딥러닝을 이용한 자연어 처리 입문
02. 텍스트 전처리(Text Preprocessing) 06~10

2021.06.20

집현전 초급반 2조: 이창훈(조장), 이문형, 유예림

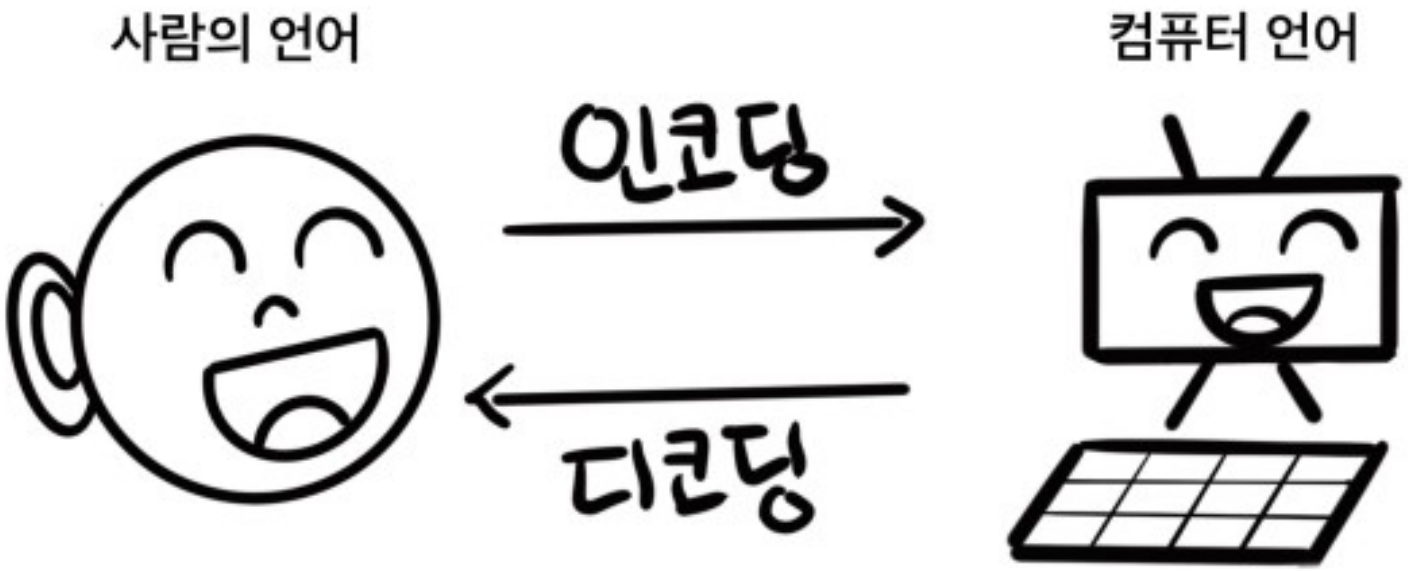
https://wikidocs.net/book/2155

목차

1. 정수 인코딩(Integer Encoding)

◆ 인코딩이란?

- 자연어를 기계가 이해할 수 있는 언어로 변환하는 과정

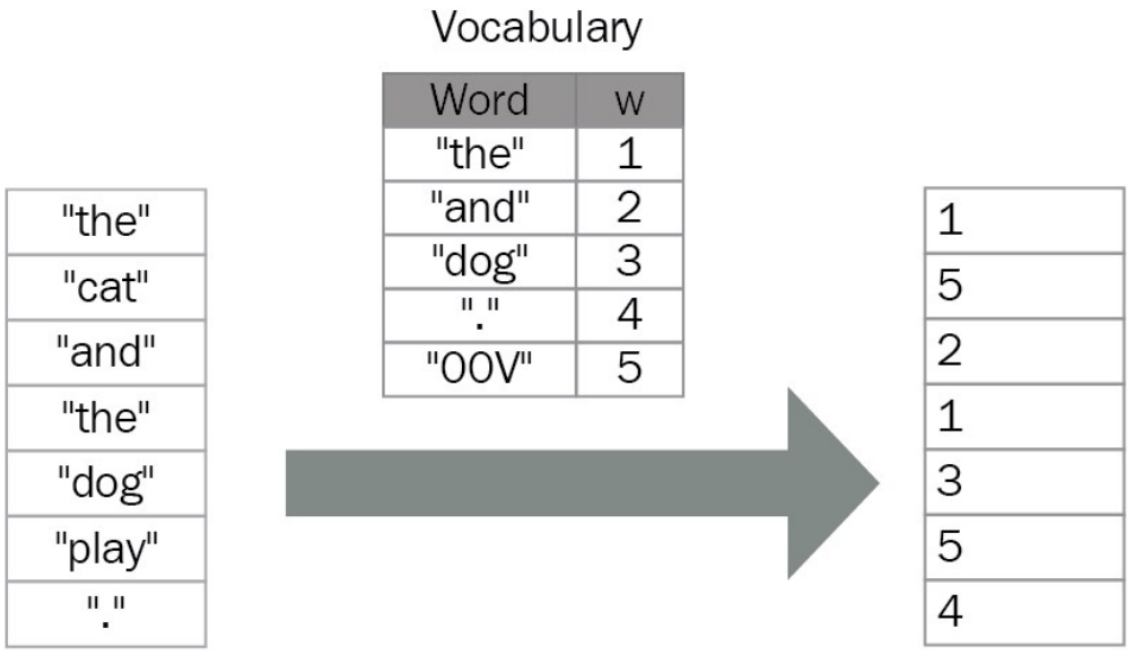


https://blog.naver.com/hs7345570/222206666130

1. 정수 인코딩(Integer Encoding)

- 정수 인코딩이란?
  - word2index 맵핑
  - cleansing, tokenization, subword segmentation 등의 전처리 작업 이후에 모델에 넣기 전에 스해하

- 인덱스 부여 방법
  1) 랜덤하게 부여함
  2) 단어에 대한 빈도수를 기준으로 정렬 후 부여함



https://subscription.packtpub.com/book/web_development/9781786465825/3/ch03lvl1sec32/encoding-and-embedding

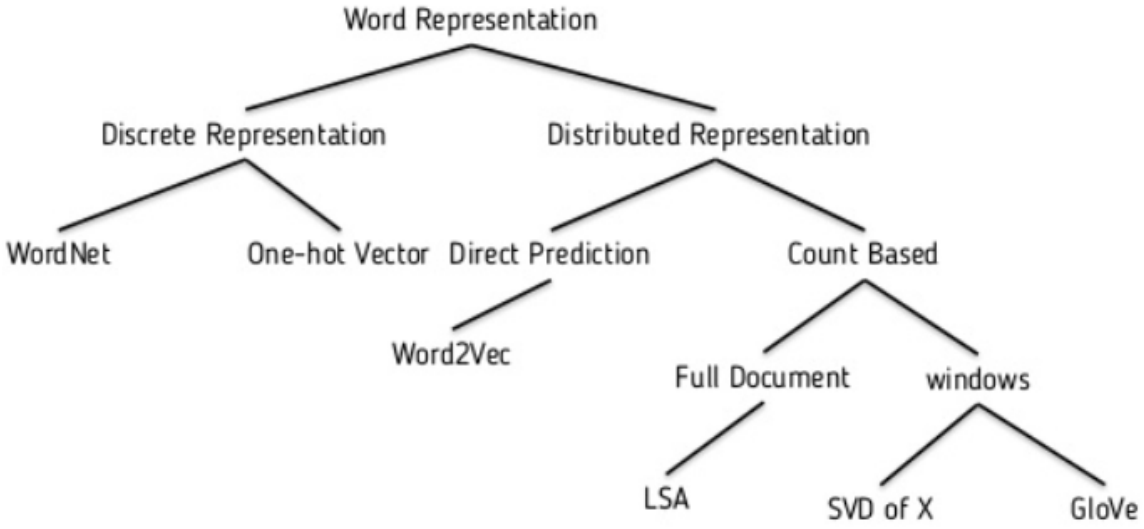
1. 정수 인코딩(Integer Encoding)

- Word Representation
  - 단어를 벡터로 잘 표현하는 방법으로 발전함

Language Representation 흐름도

- One Hot Encoding
- Word2Vec – CBOW, Skip-Gram
- Glove
- FastText
- Cove
- ELMO
- GPT
- BERT
- GPT-2
- ERNIE
- XLNET
- ERNIE 2.0
- ROBERTa

Word Representation 분류



https://www.youtube.com/watch?v=sD_cwlDZP8A&list=PL9mhQYIlKEhcyCTvJq6tGJ3_B6hRjLm2N&index=3

https://www.slideshare.net/keunbongkwak/gloveglobal-vectors-for-word-representation

1. 정수 인코딩(Integer Encoding)

- 정수 인코딩 방법
  1) dictionary 사용하기
  2) Counter 사용하기
  3) NLTK의 FreqDist 사용하기
  4) enumerate 이해하기
  5) 케라스(Keras)의 텍스트 전처리

1. 정수 인코딩(Integer Encoding)

