자연어 데이터 벡터 이해

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

03

APPLICATION

ARTIFICIAL INTELLIGENCE

Notice

THINKING

생각해보기

✓ 자연어 데이터 벡터란 무엇일까요?

학습목표

GOALS

1 정수 인코딩과 패딩에 대해 설명할 수 있다.

2 원핫 인코딩이 무엇인지 이해하고 활용할 수 있다.

3 백오브 워즈 가정에 대해 이해하고 활용할 수 있다.

4 DTM과 TF-IDF에 대해 이해하고 활용할 수 있다.

1 자연어 데이터 인코딩

2 자연어 데이터를 벡터로 표현하기

3 실습

CONTENTS

학습내용

APPLICATION

01 자연어 데이터 인코딩

정수 인코딩(Integer Encoding)

자연 언어를 컴퓨터가 이해할 수 있는 숫자(정수) 형태로 인코딩 하는 과정

- 문장 토큰화 이후에 단어를 빈도순으로 정렬하여 단어 집합(Vocabulary)를 만들고 빈도수가 높은 순으로 정수를 부여하는 방법 등이 존재함

단어집합(Vocabulary)은 중복 되지 않는 모든 단어들의 집합을 의미함

정수 인코딩 예시

- A : 배가 너무 아프다
- B : 배가 너무 고프다
- C : 저기 배가 지나간다
- D : 비가 와서 다리가 아프다

고프다	1
너무	2
다리가	3
배가	4
비가	5
아프다	6
와서	7
저기	8
지나간다	9

A	[4, 2, 6]
B	[4, 2, 1]
C	[8, 4, 9]
D	[5, 7, 3, 6]


```
In [1]: 1 text = "There is a monkey in the jungle. The monkey is very hungry. The monkey is looking for food. The monkey four
        2 text

Out[1]: "There is a monkey in the jungle. The monkey is very hungry. The monkey is looking for food. The monkey found banana
        on the tree. Banana is monkey's favorite food in the jungle. With the monkey's ability, the monkey get the banana. Th
        e monkey is very happy with banana."
```

< 가공 전 텍스트 >

텍스트 전처리 - 토큰화

In [2]:
```
from nltk.tokenize import sent_tokenize

sent_text = sent_tokenize(text)
sent_text
```

Out[2]:
```
['There is a monkey in the jungle.',
 'The monkey is very hungry.',
 'The monkey is looking for food.',
 'The monkey found banana on the tree.',
 "Banana is monkey's favorite food in the jungle.",
 "With the monkey's ability, the monkey get the banana.",
 'The monkey is very happy with banana.']
```

02 빈도수 기준 정수 인코딩 _1) 파이썬 활용

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

텍스트 전처리 - 불용어 제거 및 빈도수 측정

In [3]:
```
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

vocabulary = {}
sentences = []
stop_words = stopwords.words('english')

for sent in sent_text:
    sentence = word_tokenize(sent)
    result = []

    for word in sentence:
        word = word.lower()
        if (word not in stop_words) & (len(word) > 2):
            result.append(word)
            if word not in vocabulary:
                vocabulary[word] = 0
            vocabulary[word] += 1

    sentences.append(result)
```

텍스트 전처리 - 불용어 제거 및 빈도수 측정

```
In [4]: 1 sentences

Out[4]: [['monkey', 'jungle'],
         ['monkey', 'hungry'],
         ['monkey', 'looking', 'food'],
         ['monkey', 'found', 'banana', 'tree'],
         ['banana', 'monkey', 'favorite', 'food', 'jungle'],
         ['monkey', 'ability', 'monkey', 'get', 'banana'],
         ['monkey', 'happy', 'banana']]
```

02 빈도수 기준 정수 인코딩 _1) 파이썬 활용

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

텍스트 전처리 - 불용어 제거 및 빈도수 측정

```
In [5]: 1 vocabulary

Out[5]: {'monkey': 8,
         'jungle': 2,
         'hungry': 1,
         'looking': 1,
         'food': 2,
         'found': 1,
         'banana': 4,
         'tree': 1,
         'favorite': 1,
         'ability': 1,
         'get': 1,
         'happy': 1}
```

