D17 計數方法詞向量實作介紹

重要知識點

- 透過實作更加了解計數方法詞向量
- 了解如何使用 SVD 對詞向量進行降維

(本日課程為前日課程的實作，同學可搭配前日課程的講義進行學習)

字詞前處理

在進行字詞向量化之前，我們需要針對文本資料進行前置處理，針對文本資料處理的方法很多，本日課程會採用以下三種前處理程序：

- 文本資料分割成字詞(斷詞)
- 字詞轉換成字詞 ID 清單
- 字詞移除標點符號
- 文本轉化為數字的 ID 形式表示

將所有字詞轉為小寫並移除標點符號：

```python
#定義簡易文本資料(使用Ch17講義中的例子)
corpus = ['You say goodbye and I say hello.']

word_dic = set()
processed_sentence = []

for sentence in corpus:
    #將所有字詞轉為小寫
    sentence = sentence.lower()

    #移除標點符號(可以依據使用狀況決定是否要移除標點符號)
    pattern = r'[^\W_]+'
    sentence = re.findall(pattern, sentence)

    #添加字詞到字典中
    word_dic |= set(sentence)
    processed_sentence.append(sentence)

processed_sentence
```

```
[['you', 'say', 'goodbye', 'and', 'i', 'say', 'hello']]
```

複製程式碼

```python
#定義簡易文本資料(使用Ch17講義中的例子)
corpus = ['You say goodbye and I say hello.']

word_dic = set()
processed_sentence = []

for sentence in corpus:
  #將所有字詞轉為小寫
  sentence = sentence.lower()

  #移除標點符號(可以依據使用狀況決定是否要移除標點符號)
  pattern = r'[^\W_]+'
```

建立 ID 字典清單並將文本轉換為 ID 形式：

```python
#建立字詞ID清單
word2idx = dict()
idx2word = dict()
for word in word_dic:
    if word not in word2idx:
        idx = len(word2idx)
        word2idx[word] = idx
        idx2word[idx] = word

#將文本轉為ID型式
id_mapping = lambda x: word2idx[x]

corpus = np.array([list(map(id_mapping, sentence)) for sentence in processed_sentence])
print(corpus)
print(word2idx)
print(idx2word)
```

```
[[3 5 1 0 4 5 2]]
{'and': 0, 'goodbye': 1, 'hello': 2, 'you': 3, 'i': 4, 'say': 5}
{0: 'and', 1: 'goodbye', 2: 'hello', 3: 'you', 4: 'i', 5: 'say'}
```

文本的ID形式 ← ; → ID 字典清單

複製程式碼

```python
#建立字詞ID清單
word2idx = dict()
idx2word = dict()
for word in word_dic:
  if word not in word2idx:
    idx = len(word2idx)
    word2idx[word] = idx
    idx2word[idx] = word

#將文本轉為ID型式
id_mapping = lambda x: word2idx[x]
corpus = np.array([list(map(id_mapping, sentence)) for sentence in processed_sentence])
```

共現矩陣

將轉化處理過的文本資料轉化為共現矩陣 (字詞向量化)

以 window 為 1 建立共現矩陣：

```python
#定義共現矩陣函式
vocab_size = len(word2idx)
window_size = 1

# initialize co-occurrence matrix
co_matrix = np.zeros(shape=(vocab_size, vocab_size), dtype=np.int32)

for sentence in corpus:
    sentence_size = len(sentence)

    for idx, word_id in enumerate(sentence):
        for i in range(1, window_size+1):
            left_idx = idx - i
            right_idx = idx + i

            if left_idx >= 0:
                left_word_id = sentence[left_idx]
                co_matrix[word_id, left_word_id] += 1

            if right_idx < sentence_size:
                right_word_id = sentence[right_idx]
                co_matrix[word_id, right_word_id] += 1
co_matrix
```

```
array([[0, 1, 0, 0, 1, 0],
       [1, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 1],
       [1, 0, 0, 0, 0, 1],
       [0, 1, 1, 1, 1, 0]], dtype=int32)
```

→ 文本共現矩陣

```python
#定義共現矩陣函式
vocab_size = len(word2idx)
window_size = 1

# initialize co-occurrence matrix
co_matrix = np.zeros(shape=(vocab_size, vocab_size), dtype=np.int32)

for sentence in corpus:
  sentence_size = len(sentence)

  for idx, word_id in enumerate(sentence):
    for i in range(1, window_size+1):
```

詞向量相似度

當將字詞轉化為向量後，時常會需要比較字詞(如字詞為相似詞或反義詞)，而比較轉換為向量的字詞的方法有很多種，其中當要表示字詞的相似度時，最常使用的方法為餘弦相似度 (Cosine Similarity)。