◆ 정수 인코딩 방법 1) dictionary 사용하기

- 라이브러리 및 데이터

```
[1]  !pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.7/dist-packages (3.2.5)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from nltk) (1.15.0)
```

```
[2]  import nltk
     nltk.download('punkt')
     nltk.download('stopwords')

     from nltk.tokenize import sent_tokenize
     from nltk.tokenize import word_tokenize
     from nltk.corpus import stopwords
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
[3]  text = "A barber is a person. a barber is good person. a barber is huge person. he Knew A Secret! The Secret He Kept is huge secret. Huge secret. His barber kept his word. a l
```

1. 정수 인코딩(Integer Encoding)

- 정수 인코딩 방법 1) dictionary 사용하기
  - 문장 토큰화
  - NLTK의 sent_tokenize() 함수는 nltk.tokenize.punkt 모듈의 PunktSentenceTokenizer 인스턴스를 사용해서 문장의 시작과 끝을 구분함
  - 이미 다수의 유럽 언어에서 검증이 됨

```
# 문장 토큰화
text = sent_tokenize(text)
print(text)
```

```
['A barber is a person.', 'a barber is good person.', 'a barber is huge person.', 'he Knew A Secret!', 'The Secret He Kept is huge secret.', 'Huge secret.', 'His barber kept his word.', 'a barber k
```

1. 정수 인코딩(Integer Encoding)

- 정수 인코딩 방법 1) dictionary 사용하기
  - 정제와 단어 토큰화
  - NLTK의 word_tokenize() 함수는 TreebankWordTokenizer 클래스의 인스턴스에 tokenize() 함수를 호출하는 래퍼 함수임
  - 공백과 구두점을 이용해 단어를 분해하고, 구두점을 버리지는 않음

1. 정수 인코딩(Integer Encoding)

◆ 정수 인코딩 방법 1) dictionary 사용하기

- 정제와 단어 토큰화

```
[5]  # 정제와 단어 토큰화
     vocab = {} # 파이썬의 dictionary 자료형
     sentences = []
     stop_words = set(stopwords.words('english'))

     for i in text:
         sentence = word_tokenize(i) # 단어 토큰화를 수행합니다.
         result = []

         for word in sentence:
             word = word.lower() # 모든 단어를 소문자화하여 단어의 개수를 줄입니다.
             if word not in stop_words: # 단어 토큰화 된 결과에 대해서 불용어를 제거합니다.
                 if len(word) > 2: # 단어 길이가 2이하인 경우에 대하여 추가로 단어를 제거합니다.
                     result.append(word)
                     if word not in vocab:
                         vocab[word] = 0
                     vocab[word] += 1
         sentences.append(result)
     print(sentences)
```

```
[['barber', 'person'], ['barber', 'good', 'person'], ['barber', 'huge', 'person'], ['knew', 'secret'], ['secret', 'kept', 'huge', 'secret'], ['huge', 'secret'], ['barber', 'kept', 'word'], ['barber
```

1. 정수 인코딩(Integer Encoding)

- 정수 인코딩 방법 1) dictionary 사용하기
  - vocab에는 중복을 제거한 단어와 각 단어에 대한 빈도수가 기록됨

```
[6] print(vocab)

{'barber': 8, 'person': 3, 'good': 1, 'huge': 5, 'knew': 1, 'secret': 6, 'kept': 4, 'word': 2, 'keeping': 2, 'driving': 1, 'crazy': 1, 'went': 1, 'mountain': 1}
```

```
[7] print(vocab["barber"]) # 'barber'라는 단어의 빈도수 출력

8
```

1. 정수 인코딩(Integer Encoding)

◆ 정수 인코딩 방법 1) dictionary 사용하기

- 빈도수가 높은 순서대로 정렬함

```
[8]  vocab_sorted = sorted(vocab.items(), key = lambda x:x[1], reverse = True)
     print(vocab_sorted)
```

```
[('barber', 8), ('secret', 6), ('huge', 5), ('kept', 4), ('person', 3), ('word', 2), ('keeping', 2), ('good', 1), ('knew', 1), ('driving', 1), ('crazy', 1), ('went', 1), ('mountain', 1)]
```

1. 정수 인코딩(Integer Encoding)

◆ 정수 인코딩 방법 1) dictionary 사용하기

- 높은 빈도수를 가진 단어일수록 낮은 정수 인덱스를 부여함
- 빈도수 상위 n개의 단어만 사용함

```
[9]  word_to_index = {}
     i=0
     for (word, frequency) in vocab_sorted :
         if frequency > 1 : # 정제(Cleaning) 챕터에서 언급했듯이 빈도수가 적은 단어는 제외한다.
             i=i+1
             word_to_index[word] = i
     print(word_to_index)
```

```
{'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5, 'word': 6, 'keeping': 7}
```

```
[10] vocab_size = 5
     words_frequency = [w for w,c in word_to_index.items() if c >= vocab_size + 1] # 인덱스가 5 초과인 단어 제거
     for w in words_frequency:
         del word_to_index[w] # 해당 단어에 대한 인덱스 정보를 삭제
     print(word_to_index)
```

```
{'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5}
```

1. 정수 인코딩(Integer Encoding)

◆ 정수 인코딩 방법 1) dictionary 사용하기

- word_to_index에 단어 집합에 존재하지 않는 단어들 (Out-Of-Vocabulary, OOV)를 새롭게 추가하고,

  단어 집합에 없는 단어들은 'OOV'의 인덱스로 인코딩함

```
[11] word_to_index['OOV'] = len(word_to_index) + 1
```

```
[12] encoded = []
     for s in sentences:
         temp = []
         for w in s:
             try:
                 temp.append(word_to_index[w])
             except KeyError:
                 temp.append(word_to_index['OOV'])
         encoded.append(temp)
     print(encoded)
```

```
[[1, 5], [1, 6, 5], [1, 3, 5], [6, 2], [2, 4, 3, 2], [3, 2], [1, 4, 6], [1, 4, 6], [1, 4, 2], [6, 6, 3, 2, 6, 1, 6], [1, 6, 3, 6]]
```

1. 정수 인코딩(Integer Encoding)

◆ 정수 인코딩 방법 2) Counter 사용하기

- 라이브러리 및 데이터
- 단어 집합을 만들기 위해서 문장의 경계인 [,]를 제거하고 단어들을 하나의 리스트로 만듦

```
[13] from collections import Counter
```

```
[14] print(sentences)
```

```
[['barber', 'person'], ['barber', 'good', 'person'], ['barber', 'huge', 'person'], ['knew', 'secret'], ['secret', 'kept', 'huge', 'secret'], ['huge', 'secret'], ['barber', 'kept', 'word'], ['barber
```

```
[15] words = sum(sentences, [])
     # 위 작업은 words = np.hstack(sentences)로도 수행 가능.
     print(words)
```

```
['barber', 'person', 'barber', 'good', 'person', 'barber', 'huge', 'person', 'knew', 'secret', 'secret', 'kept', 'huge', 'secret', 'huge', 'secret', 'barber', 'kept', 'word', 'barber', 'kept', 'wor
```

1. 정수 인코딩(Integer Encoding)

◆ 정수 인코딩 방법 2) Counter 사용하기

- Counter(), most_common()

```
[16] vocab = Counter(words) # 파이썬의 Counter 모듈을 이용하면 단어의 모든 빈도를 쉽게 계산할 수 있습니다.
     print(vocab)
```

```
Counter({'barber': 8, 'secret': 6, 'huge': 5, 'kept': 4, 'person': 3, 'word': 2, 'keeping': 2, 'good': 1, 'knew': 1, 'driving': 1, 'crazy': 1, 'went': 1, 'mountain': 1})
```

```
[17] print(vocab["barber"]) # 'barber'라는 단어의 빈도수 출력
```

```
8
```

```
[18] vocab_size = 5
     vocab = vocab.most_common(vocab_size) # 등장 빈도수가 높은 상위 5개의 단어만 저장
     vocab
```

```
[('barber', 8), ('secret', 6), ('huge', 5), ('kept', 4), ('person', 3)]
```

```
[19] word_to_index = {}
     i = 0
     for (word, frequency) in vocab :
         i = i+1
         word_to_index[word] = i
     print(word_to_index)
```

```
{'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5}
```

1. 정수 인코딩(Integer Encoding)

◆ 정수 인코딩 방법 3) NLTK의 FreqDist 사용하기

- 라이브러리 및 FreqDist()

```
[20] from nltk import FreqDist
     import numpy as np
```

```
[21] # np.hstack으로 문장 구분을 제거하여 입력으로 사용 . ex) ['barber', 'person', 'barber', 'good' ... 중략 ...
     vocab = FreqDist(np.hstack(sentences))
```

```
[22] print(vocab["barber"]) # 'barber'라는 단어의 빈도수 출력
```

```
     8
```

1. 정수 인코딩(Integer Encoding)

- 정수 인코딩 방법 3) NLTK의 FreqDist 사용하기
  - most_common(), enumerate()

```
[103] vocab_size = 5
      vocab = vocab.most_common(vocab_size) # 등장 빈도수가 높은 상위 5개의 단어만 저장
      vocab

      [('barber', 8), ('secret', 6), ('huge', 5), ('kept', 4), ('person', 3)]

[104] word_to_index = {word[0] : index + 1 for index, word in enumerate(vocab)}
      print(word_to_index)

      {'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5}
```