2번 이상 등장한 단어들을 빈도수 기준으로 단어에 정수값 부여하기

In [6]:
```
sorted_vocabulary = sorted(vocabulary.items(), key = lambda x:x[1], reverse = True)
print(sorted_vocabulary)
```

```
[('monkey', 8), ('banana', 4), ('jungle', 2), ('food', 2), ('hungry', 1), ('looking', 1), ('found', 1), ('tree', 1),
('favorite', 1), ('ability', 1), ('get', 1), ('happy', 1)]
```

In [7]:
```
integer_embedding = {}
i=0
for (word, frequency) in sorted_vocabulary :
    if frequency > 1 :
        i = i + 1
        integer_embedding[word] = i
print(integer_embedding)
```

```
{'monkey': 1, 'banana': 2, 'jungle': 3, 'food': 4}
```

02 빈도수 기준 정수 인코딩 _1) 파이썬 활용

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

정수 인코딩 진행

In [8]:
```
integer_embedding['oov'] = len(integer_embedding) + 1
integer_embedding
```

Out[8]:
```
{'monkey': 1, 'banana': 2, 'jungle': 3, 'food': 4, 'oov': 5}
```

In [9]:
```
encoded = []
for s in sentences:
    temp = []
    for w in s:
        try:
            temp.append(integer_embedding[w])
        except KeyError:
            temp.append(integer_embedding['oov'])
    encoded.append(temp)
print(encoded)
```

```
[[1, 3], [1, 5], [1, 5, 4], [1, 5, 2, 5], [2, 1, 5, 4, 3], [1, 5, 1, 5, 2], [1, 5, 2]]
```

2번 이상 등장한 단어들을 빈도수 기준으로 정수값 부여하기

In [6]:
```
sorted_vocabulary = sorted(vocabulary.items(), key = lambda x:x[1], reverse = True)
print(sorted_vocabulary)
```

```
[('monkey', 8), ('banana', 4), ('jungle', 2), ('food', 2), ('hungry', 1), ('looking', 1), ('found', 1), ('tree', 1),
('favorite', 1), ('ability', 1), ('get', 1), ('happy', 1)]
```

In [7]:
```
integer_embedding = {}
i=0
for (word, frequency) in sorted_vocabulary :
    if frequency > 1 :
        i = i + 1
        integer_embedding[word] = i
print(integer_embedding)
```

```
{'monkey': 1, 'banana': 2, 'jungle': 3, 'food': 4}
```


In [10]:
```
import numpy as np
from collections import Counter

print(sentences)
```

```
[['monkey', 'jungle'], ['monkey', 'hungry'], ['monkey', 'looking', 'food'], ['monkey', 'found', 'banana', 'tree'],
['banana', 'monkey', 'favorite', 'food', 'jungle'], ['monkey', 'ability', 'monkey', 'get', 'banana'], ['monkey', 'hap
py', 'banana']]
```

< 전처리 후 텍스트 >

Counter를 활용해 단어 집합 에서 빈도수 측정하기

In [11]:
```
words = np.hstack(sentences)
words
```

Out[11]:
```
array(['monkey', 'jungle', 'monkey', 'hungry', 'monkey', 'looking',
       'food', 'monkey', 'found', 'banana', 'tree', 'banana', 'monkey',
       'favorite', 'food', 'jungle', 'monkey', 'ability', 'monkey', 'get',
       'banana', 'monkey', 'happy', 'banana'], dtype='<U8')
```

In [12]:
```
vocabulary = Counter(words)
vocabulary
```

Out[12]:
```
Counter({'monkey': 8,
         'jungle': 2,
         'hungry': 1,
         'looking': 1,
         'food': 2,
         'found': 1,
         'banana': 4,
         'tree': 1,
         'favorite': 1,
         'ability': 1,
         'get': 1,
         'happy': 1})
```