計算 'i' 與 'you' 的字詞相似度：

```python
# 定義餘弦相似度
def cos_similarity(x: np.ndarray, y: np.ndarray, eps: float=1e-8):
    nx = x / (np.sqrt(np.sum(x**2)) + eps)
    ny = y / (np.sqrt(np.sum(y**2)) + eps)

    return np.dot(nx,ny)

# calculate the similarity between I and you
cos_similarity(co_matrix[word2idx['i']], co_matrix[word2idx['you']])
```

```
0.7071067691154799
```

↓ 字詞相似度

```python
# 定義餘弦相似度
def cos_similarity(x: np.ndarray, y: np.ndarray, eps: float=1e-8):
    nx = x / (np.sqrt(np.sum(x**2)) + eps)
    ny = y / (np.sqrt(np.sum(y**2)) + eps)

    return np.dot(nx,ny)

# calculate the similarity between I and you
cos_similarity(co_matrix[word2idx['i']], co_matrix[word2idx['you']])
```

正向點間互資訊(PPMI)

由於共生矩陣在高頻字上的缺陷，而 PMI 中加入了兩字詞共同出現的機率與各自出現的機率的關係，以此解決高頻詞在共生矩陣上的缺陷。而 PPMI 即將 PMI 內會產生負值的情況排除(若出現負值則賦予 0)。

基於共現矩陣建立 PPMI 矩陣：

```python
# 建立初始PPMI矩陣
M = np.zeros_like(co_matrix, dtype=np.float32)
# 取得所有字詞語個別字詞出現次數
N = np.sum(co_matrix)
S = np.sum(co_matrix, axis=0)
total = co_matrix.shape[0]*co_matrix.shape[1]

for i in range(co_matrix.shape[0]):
    for j in range(co_matrix.shape[1]):
        pmi = np.log2(co_matrix[i, j]*N / (S[i]*S[j]))
        M[i, j] = max(0, pmi)
M
```

```
/Users/admin/Documents/Coding/Python/Deep_Learning_Implement/Pytorch_Implemen
ckages/ipykernel_launcher.py:10: RuntimeWarning: divide by zero encountered i
  # Remove the CWD from sys.path while we load stuff.
array([[0.       , 1.5849625, 0.       , 0.       , 1.5849625, 0.       ],
       [1.5849625, 0.       , 0.       , 0.       , 0.       , 0.5849625],
       [0.       , 0.       , 0.       , 0.       , 0.       , 1.5849625],
       [0.       , 0.       , 0.       , 0.       , 0.       , 1.5849625],
       [1.5849625, 0.       , 0.       , 0.       , 0.       , 0.5849625],
       [0.       , 0.5849625, 1.5849625, 1.5849625, 0.5849625, 0.       ]],
      dtype=float32)
```

PPMI (全為大於等於0的值) ←

```python
M = np.zeros_like(co_matrix, dtype=np.float32)
N = np.sum(co_matrix)
S = np.sum(co_matrix, axis=0)
total = co_matrix.shape[0]*co_matrix.shape[1]

for i in range(co_matrix.shape[0]):
  for j in range(co_matrix.shape[1]):
    pmi = np.log2(co_matrix[i, j]*N / (S[i]*S[j]))
    M[i, j] = max(0, pmi)

print(M)
```

奇異值分解(SVD)

對於稀疏矩陣因大部分的元素都為 0，此矩陣包含了許多無法提供訊息的元素，可利用奇異值分解，一方便將矩陣降維(減少計算複雜度)，並保存重要的資訊。

使用 SVD 進行降維：

```python
# 使用np的linalg.svd對PPMI矩陣進行奇異值分解

# SVD
U, S, V = np.linalg.svd(output_ppmi)

# 使用SVD將將原本的稀疏向量轉變為稠密向量
print(f'hello in co-occurrence matrix: {co_matrix[0]}')
print(f"hello in PPMI: {output_ppmi[0]}")
print(f"hello in SVD: {U[0]}")
```

```
hello in co-occurrence matrix: [0 1 0 0 1 0 0]
hello in PPMI: [0.         1.8073549 0.         0.         1.8073549 0.         0.        ]
hello in SVD: [-4.9782813e-01  3.3306691e-16 -3.3306691e-16 -6.8039632e-01
  5.3779924e-01  0.0000000e+00  1.1102230e-16]
```

```python
# 使用np的linalg.svd對PPMI矩陣進行奇異值分解

# SVD
U, S, V = np.linalg.svd(output_ppmi)

# 使用SVD將將原本的稀疏向量轉變為稠密向量
print(f'hello in co-occurrence matrix: {co_matrix[0]}')
print(f"hello in PPMI: {output_ppmi[0]}")
print(f"hello in SVD: {U[0]}")
```

詳細使用操作

請參照使用**技術方法詞向量實作.ipynb** 檔進行更詳細的使用操作

知識點回顧

在這章節我們學習到

- 實作計數方法詞向量
- 使用 SVD 對詞向量進行降維

延伸閱讀

網站：矩陣分解
針對大量資料的矩陣分解介紹

下一步：閱讀範例與完成作業