IAIP 2023-1 HW7

22100579 이진주

Part0: clean the texts

```
def remove_html(text_data):
  soup = BeautifulSoup(text_data, 'lxml')
  return soup.get_text();

def remove_punctuation(text):
  sent = []
  for t in text.split(' '):
    no_punc = "".join([c for c in t if c not in string.punctuation])
    sent.append(no_punc)
  sentence = " ".join(s for s in sent)
  return sentence

def tolower(text):
  return text.lower()

def lemmatization(text):
  nlp = spacy.load('en_core_web_sm')
  doc = nlp(text.strip())
  tok_lem_sentence = [token.lemma_ for token in doc]
  sentence = " ".join(s for s in tok_lem_sentence)
  return sentence

def removeStopword(text):
  stop_words = stopwords.words('english')
  # print("stop words: ", stop_words)
  # print(text, '\n')
  rmv_sw_sentence = [w for w in text.split() if not w in stop_words]
  # print(rmv_sw_sentence)
  removed_word = [w for w in text.split() if not w in
rmv_sw_sentence]
  # print("\nRemoved word: ", set(removed_word))
  sentence = " ".join(s for s in rmv_sw_sentence)
  return sentence

def clean(text):
  text = remove_html(text)
  text = remove_punctuation(text)
  text = lemmatization(text)
  text = tolower(text)
  text = removeStopword(text)
```

```
  return text
```

```
[10] train
```

	rating	review
0	1.0	[4]
1	1.0	[order, pb, embroider, coat, opposite, colour,...
2	1.0	[small, think, little, price, worth, £, 5, mak...
3	1.0	[thought, would, make, lovely, different, cale...
4	1.0	[sand, rubbish, messy, stick, together, like, ...
...	...	...
496	5.0	[daughter, love, run, jump, excitement]
497	5.0	[great, model]
498	5.0	[fantastic, detail, beautiful, model, traction...
499	5.0	[easy, couple, model, great, expand, set, powe...
500	5.0	[buy, 2, year, old, grandson, love, much, also...

501 rows × 2 columns

Part1: print most 5 frequent words for each review data.

```
from collections import Counter

most5 = []

for i in range(len(train)):
  tokens = train['review'][i]
  freq = Counter(tokens)
  top5 = freq.most_common(5)
  most5.append(top5)
  print("review ", i, ": ", end="")
  for j in range(len(top5)):
    if(j!= 4):
      print(top5[j][0], ", ", end="")
    else:
      print(top5[j][0])
```

```
review  432 : set , arrive , 27th , perfect , condition
review  433 : model , standard , door , epoch , 1
review  434 : kit , noncorridor , produce , excellent , exlms
review  435 : really , good , service , great , product
review  436 : great , addition , auto , city , set
review  437 : model , something , br , modeller , want
review  438 : item , moon , exemplary , service , fantastic
review  439 : fantastic , lamp , fair , price , review  440 : model , standard , shop , epoch , 1
review  441 : build , kit , model , top , spin
review  442 : thank , well , make , lyr , 242
review  443 : keep , husband , happy , well , impressed
review  444 : need , controller , speed , decide , model
review  445 : good , kit , go , together , well
review  446 : f7a , quality , walthers , proto , locomotive
review  447 : christmas , follow , toot , train , brilliant
review  448 : train , german , model , company , budget
review  449 : signal , easy , plate , head , type
review  450 : yes , would , recommend , accord , recipient
review  451 : lovely , little , engine , son , love
review  452 : son , day , love , ime , pleased
review  453 : hornby , uncouple , ramptook , session , master
review  454 : quick , job , reasonable , delivery , thank
review  455 : fab , item , review  456 : cleaning , wagon , everything , require , much
review  457 : train , crane , good , set , normal
review  458 : work , take , little , time , instruction
review  459 : excellent , signal , kit , simple , build
review  460 : chassis , wheel , easy , 8 , drive
review  461 : locomotive , piko , mak , g , dcc
review  462 : look , review  463 : really , nice , quality , ho , coach
review  464 : great , review  465 : true , product , discription , review  466 : really , train , lovely , collector , sure
review  467 : happy , purchase , 4 , year , old
review  468 : excellent , engine , arrive , day , early
```