02 빈도수 기준 정수 인코딩 _2) Counter 활용

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

정수 인코딩 진행

In [13]:
```
vocab_size = 4
common_vocabulary = vocabulary.most_common(vocab_size)
common_vocabulary
```

Out[13]:
```
[('monkey', 8), ('banana', 4), ('jungle', 2), ('food', 2)]
```

In [14]:
```
integer_embedding = {}
i = 0
for (word, frequency) in common_vocabulary:
    i = i + 1
    integer_embedding[word] = i
print(integer_embedding)
```

```
{'monkey': 1, 'banana': 2, 'jungle': 3, 'food': 4}
```

In [10]:
```
import numpy as np
from collections import Counter

print(sentences)
```

```
[['monkey', 'jungle'], ['monkey', 'hungry'], ['monkey', 'looking', 'food'], ['monkey', 'found', 'banana', 'tree'],
['banana', 'monkey', 'favorite', 'food', 'jungle'], ['monkey', 'ability', 'monkey', 'get', 'banana'], ['monkey', 'hap
py', 'banana']]
```

< 전처리 후 텍스트 >

02 빈도수 기준 정수 인코딩 _3) NLTK의 Freqdict 활용

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

정수 인코딩 진행

In [15]:
```
from nltk import FreqDist

vocabulary = FreqDist(np.hstack(sentences))
vocabulary
```

Out[15]:
```
FreqDist({'monkey': 8, 'banana': 4, 'jungle': 2, 'food': 2, 'hungry': 1, 'looking': 1, 'found': 1, 'tree': 1, 'favori
te': 1, 'ability': 1, ...})
```

In [16]:
```
vocab_size = 4
common_vocabulary = vocabulary.most_common(vocab_size)
common_vocabulary
```

Out[16]:
```
[('monkey', 8), ('banana', 4), ('jungle', 2), ('food', 2)]
```

In [17]:
```
integer_embedding = {word[0] : index + 1 for index, word in enumerate(common_vocabulary)}
print(integer_embedding)
```

```
{'monkey': 1, 'banana': 2, 'jungle': 3, 'food': 4}
```

```
In [1]:
sentences = [['monkey', 'jungle'],
             ['monkey', 'hungry'],
             ['monkey', 'looking', 'food'],
             ['monkey', 'found', 'banana', 'tree'],
             ['banana', 'monkey', 'favorite', 'food', 'jungle'],
             ['monkey', 'ability', 'monkey', 'get', 'banana'],
             ['monkey', 'happy', 'banana']]
sentences
```

```
Out[1]: [['monkey', 'jungle'],
         ['monkey', 'hungry'],
         ['monkey', 'looking', 'food'],
         ['monkey', 'found', 'banana', 'tree'],
         ['banana', 'monkey', 'favorite', 'food', 'jungle'],
         ['monkey', 'ability', 'monkey', 'get', 'banana'],
         ['monkey', 'happy', 'banana']]
```

< 전처리 후 텍스트 >

02 빈도수 기준 정수 인코딩 _4) Tensorflow Keras 활용

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

.fit_on_texts()를 활용해 빈도수를 기준으로 단어 집합 생성하기

```
In [2]:
from tensorflow.keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences)
tokenizer
```

```
Out[2]: <keras_preprocessing.text.Tokenizer at 0x106bb8940>
```

word_index를 통해 빈도수를 기준으로 정수가 부여된 것을 확인하기

```
In [3]: 1 tokenizer.word_index

Out[3]: {'monkey': 1,
         'banana': 2,
         'jungle': 3,
         'food': 4,
         'hungry': 5,
         'looking': 6,
         'found': 7,
         'tree': 8,
         'favorite': 9,
         'ability': 10,
         'get': 11,
         'happy': 12}
```