1. 정수 인코딩(Integer Encoding)

- 정수 인코딩 방법 4) enumerate 이해하기
  - enumerate()는 순서가 있는 자료형(list, set, tuple, dictionary, string)을 입력으로 받아 인덱스를 순차적으로 함께 리턴함

```
[105] test=['a', 'b', 'c', 'd', 'e']
      for index, value in enumerate(test): # 입력의 순서대로 0부터 인덱스를 부여함.
        print("value : {}, index: {}".format(value, index))

      value : a, index: 0
      value : b, index: 1
      value : c, index: 2
      value : d, index: 3
      value : e, index: 4
```

1. 정수 인코딩(Integer Encoding)

- 정수 인코딩 방법 5) 케라스(Keras)의 텍스트 전처리
  - 라이브러리 및 데이터
  - keras에서는 기본적인 전처리를 위한 도구들을 제공함
  - 자동으로 높은 빈도수를 가진 단어일수록 낮은 정수 인덱스를 부여함

```
[106] from tensorflow.keras.preprocessing.text import Tokenizer
```

```
[107] sentences=[['barber', 'person'], ['barber', 'good', 'person'], ['barber', 'huge', 'person'], ['knew', 'secret'], ['secret', 'kept', 'huge', 'secret'], ['huge', 'secret'], ['barber', 'kept', 'word
```

1. 정수 인코딩(Integer Encoding)

◆ 정수 인코딩 방법 5) 케라스(Keras)의 텍스트 전처리

- fit_on_texts(), word_index, index_word, word_counts, text_to_sequences()

```
[156] tokenizer = Tokenizer()
      tokenizer.fit_on_texts(sentences) # fit_on_texts()안에 코퍼스를 입력으로 하면 빈도수를 기준으로 단어 집합을 생성한다.
```

```
[157] print(tokenizer.word_index)
```

```
{'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5, 'word': 6, 'keeping': 7, 'good': 8, 'knew': 9, 'driving': 10, 'crazy': 11, 'went': 12, 'mountain': 13}
```

```
[158] print(tokenizer.index_word)
```

```
{1: 'barber', 2: 'secret', 3: 'huge', 4: 'kept', 5: 'person', 6: 'word', 7: 'keeping', 8: 'good', 9: 'knew', 10: 'driving', 11: 'crazy', 12: 'went', 13: 'mountain'}
```

```
[159] print(tokenizer.word_counts)
```

```
OrderedDict([('barber', 8), ('person', 3), ('good', 1), ('huge', 5), ('knew', 1), ('secret', 6), ('kept', 4), ('word', 2), ('keeping', 2), ('driving', 1), ('crazy', 1), ('went', 1), ('mountain', 1
```

```
[160] print(tokenizer.texts_to_sequences(sentences))
```

```
[[1, 5], [1, 8, 5], [1, 3, 5], [9, 2], [2, 4, 3, 2], [3, 2], [1, 4, 6], [1, 4, 6], [1, 4, 2], [7, 7, 3, 2, 10, 1, 11], [1, 12, 3, 13]]
```

1. 정수 인코딩(Integer Encoding)

- 정수 인코딩 방법 5) 케라스(Keras)의 텍스트 전처리
  - tokenizer = Tokenizer(num_words = vocab_size + 1)
  - most_common()과 같은 기능임
  - keras tokenizer가 숫자 0까지 단어 집합의 크기로

```
[112] vocab_size = 5
      tokenizer = Tokenizer(num_words = vocab_size + 1) # 상위 5개 단어만 사용
      tokenizer.fit_on_texts(sentences)
```

```
[113] print(tokenizer.word_index)

      {'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5, 'word': 6, 'keeping': 7, 'good': 8, 'knew': 9, 'driving': 10, 'crazy': 11, 'went': 12, 'mountain': 13}
```

```
[114] print(tokenizer.word_counts)

      OrderedDict([('barber', 8), ('person', 3), ('good', 1), ('huge', 5), ('knew', 1), ('secret', 6), ('kept', 4), ('word', 2), ('keeping', 2), ('driving', 1), ('crazy', 1), ('went', 1), ('mountain', 1)
```

```
[115] print(tokenizer.texts_to_sequences(sentences))

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

1. 정수 인코딩(Integer Encoding)

◆ 정수 인코딩 방법 5) 케라스(Keras)의 텍스트 전처리

- word_counts, word_index를 출력해도 같은 결과가 나오지만, text_to_sequences()를 하면 적용이 됨

```
[116] tokenizer = Tokenizer() # num_words를 여기서는 지정하지 않은 상태
      tokenizer.fit_on_texts(sentences)
```

```
[117] vocab_size = 5
      words_frequency = [w for w,c in tokenizer.word_index.items() if c >= vocab_size + 1] # 인덱스가 5 초과인 단어 제거
      for w in words_frequency:
          del tokenizer.word_index[w] # 해당 단어에 대한 인덱스 정보를 삭제
          del tokenizer.word_counts[w] # 해당 단어에 대한 카운트 정보를 삭제
      print(tokenizer.word_index)
      print(tokenizer.word_counts)
      print(tokenizer.texts_to_sequences(sentences))
```

```
{'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5}
OrderedDict([('barber', 8), ('person', 3), ('huge', 5), ('secret', 6), ('kept', 4)])
[[1, 5], [1, 5], [1, 3, 5], [2], [2, 4, 3, 2], [3, 2], [1, 4], [1, 4], [1, 4, 2], [3, 2, 1], [1, 3]]
```

1. 정수 인코딩(Integer Encoding)

◆ 정수 인코딩 방법 5) 케라스(Keras)의 텍스트 전처리

- keras tokenizer는 정수 인코딩 과정 중 단어 집합에 없는 단어(OOV)에 대해서는 아예 해당 단어를 제거해버림
- tokenizer = Tokenizer(num_words = vocab_size + 2, oov token = 'OOV')

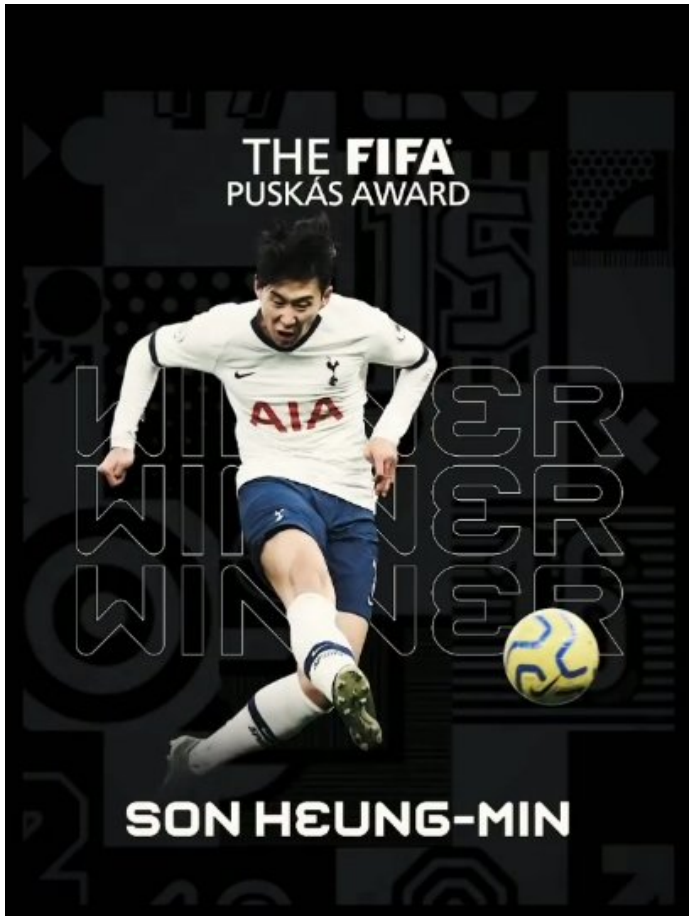
```
[118] vocab_size = 5
      tokenizer = Tokenizer(num_words = vocab_size + 2, oov_token = 'OOV')
      # 빈도수 상위 5개 단어만 사용. 숫자 0과 OOV를 고려해서 단어 집합의 크기는 +2
      tokenizer.fit_on_texts(sentences)
```

```
[119] print('단어 OOV의 인덱스 : {}'.format(tokenizer.word_index['OOV']))

      단어 OOV의 인덱스 : 1
```

```
[120] print(tokenizer.texts_to_sequences(sentences))

