https://www.nyu.edu/projects/bowman/multinli/</a></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>

Summary

• Traditional Feature Engineering for Text Data
  • Bag of Words Model
  • Bag of N-Grams Model
  • TF-IDF Model
• Advanced Word Embeddings with Deep Learning
  • Word2Vec Model
  • Robust Word2Vec Models with Gensim
  • GloVe Model
  • FastText Model
References