02 빈도수 기준 정수 인코딩 _4) Tensorflow Keras 활용

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

word_counts를 활용해 빈도수 확인하기

```
In [4]: 1 tokenizer.word_counts

Out[4]: OrderedDict([('monkey', 8),
                     ('jungle', 2),
                     ('hungry', 1),
                     ('looking', 1),
                     ('food', 2),
                     ('found', 1),
                     ('banana', 4),
                     ('tree', 1),
                     ('favorite', 1),
                     ('ability', 1),
                     ('get', 1),
                     ('happy', 1)])
```

texts_to_sequences()를 활용해 정수 인코딩하기

```
In [5]: 1 tokenizer.texts_to_sequences(sentences)

Out[5]: [[1, 3],
         [1, 5],
         [1, 6, 4],
         [1, 7, 2, 8],
         [2, 1, 9, 4, 3],
         [1, 10, 1, 11, 2],
         [1, 12, 2]]
```


패딩(Padding)

자연 언어를 컴퓨터가 이해할 수 있는 숫자(정수) 형태로 인코딩 하는 과정에서 문장들의 길이를 모두 동일하게 맞추는 작업

- 컴퓨터는 동일한 길이의 문장을 하나의 행렬 형태로 인식하여 병렬적으로 연산할 수 있음
- 보통 문장의 길이를 맞추기 위해 0을 붙이는 제로패딩 방식을 사용함. 자연어처리 뿐만 아니라 비전 등의 분야에서도 사용됨

패딩 예시

- A : 배가 너무 아프다
- B : 배가 너무 고프다
- C : 저기 배가 지나간다
- D : 비가 와서 다리가 아프다

고프다	1
너무	2
다리가	3
배가	4
비가	5
아프다	6
와서	7
저기	8
지나간다	9

A	[4, 2, 6]
B	[4, 2, 1]
C	[8, 4, 9]
D	[5, 7, 3, 6]

제로패딩

A	[4, 2, 6, 0]
B	[4, 2, 1, 0]
C	[8, 4, 9, 0]
D	[5, 7, 3, 6]

In [2]:
```
from tensorflow.keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences)

integer_encoding = tokenizer.texts_to_sequences(sentences)
print(integer_encoding)
```

```
[[1, 3], [1, 5], [1, 6, 4], [1, 7, 2, 8], [2, 1, 9, 4, 3], [1, 10, 1, 11, 2], [1, 12, 2]]
```

< 인코딩 후 텍스트 >

패딩을 통해 가장 긴 문장길이로 맞추기

In [3]:
```
max_len = max(len(encode) for encode in integer_encoding)
max_len
```

Out[3]: 5

In [4]:
```
import numpy as np
for encode in integer_encoding:
    while len(encode) < max_len:
        encode.append(0)

padding = np.array(integer_encoding)
padding

```

Out[4]:
```
array([[ 1,  3,  0,  0,  0],
       [ 1,  5,  0,  0,  0],
       [ 1,  6,  4,  0,  0],
       [ 1,  7,  2,  8,  0],
       [ 2,  1,  9,  4,  3],
       [ 1, 10,  1, 11,  2],
       [ 1, 12,  2,  0,  0]])
```


In [2]:
```
from tensorflow.keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences)

integer_encoding = tokenizer.texts_to_sequences(sentences)
print(integer_encoding)
```

```
[[1, 3], [1, 5], [1, 6, 4], [1, 7, 2, 8], [2, 1, 9, 4, 3], [1, 10, 1, 11, 2], [1, 12, 2]]
```

< 인코딩 후 텍스트 >

pad_sequences()를 통해 패딩 진행하기

```
In [3]:
1 from tensorflow.keras.preprocessing.sequence import pad_sequences
2 padding = pad_sequences(integer_encoding)
3 padding

Out[3]: array([[ 0,  0,  0,  1,  3],
               [ 0,  0,  0,  1,  5],
               [ 0,  0,  1,  6,  4],
               [ 0,  1,  7,  2,  8],
               [ 2,  1,  9,  4,  3],
               [ 1, 10,  1, 11,  2],
               [ 0,  0,  1, 12,  2]], dtype=int32)
```