      [[2, 6], [2, 1, 6], [2, 4, 6], [1, 3], [3, 5, 4, 3], [4, 3], [2, 5, 1], [2, 5, 1], [2, 5, 3], [1, 1, 4, 3, 1, 2, 1], [2, 1, 4, 1]]
```

1. 정수 인코딩(Integer Encoding)

- 실습 : 유튜브 동영상(손흥민 푸스카스상) 댓글 정수 인코딩



https://www.youtube.com/watch?v=2SATIjljXCY&list=PLCGIzmTE4d0gdzAp_Q2y_yrZBKvbWMzbX

1. 정수 인코딩(Integer Encoding)

◆ 실습 : 유튜브 동영상(손흥민 푸스카스상) 댓글 정수 인코딩

- 동영상 댓글 크롤링 (3929개)

In [14]:
```
# 손흥민 골 댓글
dfs[9].head(20)
```

Out[14]:

	0	text	vote	time
0	{"cid": "UgxydcFfKsrXwwH1X-N4AaABAg", "text": ...	"금호타이어 광고 제대로 했네"	"0"	"3주 전"
1	{"cid": "UgxQZfMv3XCAFAHAUbN4AaABAg", "text": ...	"What a amazing goal"	"0"	"3주 전"
2	{"cid": "UgyRHAhvQqg6zil2hLR4AaABAg", "text": ...	"Rancak Bana..!!!"	"0"	"3주 전"
3	{"cid": "UgyILPpkWGthnm8nAOt4AaABAg", "text": ...	"Show...!!!"	"0"	"3주 전"
4	{"cid": "UgzsZCepVjnMouQ8WjF4AaABAg", "text": ...	"Well deserved"	"0"	"2주 전"
5	{"cid": "Ugyz1nSywaPSpWHo9T94AaABAg", "text": ...	"Superrr"	"0"	"2주 전"
6	{"cid": "Ugxl_0aVVMGUj9zT0fl4AaABAg", "text": ...	"Barcelona vs PSG Messi rocket 🚀"	"0"	"2주 전"
7	{"cid": "UgzeS6lVo4SpWTNenFN4AaABAg", "text": ...	"👏👏👏👏👏👏👏👏"	"0"	"2주 전"
8	{"cid": "UgwOEBCb9lMLg76czcJ4AaABAg", "text": ...	"Suarez snd Arrascaetas were both much better ...	"0"	"2주 전(수정됨)"
9	{"cid": "Ugxknx0_gv--W7jza6F4AaABAg", "text": ...	"Better than Maradona"	"0"	"2주 전"
10	{"cid": "UgxCAiciw6jidKDXZfN4AaABAg", "text": ...	"아시아 최고의 축구 선수"	"0"	"2주 전"
11	{"cid": "Ugzy-KnVawoR9G1g_5l4AaABAg", "text": ...	"VİDA"	"0"	"2주 전"
12	{"cid": "UgwQv_kaVR6w-B0hFB94AaABAg", "text": ...	"In any other country he would have been unfai...	"2"	"2주 전"
13	{"cid": "UgxTMw94B4L_re8Lnvd4AaABAg", "text": ...	"Nice one"	"0"	"2주 전"
14	{"cid": "Ugybribs6TNtThQ9SQR4AaABAg", "text": ...	"😆👍☄🔥🔥🔥⚽🤹"	"0"	"1주 전"
15	{"cid": "Ugy9N7fLIKbSJDFDWxt4AaABAg", "text": ...	"Son needs to leave Tottenham or he will never...	"0"	"1주 전(수정됨)"
16	{"cid": "Ugwnlkx6LRu7H5KsC094AaABAg", "text": ...	"O cara canta sbt,joga na Inglaterra e ainda g...	"3"	"1주 전"
17	{"cid": "Ugw8h6qzunzVe8mELaB4AaABAg", "text": ...	"Era de suarez para mi pero no puedo negar que...	"0"	"6일 전"
18	{"cid": "UgwQgGK5y0LNutoVOG14AaABAg", "text": ...	"OGM he is world class!"	"0"	"1일 전"
19	{"cid": "UgwnVbiHxi5CDNqif_p4AaABAg", "text": ...	"amazing goal but didn't deserve a puskas award"	"0"	"14시간 전"

1. 정수 인코딩(Integer Encoding)

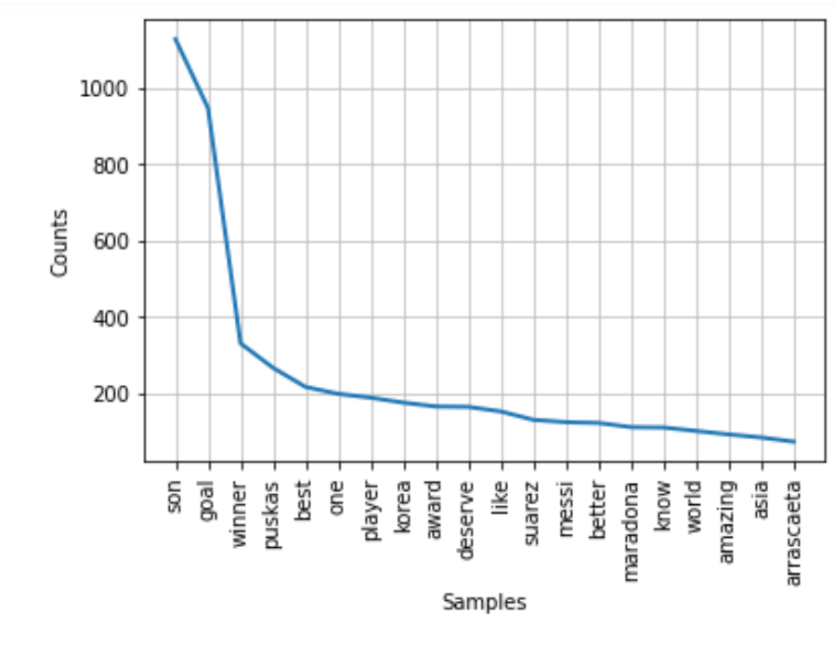
- 실습 : 유튜브 동영상(손흥민 푸스카스상) 댓글 정수 인코딩
  - 간단한 텍스트 전처리
  - NLTK의 word_tokenize()를 활용하여 토큰화를 수행함
  - 단어의 토큰화는 단어 전체, 명사, 형용사, bigram, trigram에 대하여 진행함
  - 불용어 제거 (영어, 특수문자, 사용자 정의)
  - 대소문자 통일 및 단어 길이 2이하 제거
  - 빈도수 TOP 100의 결과를 비교하여 단어 치환을 수행함
    (유사단어, 고유명사, 복수형 등)
    (예시)
    son heung min, sonny, sonaldo, ... → son
    deserved, deserve, ... → deserve
    jumo, south korea, korean, ... → korea
    girl, woman, female, ... → women 등

1. 정수 인코딩(Integer Encoding)

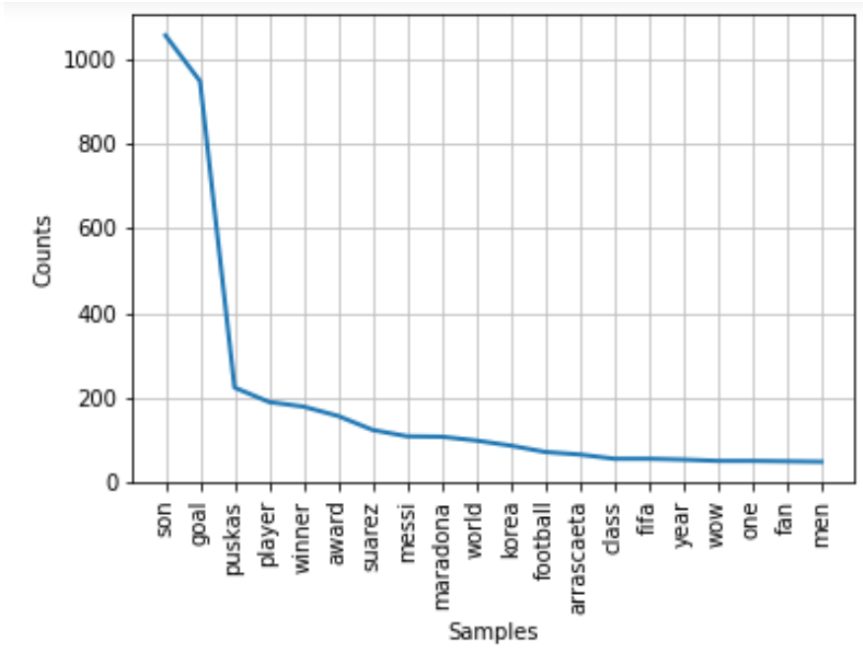
◆ 실습 : 유튜브 동영상(손흥민 푸스카스상) 댓글 정수 인코딩

- 댓글 빈도수 분석

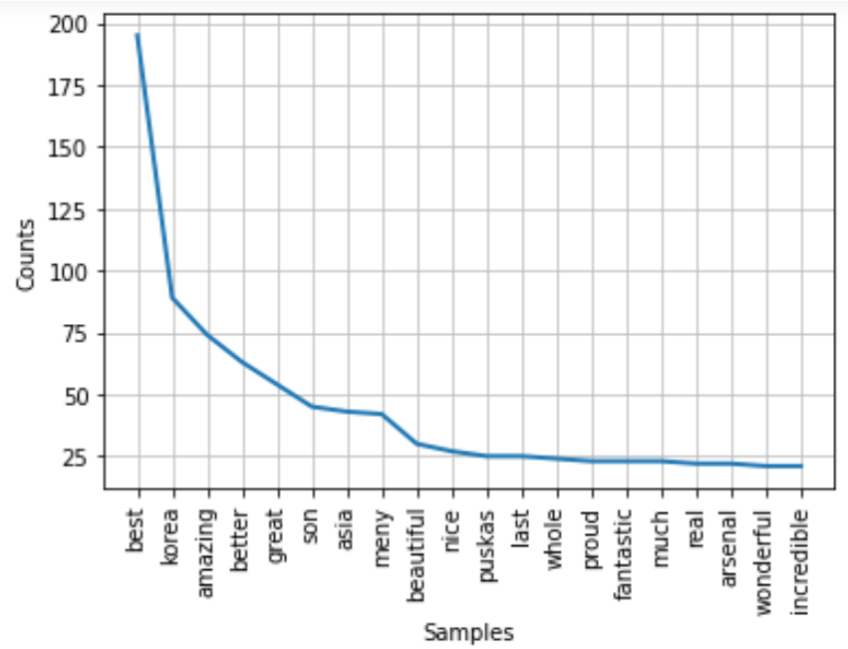
모든 단어(unigram) TOP 20



명사 TOP 20



형용사 TOP 20



1. 정수 인코딩(Integer Encoding)

◆ 실습 : 유튜브 동영상(손흥민 푸스카스상) 댓글 정수 인코딩

- 댓글 빈도수 분석
- 손흥민을 지칭하는 단어들이 가장 많이 나옴
- winner, best, award, deserve가 높게 나타나는 것을 보아, 손흥민이 1등을 했다는 것을 짐작해볼 수 있음
- korea, asia가 높은 것을 보아 손흥민의 국적을 알 수 있음
- messi, maradona를 보아 손흥민이 메시, 마라도나를 연상시킨다는 말이 나왔음을 짐작해봄

1. 정수 인코딩(Integer Encoding)

- 실습 : 유튜브 동영상(손흥민 푸스카스상) 댓글 정수 인코딩
  - 워드 클라우드

1. 정수 인코딩(Integer Encoding)

- 실습 : 유튜브 동영상(손흥민 푸스카스상) 댓글 정수 인코딩
  - 정수 인코딩(NLTK의 FreqDist)
  - 약 4000개의 단어 중에서 빈도수가 3 이상인 단어의 수를 num_words(=1060)으로 설정함
  - 정밀한 전처리, 데이터 수집 등으로 단어 희소성을 줄여야 함

In [35]:

```
# 정수 인코딩 (NLTK)

vocab = top3_fd[0].most_common(1060)
word_to_index = {word[0] : index + 1 for index, word in enumerate(vocab)}
word_to_index['OOV'] = 0

sorted_list = sorted(word_to_index.items(), key=lambda item: item[1])
word_to_index = {k: v for k, v in sorted_list}

print(word_to_index)
```

```
{'OOV': 0, 'son': 1, 'goal': 2, 'winner': 3, 'puskas': 4, 'best': 5, 'one': 6, 'player': 7, 'korea': 8, 'award': 9, 'deserve': 10, 'like': 11, 'suarez': 12, 'messi': 13, 'better': 14, 'maradona': 15, 'know': 16, 'world': 17, 'amazing': 18, 'asia': 19, 'arrascaeta': 20, 'football': 21, 'scored': 22, 'fifa': 23, 'great': 24, 'wow': 25, 'even': 26, 'think': 27, 'get': 28, 'run': 29, 'would': 30, 'class': 31, 'lol': 32, 'year': 33, 'nice': 34, 'arsenal': 35, 'ball': 36, 'fan': 37, 'men': 38, 'see': 39, 'solo': 40, '2020': 41, 'ever': 42, 'burnley': 43, 'really': 44, 'vote': 45, 'doubt': 46, 'well': 47, 'meny': 48, 'proud': 49, 'super': 50, 'sead': 51, 'never': 52, 'team': 53, 'every': 54, 'kolašinac': 55, 'much': 56, 'yes': 57, 'could': 58, 'salah': 59, 'dribbling': 60, 'time': 61, 'fast': 62, 'way': 63, 'running': 64, 'got': 65, 'ronaldo': 66, 'beautiful': 67, 'seen': 68, 'newflow': 69, 'absolutely': 70, 'another': 71, 'love': 72, 'must': 73, 'gol': 74, 'skill': 75, 'say': 76, 'times': 77, 'speed': 78, 'league': 79, 'que': 80, 'legend': 81, 'congratulations': 82, 'still': 83, 'past': 84, 'play': 85, 'look': 86, 'definitely': 87, 'people': 88, 'two': 89, 'real': 90, 'whole': 91, 'dribble': 92, 'also': 93, 'last': 94, 'defenders': 95, 'score': 96, 'fantastic': 97, 'dont': 98, 'watch': 99, 'right': 100, 'everyone': 101, 'level': 102, 'overrated': 103, 'video': 104, 'let': 105, 'make': 106, 'give': 107, 'gon': 108, 'assist': 109, 'south': 110, 'crazy': 111, 'first': 112, 'winnerning': 113, 'made': 114, 'prize': 115, 'wonderful': 116, 'game': 117, 'tottenham': 118, 'easy': 119, 'winners': 120, 'fans': 121, 'back': 122, 'incredible': 123, 'kick': 124, 'no.1': 125, 'far': 126, 'without': 127, 'years': 128, 'spurs': 129, 'yeah': 130, 'sure': 131, 'del': 132, 'nothing': 133, "'ve": 134, '🔥🔥🔥': 135, 'defender': 136, 'come': 137, 'soccer': 138, 'dribbled': 139, 'nosonated': 140, 'century': 141, 'ran': 142, 'puskás': 143, 'crowd': 144, 'history': 145, 'congrats': 146, 'stop': 147, 'long': 148, 'else': 149, 'others': 150, 'agree': 151, 'mean': 152, 'style': 153, 'coys': 154, 'special': 155, 'box': 156, 'watched': 157, 'defending': 158, 'pace': 159, 'field': 160, 'king': 161, '.....': 162, 'anyone': 163, 'life': 164, 'diego': 165, 'next': 166, 'literally': 167, 'respect': 168, 'top': 169, 'maybe': 170, 'makes': 171, 'robbed': 172, 'something': 173, 'actually': 174, 'done': 175, 'nosonee': 176, 'already': 177, 'golazo': 178, 'second': 179, 'course': 180, 'brilliant': 181, 'premier': 182, 'similar': 183, 'voted': 184, 'week': 185, 'simply': 186, 'wonder': 187, 'baba': 188, 'sprint': 189, 'joke': 190, 'played': 191, 'believ
```

In [36]:

```
encoded = []
for s in top3_tokens[0]:
    temp = []
    for w in s:
        try:
            temp.append(word_to_index[w])  # 단어를 인덱스로 치환
        except:
            temp.append(word_to_index['OOV'])
    encoded.append(temp)

print(encoded)
```

```
[[0, 912], [18, 2], [0, 0], [294], [47, 10], [0], [244, 0, 13, 913], [0], [12, 0, 0, 56, 14, 2], [14, 15], [0, 0], [0], [266, 30, 0, 453, 679, 168, 454, 0, 0], [34, 6], [0], [1, 338, 0, 118, 52, 3, 267, 268, 53, 914, 7, 0], [0, 0, 0, 0, 680, 0, 0, 0, 0], [385, 12, 245, 269, 0, 0, 80, 222, 178], [0, 17, 31], [18, 2, 10, 4, 9], [0, 1], [1, 295, 681, 2, 339, 0, 109, 0, 246, 2, 915, 0], [0, 0, 0, 0], [0, 0], [916, 132, 682, 917, 0, 0], [918], [0], [1, 386], [0, 0, 0, 0, 0, 0, 0, 0], [296, 5, 109, 387], [20], [0, 0, 161, 19], [19, 125, 0], [12, 10], [1], [0, 919], [920, 0], [340, 6, 223, 0, 341, 8], [4, 9, 141], [297, 84, 89, 7, 99, 4, 102], [0], [683, 1], [109, 298, 0], [43, 224, 223, 342, 14, 53, 1, 103, 0, 225, 455], [0, 0, 0], [0], [2, 12, 2, 1], [299, 538, 94, 185, 539, 0, 921, 270, 922, 923, 684, 226], [25], [0, 0, 0, 0], [0], [0, 0], [1], [0, 685, 456], [154], [1, 540, 186, 0], [0, 0, 0, 0, 0, 0, 74, 0], [], [5, 126, 142, 300, 186, 0, 0, 686, 11, 95, 133, 155, 687, 924, 7], [], [0, 0, 0, 118, 1, 89, 2, 6, 109, 925, 457, 225, 541], [388], [0, 0, 54, 53, 30, 343, 85], [119, 54, 185, 7, 0, 458, 96, 11, 686, 11, 95, 2, 10, 4], [247, 344], [542, 688, 542, 1, 542, 542, 688, 688], [90, 227, 926, 38, 25], [0], [0, 0, 0, 0, 0, 0], [927, 928, 0, 11, 0, 927, 928, 0, 0, 0], [35, 81, 10, 9, 119, 0], [0, 389], [15, 2, 0], [10], [1, 120, 4, 60, 84, 95, 187, 48, 4, 13, 10], [82], [543, 14, 2, 17, 21, 91, 0], [0, 43, 0, 0, 0, 0, 0, 0, 689, 0, 0, 74, 0, 0], [343, 0, 1], [49, 50, 1, 0, 690], [1, 1, 929], [49, 19], [110, 8, 49, 1], [188, 1], [13, 11, 459, 0], [247, 344], [57], [691, 0, 544], [0, 0, 543, 545, 460, 461, 58, 143, 9], [204, 301, 156], [345, 13, 62, 1], [0, 2, 1], [346, 0, 0, 0], [1, 930], [32], [17, 692, 70, 17, 692], [1, 1, 0, 205, 5, 7], [24, 7, 18, 2], [1, 693], [0, 694, 302, 2, 35, 71, 546, 2], [0, 931, 13, 2], [72, 0], [1, 1, 19, 206, 8, 0, 118, 347, 19, 462, 271, 169, 17, 170, 6, 223, 462, 6, 17], [456, 695, 10], [14, 15, 2, 248], [12, 2, 14], [111, 0, 162], [932, 0, 0, 920, 0, 0, 0, 0, 932], [0, 933, 547, 463], [249, 104], [0], [0, 0], [0, 0], [303, 301, 156, 171, 390], [169, 17], [169, 696, 17], [32], [0], [1, 540], [81], [934, 934, 0], [0, 0, 189], [348, 935], [304, 105, 90, 349, 179, 11, 464, 272, 305, 112, 53, 21, 387, 91, 17, 391, 189, 2, 0, 3, 4], [0, 0], [113, 1], [100, 24, 2, 4, 9], [5, 2, 306], [0, 697, 936, 0, 12, 0, 0], [350, 1, 1, 392, 1, 1, 49, 1, 1, 1, 937, 190], [0], [465, 393, 172], [938, 30, 0], [34, 6, 1, 34, 6, 0, 6, 1, 939, 71, 0, 0], [191, 0, 351, 0], [0], [8, 9
```

1. 정수 인코딩(Integer Encoding)

◆ 실습 : 유튜브 동영상(손흥민 푸스카스상) 댓글 정수 인코딩

- 정수 인코딩(Keras tokenizer)

In [37]:
```
# 정수 인코딩 (keras tokenizer)