Part2 -v1: Make a word-to-index dictionary from the train data set

```
#part2-v1: word to index dictionary
#case2 통합
import numpy as np

dictionary2 = {}

def make_frequency_dict2(text):
  for word in text:
    if word not in dictionary2:
      dictionary2[word] = 0
    dictionary2[word] += 1

for i in train['review']:
  make_frequency_dict2(i)


vocab_sorted2 = {}
vocab_sorted2 = sorted(dictionary2.items(), key=lambda x:x[1],
reverse = True)

word2index2 = {}

index = 0
for (word, freq) in vocab_sorted2:
  if freq > 1:
```

```
    word2index2[word] = index
    index += 1
word2index2['OOV'] = index

encoded2 = []

for w in most5:
  tmp = []
  print(w)
  for one in w:
    tmp.append(word2index2.get(one, word2index2['OOV']))
  print(tmp)
  encoded2.append(tmp)
```

```
['tts', 'function', 'first', 'venture', 'hornbys']
[1077, 665, 85, 1576, 1576]
['fully', 'satisfied', 'supplierdelivery', 'performance', 'product']
[671, 1576, 1576, 1576, 12]
['good', 'product', 'fast', 'delivery']
[2, 12, 330, 197]
['add', 'hornby', 'collection']
[145, 164, 280]
['absolutely', 'great', 'exactly', 'begood', 'quality']
[423, 14, 429, 1576, 7]
['great', 'item']
[14, 37]
['deliver', 'ahead', 'estimate', 'date', 'first']
[163, 1576, 769, 324, 85]
['excellent', 'swift', 'service', 'quality', 'product']
[72, 1068, 315, 7, 12]
['well', 'time', 'good', 'thank', 'high']
[4, 15, 2, 255, 169]
['first', 'class']
[85, 321]
['loco', 'superb', 'model', 'hornby', 'exger']
[359, 1057, 9, 164, 1576]
['model', 'standard', 'epoch', '1', 'two']
[9, 107, 562, 135, 53]
['mum', 'buy', 'send', '2', 'month']
[1008, 0, 127, 31, 522]
['car', 'walthers', 'trainline', 'make', 'great']
[78, 1559, 1576, 10, 14]
['model', 'shop', 'excellent', 'would', 'pay']
[9, 128, 72, 3, 84]
['first', 'class', 'model', 'usual', 'kato']
[85, 321, 9, 657, 648]
['excellent', 'purpose', 'buy', 'make', 'rock']
[72, 375, 0, 10, 1576]
['receive', 'good', 'train', 'set', 'excellent']
```