03 패딩_2) Tensorflow keras 활용

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

pad_sequences()의 max_len 인자를 통해 문장 길이 정하기

```
In [5]:
1 padding = pad_sequences(integer_encoding, padding='post', maxlen=6)
2 padding

Out[5]: array([[ 1,  3,  0,  0,  0,  0],
               [ 1,  5,  0,  0,  0,  0],
               [ 1,  6,  4,  0,  0,  0],
               [ 1,  7,  2,  8,  0,  0],
               [ 2,  1,  9,  4,  3,  0],
               [ 1, 10,  1, 11,  2,  0],
               [ 1, 12,  2,  0,  0,  0]], dtype=int32)
```

pad_sequences()의 value 인자를 통해 추가될 값 설정하기

```
In [6]: 1 padding = pad_sequences(integer_encoding, padding='post', value=-1)
        2 padding

Out[6]: array([[ 1,  3, -1, -1, -1],
               [ 1,  5, -1, -1, -1],
               [ 1,  6,  4, -1, -1],
               [ 1,  7,  2,  8, -1],
               [ 2,  1,  9,  4,  3],
               [ 1, 10,  1, 11,  2],
               [ 1, 12,  2, -1, -1]], dtype=int32)
```


DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

원핫 인코딩(One-Hot Encoding)

정수로 표현되었지만 실제로는 문자인 이 데이터를 기계가 인식할 수 있도록 바꿔주는 방법

원핫 인코딩(One-Hot Encoding)

> **자연 언어에 정수가 부여되었다고 해서 자연 언어 자체가 그 숫자 그대로의 의미를 갖는 것은 아님**

ex) monkey에는 1이, happy에는 12가 부여되었다고 해서 monkey가 happy보다 12배 큰 의미를 갖지는 않음

04 원핫 인코딩

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

원핫 인코딩을 하는 방법

1. 상황과 목적에 맞는 토큰화 툴을 사용해야 한다.
2. 표현하고 싶은 단어의 인덱스 위치에 1을 부여하고, 다른 단어의 인덱스 위치에는 0을 부여한다.

원핫 인코딩 예시

A : 배가 너무 아프다

B : 배가 너무 고프다

C : 저기 배가 지나간다

D : 비가 와서 다리가 아프다

고프다	0
너무	1
다리가	2
배가	3
비가	4
아프다	5
와서	6
저기	7
지나간다	8

고프다	[1,0,0,0,0,0,0,0,0]
너무	[0,1,0,0,0,0,0,0,0]
다리가	[0,0,1,0,0,0,0,0,0]
배가	[0,0,0,1,0,0,0,0,0]
비가	[0,0,0,0,1,0,0,0,0]
아프다	[0,0,0,0,0,1,0,0,0]
와서	[0,0,0,0,0,0,1,0,0]
저기	[0,0,0,0,0,0,0,1,0]
지나간다	[0,0,0,0,0,0,0,0,1]

단어 집합 내의 단어들로 만든 문장을 원핫 인코딩하기

```
In [2]:
sub_text = "오늘 업무를 진행하고 이후 공부를 해야지"
encoded = tokenizer.texts_to_sequences([sub_text])[0]
print(encoded)
```

```
[3, 4, 5, 6, 9, 10]
```

```
In [3]:
onehot_encoding = to_categorical(encoded)
print(onehot_encoding)
```

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

문장 토큰화 이후 정수 인코딩 확인하기

In [1]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import to_categorical

text = "오늘 업무 시간에는 업무를 진행하고 업무 이후 시간에는 자연어 처리 공부를 해야지"

tokenizer = Tokenizer()
tokenizer.fit_on_texts([text])
tokenizer.word_index
```

Out[1]:
```
{'업무': 1,
 '시간에는': 2,
 '오늘': 3,
 '업무를': 4,
 '진행하고': 5,
 '이후': 6,
 '자연어': 7,
 '처리': 8,
 '공부를': 9,
 '해야지': 10}
```


DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

단어의 갯수가 늘어날 수록 벡터를 저장하기 위해 필요한 공간이 계속 늘어남

- 단어 집합에 1000개의 단어가 존재하면 총 1000차원의 벡터가 필요함
- 저장공간 측면에서 매우 비효율적임

단어 간의 유사도를 표현하지 못함

APPLICATION

02 자연어 데이터를 벡터로 표현하기

01 백오브 워즈 가정

백오브 워즈(Bag of Words, BoW)

- 수학에서 bag은 원소의 순서는 고려하지 않고 중복 원소를 허용하는 집합을 의미
- 단어의 등장 순서를 고려하지 않고 문서 내 단어의 등장 빈도를 임베딩으로 사용하는 기법

> **"저자가 생각한 주제가 문서에서의 단어 사용에 녹아 있다"라는 가정이 깔려 있다.**

BoW를 만드는 과정

- 각 단어에 고유한 인덱스를 부여한다.
- 각 인덱스의 위치에 단어 토큰의 등장 횟수를 기록한 벡터를 만든다.

02 백오브 워즈

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

파이썬을 활용해 BoW 만들기