from tensorflow.keras.preprocessing.text import Tokenizer

vocab_size = 1060
tokenizer = Tokenizer(num_words = vocab_size + 2, oov_token = 'OOV')
tokenizer.fit_on_texts(top3_tokens[0])
print(tokenizer.word_index)
```

```
{'OOV': 1, 'son': 2, 'goal': 3, 'winner': 4, 'puskas': 5, 'best': 6, 'one': 7, 'player': 8, 'korea': 9, 'award': 10, 'deserve': 11,
'like': 12, 'suarez': 13, 'messi': 14, 'better': 15, 'maradona': 16, 'know': 17, 'world': 18, 'amazing': 19, 'asia': 20, 'arrascaet
a': 21, 'football': 22, 'scored': 23, 'fifa': 24, 'great': 25, 'wow': 26, 'even': 27, 'think': 28, 'get': 29, 'run': 30, 'would': 31,
'class': 32, 'lol': 33, 'year': 34, 'nice': 35, 'arsenal': 36, 'ball': 37, 'fan': 38, 'men': 39, 'see': 40, 'solo': 41, '2020': 42,
'ever': 43, 'burnley': 44, 'really': 45, 'vote': 46, 'doubt': 47, 'well': 48, 'meny': 49, 'proud': 50, 'super': 51, 'sead': 52, 'neve
r': 53, 'team': 54, 'every': 55, 'kolašinac': 56, 'much': 57, 'yes': 58, 'could': 59, 'salah': 60, 'dribbling': 61, 'time': 62, 'fas
t': 63, 'way': 64, 'running': 65, 'got': 66, 'ronaldo': 67, 'beautiful': 68, 'seen': 69, 'newflow': 70, 'absolutely': 71, 'another':
72, 'love': 73, 'must': 74, 'gol': 75, 'skill': 76, 'say': 77, 'times': 78, 'speed': 79, 'league': 80, 'que': 81, 'legend': 82, 'cong
ratulations': 83, 'still': 84, 'past': 85, 'play': 86, 'look': 87, 'definitely': 88, 'people': 89, 'two': 90, 'real': 91, 'whole': 9
2, 'dribble': 93, 'also': 94, 'last': 95, 'defenders': 96, 'score': 97, 'fantastic': 98, 'dont': 99, 'watch': 100, 'right': 101, 'eve
ryone': 102, 'level': 103, 'overrated': 104, 'video': 105, 'let': 106, 'make': 107, 'give': 108, 'gon': 109, 'assist': 110, 'south':
111, 'crazy': 112, 'first': 113, 'winnerning': 114, 'made': 115, 'prize': 116, 'wonderful': 117, 'game': 118, 'tottenham': 119, 'eas
```

In [39]:
```
res = sorted(tokenizer.word_counts.items(), key=lambda x:x[1], reverse=True)
print(res)
```

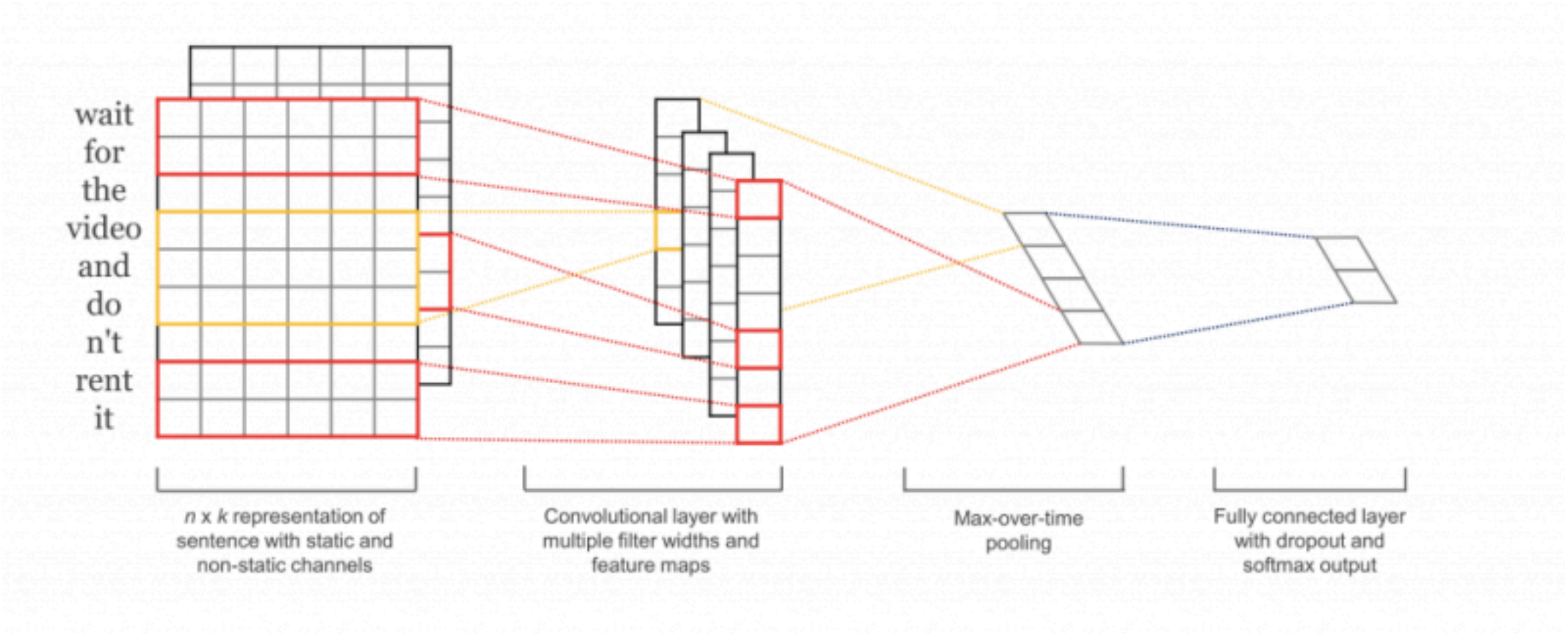
```
[('son', 1127), ('goal', 947), ('winner', 332), ('puskas', 268), ('best', 218), ('one', 200), ('player', 190), ('korea', 177), ('awar
d', 167), ('deserve', 166), ('like', 154), ('suarez', 132), ('messi', 126), ('better', 124), ('maradona', 113), ('know', 112), ('worl
d', 103), ('amazing', 94), ('asia', 86), ('arrascaeta', 75), ('football', 73), ('scored', 69), ('fifa', 68), ('great', 67), ('wow', 6
5), ('even', 65), ('think', 65), ('get', 64), ('run', 62), ('would', 57), ('class', 56), ('lol', 54), ('year', 54), ('nice', 52), ('a
rsenal', 52), ('ball', 51), ('fan', 50), ('men', 49), ('see', 49), ('solo', 48), ('2020', 47), ('ever', 46), ('burnley', 44), ('reall
y', 44), ('vote', 43), ('doubt', 43), ('well', 42), ('meny', 42), ('proud', 42), ('super', 42), ('sead', 42), ('never', 41), ('team',
41), ('every', 40), ('kolašinac', 40), ('much', 39), ('yes', 38), ('could', 38), ('salah', 38), ('dribbling', 37), ('time', 37), ('fa
st', 36), ('way', 36), ('running', 36), ('got', 35), ('ronaldo', 35), ('beautiful', 32), ('seen', 32), ('newflow', 32), ('absolutel
y', 31), ('another', 31), ('love', 31), ('must', 31), ('gol', 30), ('skill', 30), ('say', 30), ('times', 29), ('speed', 29), ('leagu
e', 29), ('que', 28), ('legend', 28), ('congratulations', 28), ('still', 28), ('past', 27), ('play', 27), ('look', 27), ('definitel
y', 27), ('people', 27), ('two', 26), ('real', 26), ('whole', 26), ('dribble', 26), ('also', 26), ('last', 25), ('defenders', 25),
('score', 25), ('fantastic', 25), ('dont', 25), ('watch', 24), ('right', 24), ('everyone', 24), ('level', 23), ('overrated', 23), ('v
ideo', 23), ('let', 23), ('make', 23), ('give', 23), ('gon', 23), ('assist', 22), ('south', 22), ('crazy', 22), ('first', 22), ('winn
erning', 22), ('made', 22), ('prize', 22), ('wonderful', 22), ('game', 22), ('tottenham', 21), ('easy', 21), ('winners', 21), ('fan
s', 21), ('back', 21), ('incredible', 21), ('kick', 21), ('no.1', 20), ('far', 20), ('without', 20), ('years', 20), ('spurs', 20),
('yeah', 20), ('sure', 20), ('del', 19), ('nothing', 19), ("'ve", 19), ('🔥🔥🔥', 19), ('defender', 19), ('come', 19), ('soccer', 1
9), ('dribbled', 19), ('nosonated', 19), ('century', 18), ('ran', 18), ('puskás', 18), ('crowd', 18), ('history', 18), ('congrats', 1
8), ('stop', 18), ('long', 18), ('else', 18), ('others', 18), ('agree', 18), ('mean', 18), ('style', 18), ('coys', 17), ('special', 1
7), ('box', 17), ('watched', 17), ('defending', 17), ('pace', 17), ('field', 17), ('king', 16), ('.....', 16), ('anyone', 16), ('lif
```

In [41]:
```
print(tokenizer.texts_to_sequences(top3_tokens[0]))
```