Part2 -v2: Make a word-to-rating dictionary.

```
#part2-v2 : word to rating dictionary

word2rating = []
five_rating_dict = [] #(dict indexs) /  0, 1, 2, 3, 4(ratings)

for i in range(index+1):
  tmp = [0,0,0,0,0]
  five_rating_dict.append(tmp)

#일단 five_rating_dict[i]의 0-4(1-5)에다가 별점 쌓기
for i in range(len(encoded2)):
  for j in encoded2[i]:
```

```
    five_rating_dict[j][int(train['rating'][i]-1)] += 1

print("rating 1  2  3  4  5")
for i in range(index+1):
  print(i,": ", end='')
  print(five_rating_dict[i], end='')
  pred = five_rating_dict[i].index(max(five_rating_dict[i]))+1
  word2rating.append(pred)
  print(": ", pred)

print("\nword to rating = ", end='')
for i in range(index+1):
  print(word2rating[i], ",", end='')
```

```
rating 1  2  3  4  5
0 : [12, 11, 6, 6, 5]:  1
1 : [6, 6, 7, 6, 1]:  3
2 : [2, 6, 13, 9, 13]:  3
3 : [8, 1, 2, 1, 2]:  1
4 : [5, 1, 4, 6, 6]:  4
5 : [4, 5, 6, 4, 1]:  3
6 : [9, 9, 10, 5, 1]:  3
7 : [10, 7, 6, 6, 5]:  1
8 : [3, 3, 5, 3, 2]:  3
9 : [1, 4, 1, 7, 19]:  5
10 : [4, 1, 0, 5, 6]:  5
11 : [4, 5, 6, 6, 3]:  3
12 : [6, 7, 4, 3, 9]:  5
13 : [3, 3, 5, 4, 0]:  3
14 : [2, 4, 5, 6, 13]:  5
15 : [2, 0, 2, 3, 3]:  4
16 : [1, 2, 2, 6, 5]:  4
17 : [0, 1, 9, 16, 0]:  4
18 : [4, 1, 4, 3, 3]:  1
19 : [1, 2, 2, 1, 0]:  2
20 : [2, 5, 2, 4, 5]:  2
21 : [5, 3, 4, 2, 1]:  1
22 : [1, 0, 4, 0, 2]:  3
23 : [3, 2, 6, 6, 1]:  3
24 : [3, 4, 5, 2, 1]:  3
25 : [3, 3, 4, 6, 0]:  4
26 : [0, 0, 2, 4, 9]:  5
27 : [2, 5, 4, 1, 0]:  2
28 : [0, 1, 5, 1, 0]:  3
29 : [3, 4, 6, 4, 0]:  3
30 : [2, 1, 4, 2, 5]:  5
31 : [3, 2, 2, 1, 1]:  1
32 : [4, 3, 0, 4, 2]:  1
```

Part4-1: encode test data and predict the rating of test review,

```
# Part4-1: encode test data and predict the rating of test review,
from collections import Counter

#하나의 string review에 대해 예측 rating을 리턴하는 함수
def getPredictedRating(review):
  enc = []
  pred = []
  for i in range(len(review)):
    enc.append(word2index2.get(review[i], word2index2['OOV']))
    pred.append(word2rating[enc[i]])

  counter = Counter(pred)
  most_common = counter.most_common(1)[0]
```

```
  return most_common[0]

correct = 0

print("actial : predicted")
for i in range(len(test)):
  print(test['rating'][i], end='')
  p = getPredictedRating(test['review'][i])
  print('    : ', p, '.0')
  if(int(test['rating'][i] == p)):
    correct += 1

print("count of correct : ", correct, "/", len(test))
```

```
actial : predicted
1.0    :  2 .0
1.0    :  4 .0
1.0    :  4 .0
1.0    :  1 .0
1.0    :  1 .0
1.0    :  1 .0
2.0    :  4 .0
2.0    :  1 .0
2.0    :  1 .0
2.0    :  1 .0
2.0    :  3 .0
3.0    :  4 .0
3.0    :  1 .0
3.0    :  1 .0
3.0    :  1 .0
3.0    :  1 .0
4.0    :  1 .0
4.0    :  4 .0
4.0    :  4 .0
4.0    :  1 .0
4.0    :  4 .0
5.0    :  4 .0
5.0    :  5 .0
5.0    :  3 .0
5.0    :  5 .0
5.0    :  1 .0
count of correct :  8 / 26
```

Part4-2: suggest how to evaluate your predicted result.

```
count of correct :  8 / 26
```

- 전체 test data들의 예측된 rating들과 실제 rating의 값을 비교하여 일치하는 data의 비율을 계산한다.

Part4-3: suggest how to improve your results.

- 현재 코드상에서는 word to index로 매핑되는 단어의 기준을 1개보다 많이 등장했을 때로 설정했는데, 이로 인해 인덱스의 스펙트럼이 너무 넓어져, 상위 5개 모음에 한번도 등장하지 않은 단어 index는 매칭되는 rating도 없어 별점 1점으로 고정되고, 이로 인해 예측 rating에서 1의 비율이 증가했다. 여러 시도를 통해 word to index로 매핑되는 단어 기준을 적절하게 조절하여 정확도를 올릴 수 있을 것 같다.