```
In [1]:
import re
from nltk.tokenize import word_tokenize

sentence = re.sub("(\.)","","The male lion and female lion is walking through the jungle")
token = word_tokenize(sentence)
token
```

```
Out[1]: ['The',
 'male',
 'lion',
 'and',
 'female',
 'lion',
 'is',
 'walking',
 'through',
 'the',
 'jungle']
```

< 토큰화 후 텍스트 >

파이썬을 활용해 BoW 만들기

In [2]:
```
word2index = {}
bow = []
for vocab in token:
    vocab = vocab.lower()
    if vocab not in word2index.keys():
        word2index[vocab] = len(word2index)
        bow.insert(len(word2index)-1, 1)
    else:
        index = word2index.get(vocab)
        bow[index] = bow[index] + 1

print(word2index)
```

```
{'the': 0, 'male': 1, 'lion': 2, 'and': 3, 'female': 4, 'is': 5, 'walking': 6, 'through': 7, 'jungle': 8}
```

In [3]:
```
bow
```

Out[3]:
```
[2, 1, 2, 1, 1, 1, 1, 1, 1]
```

02 백오브 워즈

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

싸이킷런을 활용해 BoW 만들기

In [1]:
```
import re
from nltk.tokenize import word_tokenize

sentence = re.sub("(\.)","","The male lion and female lion is walking through the jungle")
token = word_tokenize(sentence)
token
```

Out[1]:
```
['The',
 'male',
 'lion',
 'and',
 'female',
 'lion',
 'is',
 'walking',
 'through',
 'the',
 'jungle']
```

< 토큰화 후 텍스트 >

싸이킷런을 활용해 BoW 만들기

In [4]:
```
from sklearn.feature_extraction.text import CountVectorizer

sentence = ["The male lion and female lion is walking through the jungle"]
vector = CountVectorizer()

print(vector.fit_transform(sentence).toarray())
print(vector.vocabulary_)
```

```
[[1 1 1 1 2 1 2 1 1]]
{'the': 6, 'male': 5, 'lion': 4, 'and': 0, 'female': 1, 'is': 2, 'walking': 8, 'through': 7, 'jungle': 3}
```

02 백오브 워즈

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

CounterVectorizer에서 제공하는 불용어를 제거한 BoW 만들기

In [5]:
```
from sklearn.feature_extraction.text import CountVectorizer

text=["The male lion and female lion is walking through the jungle"]
vect = CountVectorizer(stop_words="english")
print(vect.fit_transform(text).toarray())
print(vect.vocabulary_)
```

```
[[1 1 2 1 1]]
{'male': 3, 'lion': 2, 'female': 0, 'walking': 4, 'jungle': 1}
```

nltk에서 제공하는 불용어를 제거한 BoW 만들기

In [6]:
```
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import stopwords

text=["The male lion and female lion is walking through the jungle"]
sw = stopwords.words("english")
vect = CountVectorizer(stop_words =sw)
print(vect.fit_transform(text).toarray())
print(vect.vocabulary_)
```