```
[[1, 913], [19, 3], [1, 1], [295], [48, 11], [1], [245, 1, 14, 914], [1], [13, 1, 1, 57, 15, 3], [15, 16], [1, 1], [1], [267, 31, 1,
454, 680, 169, 455, 1, 1], [35, 7], [1], [2, 339, 1, 119, 53, 4, 268, 269, 54, 915, 8, 1], [1, 1, 1, 1, 681, 1, 1, 1, 1], [386, 13, 2
46, 270, 1, 1, 81, 223, 179], [1, 18, 32], [19, 3, 11, 5, 10], [1, 2], [2, 296, 682, 3, 340, 1, 110, 1, 247, 3, 916, 1], [1, 1, 1,
1], [1, 1], [917, 133, 683, 918, 1, 1], [919], [1], [2, 387], [1, 1, 1, 1, 1, 1, 1, 1], [297, 6, 110, 388], [21], [1, 1, 162, 20], [2
0, 126, 1], [13, 11], [2], [1, 920], [921, 1], [341, 7, 224, 1, 342, 9], [5, 10, 142], [298, 85, 90, 8, 100, 5, 103], [1], [684, 2],
[110, 299, 1], [44, 225, 224, 343, 15, 54, 2, 104, 1, 226, 456], [1, 1, 1], [1], [3, 13, 3, 2], [300, 539, 95, 186, 540, 1, 922, 271,
923, 924, 685, 227], [26], [1, 1, 1, 1], [1], [1, 1], [2], [1, 686, 457], [155], [2, 541, 187, 1], [1, 1, 1, 1, 1, 1, 75, 1], [], [6,
127, 143, 301, 187, 1, 1, 687, 12, 96, 134, 156, 688, 925, 8], [], [1, 1, 1, 119, 2, 90, 3, 7, 110, 926, 458, 226, 542], [389], [1,
1, 55, 54, 31, 344, 86], [120, 55, 186, 8, 1, 459, 97, 12, 687, 12, 96, 3, 11, 5], [248, 345], [543, 689, 543, 2, 543, 543, 689, 68
9], [91, 228, 927, 39, 26], [1], [1, 1, 1, 1, 1], [928, 929, 1, 12, 1, 928, 929, 1, 1, 1], [36, 82, 11, 10, 120, 1], [1, 390], [16,
3, 1], [11], [2, 121, 5, 61, 85, 96, 188, 49, 5, 14, 11], [83], [544, 15, 3, 18, 22, 92, 1], [1, 44, 1, 1, 1, 1, 1, 690, 1, 1, 75, 1,
1], [344, 1, 2], [50, 51, 2, 1, 691], [2, 2, 930], [50, 20], [111, 9, 50, 2], [189, 2], [14, 12, 460, 1], [248, 345], [58], [692, 1,
545], [1, 1, 544, 546, 461, 462, 59, 144, 10], [205, 302, 157], [346, 14, 63, 2], [1, 3, 2], [347, 1, 1, 1], [2, 931], [33], [18, 69
3, 71, 18, 693], [2, 2, 1, 206, 6, 8], [25, 8, 19, 3], [2, 694], [1, 695, 303, 3, 36, 72, 547, 3], [1, 932, 14, 3], [73, 1], [2, 2, 2
0, 207, 9, 1, 119, 348, 20, 463, 272, 170, 18, 171, 7, 224, 463, 7, 18], [457, 696, 11], [15, 16, 3, 249], [13, 3, 15], [112, 1, 16
3], [933, 1, 1, 921, 1, 1, 1, 1, 933], [1, 934, 548, 464], [250, 105], [1], [1, 1], [1, 1], [304, 302, 157, 172, 391], [170, 18], [17
0, 697, 18], [33], [1], [2, 541], [82], [935, 935, 1], [1, 1, 190], [349, 936], [305, 106, 91, 350, 180, 12, 465, 273, 306, 113, 54,
22, 388, 92, 18, 392, 190, 3, 1, 4, 5], [1, 1], [114, 2], [101, 25, 3, 5, 10], [6, 3, 307], [1, 698, 937, 1, 13, 1, 1], [351, 2, 2, 3
93, 2, 2, 50, 2, 2, 2, 938, 191], [1], [466, 394, 173], [939, 31, 1], [35, 7, 2, 35, 7, 1, 7, 2, 940, 72, 1, 1], [192, 1, 352, 1],
[1, 9, 91], [941, 549, 1, 3, 12, 394, 1, 3, 174, 942, 40, 550], [943, 1], [944, 145, 1, 1, 193], [1, 12, 1], [699, 395, 700, 66, 1
```

2. 패딩(Padding)

- 왜 패딩을 하지?
  자연어 처리시 동일 길이를 묶어서 연산하기 좋음



2. 패딩(Padding)

◆ Numpy로 패딩

```
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
```

```
sentences = [['barber', 'person'], ['barber', 'good', 'person'], ['barber', 'huge', 'person'], ['knew', 'secret'], ['
secret', 'kept', 'huge', 'secret'], ['huge', 'secret'], ['barber', 'kept', 'word'], ['barber', 'kept', 'word'], ['bar
ber', 'kept', 'secret'], ['keeping', 'keeping', 'huge', 'secret', 'driving', 'barber', 'crazy'], ['barber', 'went', '
huge', 'mountain']]
```

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences) # fit_on_texts()안에 코퍼스를 입력으로 하면 빈도수를 기준으로 단어 집합을 생성한다.
```

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

```
[[1, 5], [1, 8, 5], [1, 3, 5], [9, 2], [2, 4, 3, 2], [3, 2], [1, 4, 6], [1, 4, 6], [1, 4, 2], [7, 7, 3, 2, 10, 1, 1
1], [1, 12, 3, 13]]
```

2. 패딩(Padding)

◆ Numpy로 패딩

```
max_len = max(len(item) for item in encoded)
print(max_len)
```

```
for item in encoded: # 각 문장에 대해서
    while len(item) < max_len:   # max_len보다 작으면
        item.append(0)

padded_np = np.array(encoded)
padded_np
```

```
array([[ 1,  5,  0,  0,  0,  0,  0],
       [ 1,  8,  5,  0,  0,  0,  0],
       [ 1,  3,  5,  0,  0,  0,  0],
       [ 9,  2,  0,  0,  0,  0,  0],
       [ 2,  4,  3,  2,  0,  0,  0],
       [ 3,  2,  0,  0,  0,  0,  0],
       [ 1,  4,  6,  0,  0,  0,  0],
       [ 1,  4,  6,  0,  0,  0,  0],
       [ 1,  4,  2,  0,  0,  0,  0],
       [ 7,  7,  3,  2, 10,  1, 11],
       [ 1, 12,  3, 13,  0,  0,  0]])
```

2. 패딩(Padding)