```
[[1 1 2 1 1]]
{'male': 3, 'lion': 2, 'female': 0, 'walking': 4, 'jungle': 1}
```


문서 단어 행렬(Document Term Matrix, DTM)

> **각 문서에 대한 BoW 표현 방법을 그대로 갖고 와서, 서로 다른 문서들의 BoW들을 결합한 표현 방법**

- 행과 열이 바뀌면 단어 문서 행렬(Term Document Matrix, TDM) 이라고 하기도 함
- 각 문서에서 등장한 단어의 빈도를 행렬의 값으로 표기함

문서 단어 행렬 예시

- A : 배가 너무 아프다
- B : 배가 너무 고프다
- C : 저기 배가 지나간다
- D : 비가 와서 다리가 아프다

	고프다	너무	다리가	배가	비가	아프다	와서	저기	지나간다
A	0	2	0	1	0	1	0	0	0
B	1	1	0	1	0	0	0	0	0
C	0	0	0	1	0	0	0	2	1
D	0	0	1	0	1	1	1	0	0

03 문서 단어 행렬

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

문서 단어 행렬의 한계

원핫 인코딩과 마찬가지로
공간적 낭비와 계산을 증가시킬 수 있음

많은 문서 벡터가 대부분이 0인 벡터로 존재할 것임

The와 같은 불용어가 문서 내 중요한 단어보다
더 큰 값을 갖게 됨

TF-IDF(Term Frequency-Inverse Document Frequency)

단어의 빈도(TF)와 역 문서 빈도(IDF)를 사용하여 DTM내의 각 단어들마다 중요한 정도를 가중치로 주는 방법

- TF와 IDF를 곱한 값으로 점수가 높은 단어일수록 다른 문서에는 많지 않고 해당 문서에서 자주 등장하는 단어를 의미함
- 벡터 형태이므로 후에 인공신경망의 입력으로도 사용할 수 있음

04 TF-IDF

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

tf(d,t) (Term Frequency)

- 단어의 빈도. DTM에서 각 단어들이 가진 값

df(t) (Document Frequency)

- 특정 단어 t가 등장한 문서의 수
- 특정 단어가 문서에서 몇 번 등장 했는지는 고려하지 않음

idf(d,t) (Inverse Term Frequency)

$$Idf(d,t) = \log\left(\frac{n}{1+df(t)}\right)$$

- 기본적인 개념은 df의 역수
- Log 사용 이유 : 문서의 갯수 n이 커질 수록 IDF 값이 급격하게 증가
- 1을 더한 이유 : 분모가 0이 되지 않도록 하기 위해

예제의 IDF

고프다	$ln(\frac{4}{(1+1)}) = 0.693147$
너무	$ln(\frac{4}{(2+1)}) = 0.287682$
다리가	0.693147
배가	0
비가	0.693147
아프다	0.287682
와서	0.693147
저기	0.693147
지나간다	0.693147

예제의 TF-IDF

동음이의어를 구분하지
못하여 값이 0이 되어버림

	고프다	너무	다리가	배가	비가	아프다	와서	저기	지나간다
A	0	0.57536	0	0	0	0.287682	0	0	0
B	0.693147	0.287682	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0.693147	0.693147
D	0	0	0	0	0.693147	0.287682	0.693147	0	0

04 TF-IDF

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

파이썬으로 TF-IDF 구현하기

In [1]:
```
documents = [
    "배가 너무 너무 아프다",
    "배가 너무 고프다",
    "저기 저기 배가 지나간다",
    "비가 와서 다리가 아프다"
]

vocabulary = list(set(w for doc in documents for w in doc.split()))
vocabulary.sort()

vocabulary
```

Out[1]: ['고프다', '너무', '다리가', '배가', '비가', '아프다', '와서', '저기', '지나간다']

TF, IDF, TF-IDF를 계산하는 함수

In [2]:

```
import math

num_documents = len(documents)

def tf(t, d):
    return d.count(t)

def idf(t):
    df = 0
    for doc in documents:
        df += t in doc
    return math.log(num_documents/(df + 1))

def tfidf(t, d):
    return tf(t,d)* idf(t)
```

04 TF-IDF

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

TF 계산하기

In [3]:

```
import pandas as pd

result = []
for i in range(num_documents):
    result.append([])
    d = documents[i]
    for j in range(len(vocabulary)):
        t = vocabulary[j]
        result[-1].append(tf(t, d))

term_frequency = pd.DataFrame(result, columns = vocabulary)
term_frequency
```

Out[3]:

	고프다	너무	다리가	배가	비가	아프다	와서	저기	지나간다
0	0	2	0	1	0	1	0	0	0
1	1	1	0	1	0	0	0	0	0
2	0	0	0	1	0	0	0	2	1
3	0	0	1	0	1	1	1	0	0

IDF 계산하기

In [4]:
```
result = []
for j in range(len(vocabulary)):
    t = vocabulary[j]
    result.append(idf(t))

inverse_tf = pd.DataFrame(result, index = vocabulary, columns = ["IDF"])
inverse_tf
```

Out[4]:

	IDF
고프다	0.693147
너무	0.287682
다리가	0.693147
배가	0.000000
비가	0.693147
아프다	0.287682
와서	0.693147
저기	0.693147
지나간다	0.693147

04 TF-IDF

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

TF-IDF 계산하기

In [5]:
```
result = []
for i in range(num_documents):
    result.append([])
    d = documents[i]
    for j in range(len(vocabulary)):
        t = vocabulary[j]
        result[-1].append(tfidf(t,d))

tf_idf = pd.DataFrame(result, columns = vocabulary)
tf_idf
```

Out[5]:

	고프다	너무	다리가	배가	비가	아프다	와서	저기	지나간다
0	0.000000	0.575364	0.000000	0.0	0.000000	0.287682	0.000000	0.000000	0.000000
1	0.693147	0.287682	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	1.386294	0.693147
3	0.000000	0.000000	0.693147	0.0	0.693147	0.287682	0.693147	0.000000	0.000000

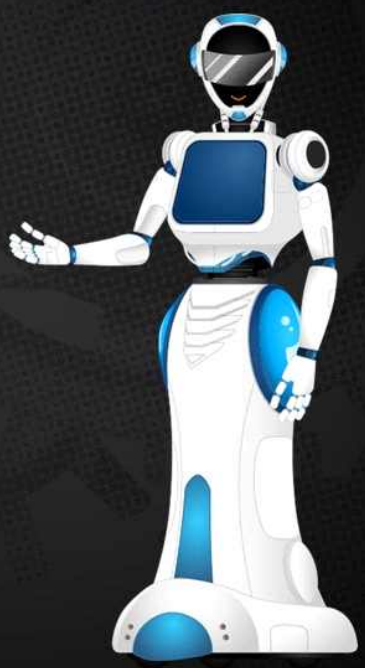
APPLICATION

03 실습

SUMMARY

학습정리

- 자연어에 정수를 부여하는 정수 인코딩
- 자연어의 병렬 연산을 위한 패딩과 원핫 인코딩
- 문서 내 단어의 등장 빈도를 기준으로하는 백오브워즈
- 단어들마다 가중치를 고려한 TF-IDF



EXPANSION

확장하기

1. 정수 인코딩은 어떤 것이고 어떻게 진행할 수 있을까요?
2. 패딩은 어떤 작업이고 어떻게 적용할 수 있을까요?
3. 원 핫 인코딩은 무엇이고 자연어에는 어떻게 적용될 수 있을까요?
4. 백 오브 워즈는 어떤 방법으로 만들 수 있을까요?
5. TF-IDF에는 어떤 특징이 있을까요?

참고 문헌

REFERENCE

◆ 참고 사이트
- 용어들에 대한 정의 : https://ko.wikipedia.org/wiki.
- 점프 투 파이썬 : https://wikidocs.net/1669
- 딥러닝을 이용한 자연어 처리 입문 : https://wikidocs.net/book/2155

◆ 참고 서적
- 김기현, 『김기현의 자연어 처리 딥러닝 캠프 파이토치 편』, 한빛미디어, 2019
- 잘라지 트하나키, 『파이썬 자연어 처리의 이론과 실제』, 에이콘, 2017

서체 출처 : 에스코어드림체-㈜에스코어, 나눔글꼴체-㈜네이버