◆ 케라스 전처리 도구로 패딩하기

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

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

```
[[1, 5], [1, 8, 5], [1, 3, 5], [9, 2], [2, 4, 3, 2], [3, 2], [1, 4, 6], [1, 4, 6], [1, 4, 2], [7, 7, 3, 2, 10, 1, 1
1], [1, 12, 3, 13]]
```

```
padded = pad_sequences(encoded)
padded
```

```
array([[ 0,  0,  0,  0,  0,  1,  5],
       [ 0,  0,  0,  0,  1,  8,  5],
       [ 0,  0,  0,  0,  1,  3,  5],
       [ 0,  0,  0,  0,  0,  9,  2],
       [ 0,  0,  0,  2,  4,  3,  2],
       [ 0,  0,  0,  0,  0,  3,  2],
       [ 0,  0,  0,  0,  1,  4,  6],
       [ 0,  0,  0,  0,  1,  4,  6],
       [ 0,  0,  0,  0,  1,  4,  2],
       [ 7,  7,  3,  2, 10,  1, 11],
       [ 0,  0,  0,  1, 12,  3, 13]], dtype=int32)
```

2. 패딩(Padding)

- 케라스 전처리 도구로 패딩하기 (post padding)

```
padded = pad_sequences(encoded, padding = 'post')
padded
```

```
array([[ 1,  5,  0,  0,  0,  0,  0],
       [ 1,  8,  5,  0,  0,  0,  0],
       [ 1,  3,  5,  0,  0,  0,  0],
       [ 9,  2,  0,  0,  0,  0,  0],
       [ 2,  4,  3,  2,  0,  0,  0],
       [ 3,  2,  0,  0,  0,  0,  0],
       [ 1,  4,  6,  0,  0,  0,  0],
       [ 1,  4,  6,  0,  0,  0,  0],
       [ 1,  4,  2,  0,  0,  0,  0],
       [ 7,  7,  3,  2, 10,  1, 11],
       [ 1, 12,  3, 13,  0,  0,  0]], dtype=int32)
```

```
(padded == padded_np).all()
```

2. 패딩(Padding)

◆ 케라스 전처리 도구로 패딩하기 (max len)

```
padded = pad_sequences(encoded, padding = 'post', maxlen = 5)
padded
```

```
array([[ 1,  5,  0,  0,  0],
       [ 1,  8,  5,  0,  0],
       [ 1,  3,  5,  0,  0],
       [ 9,  2,  0,  0,  0],
       [ 2,  4,  3,  2,  0],
       [ 3,  2,  0,  0,  0],
       [ 1,  4,  6,  0,  0],
       [ 1,  4,  6,  0,  0],
       [ 1,  4,  2,  0,  0],
       [ 3,  2, 10,  1, 11],
       [ 1, 12,  3, 13,  0]], dtype=int32)
```

2. 패딩(Padding)

◆ 케라스 전처리 도구로 패딩하기 (last index)

```
last_value = len(tokenizer.word_index) + 1 # 단어 집합의 크기보다 1 큰 숫자를 사용
print(last_value)
```

```
padded = pad_sequences(encoded, padding = 'post', value = last_value)
padded
```

```
array([[ 1,  5, 14, 14, 14, 14, 14],
       [ 1,  8,  5, 14, 14, 14, 14],
       [ 1,  3,  5, 14, 14, 14, 14],
       [ 9,  2, 14, 14, 14, 14, 14],
       [ 2,  4,  3,  2, 14, 14, 14],
       [ 3,  2, 14, 14, 14, 14, 14],
       [ 1,  4,  6, 14, 14, 14, 14],
       [ 1,  4,  6, 14, 14, 14, 14],
       [ 1,  4,  2, 14, 14, 14, 14],
       [ 7,  7,  3,  2, 10,  1, 11],
       [ 1, 12,  3, 13, 14, 14, 14]], dtype=int32)
```

3. 원-핫 인코딩(One-Hot Encoding)

- 컴퓨터 연산 효율을 위해 문자 보다는 숫자로 처리
- Vocabulary: 단어 집합
- 원-핫 인코딩: 표현하고 싶은 단어 인덱스는 1, 그 외 단어는 0으로 표현, 원-핫 벡터(One-Hot vector)

3. 원-핫 인코딩(One-Hot Encoding)

```
from konlpy.tag import Okt
okt=Okt()
token=okt.morphs("나는 자연어 처리를 배운다")
print(token)
```

```
['나', '는', '자연어', '처리', '를', '배운다']
```

```
def one_hot_encoding(word, word2index):
        one_hot_vector = [0]*(len(word2index))
        index=word2index[word]
        one_hot_vector[index]=1
        return one_hot_vector
```

```
one_hot_encoding("자연어",word2index)
```

```
[0, 0, 1, 0, 0, 0]
```

3. 원-핫 인코딩(One-Hot Encoding)

◆ 케라스를 이용한 원-핫 인코딩

```
text="나랑 점심 먹으러 갈래 점심 메뉴는 햄버거 갈래 갈래 햄버거 최고야"
```

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import to_categorical

text="나랑 점심 먹으러 갈래 점심 메뉴는 햄버거 갈래 갈래 햄버거 최고야"

t = Tokenizer()
t.fit_on_texts([text])
print(t.word_index) # 각 단어에 대한 인코딩 결과 출력.
```

```
{'갈래': 1, '점심': 2, '햄버거': 3, '나랑': 4, '먹으러': 5, '메뉴는': 6, '최고야': 7}
```

```
sub_text="점심 먹으러 갈래 메뉴는 햄버거 최고야"
encoded=t.texts_to_sequences([sub_text])[0]
print(encoded)
```

```
[2, 5, 1, 6, 3, 7]
```

3. 원-핫 인코딩(One-Hot Encoding)

◆ 케라스를 이용한 원-핫 인코딩

```
one_hot = to_categorical(encoded)
print(one_hot)
```

```
[[0. 0. 1. 0. 0. 0. 0. 0.] #인덱스 2의 원-핫 벡터
 [0. 0. 0. 0. 0. 1. 0. 0.] #인덱스 5의 원-핫 벡터
 [0. 1. 0. 0. 0. 0. 0. 0.] #인덱스 1의 원-핫 벡터
 [0. 0. 0. 0. 0. 0. 1. 0.] #인덱스 6의 원-핫 벡터
 [0. 0. 0. 1. 0. 0. 0. 0.] #인덱스 3의 원-핫 벡터
 [0. 0. 0. 0. 0. 0. 0. 1.]] #인덱스 7의 원-핫 벡터
```

3. 원-핫 인코딩(One-Hot Encoding): 한계

- 단어의 갯수 만큼 저장 공간 필요
- 단어 유사도를 표현하지 못함
- 카운트 기반 벡터화 방법: LSA, HAL
- 예측 기반 벡터화: NNLM, RNNLM, Word2Vec, FastText
- 카운트 기반 + 예측 기반 벡터화: GloVe

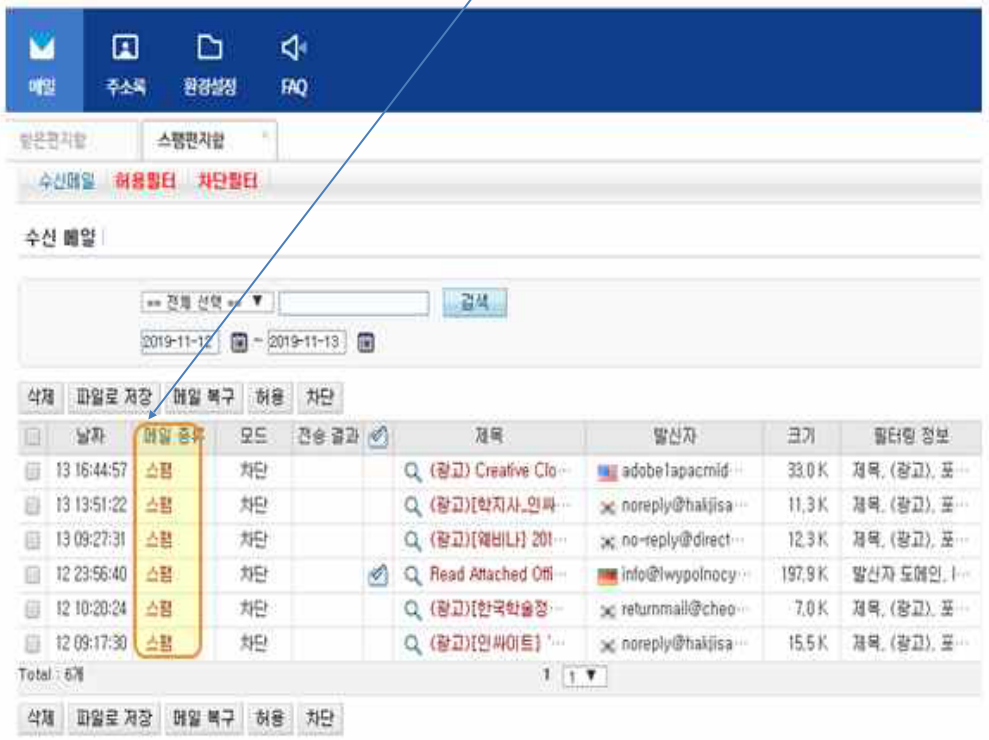
4. 데이터의 분리(Splitting Data)

◆ 지도 학습

정답지[스팸 여부]

문제지[메일 내용]

- 훈련 데이터(training set)을 통해 컴퓨터를 학습시키고, 이를 바탕으로 새로운 데이터(test data)를 분류/예측하는 방식
- 훈련 데이터에 '레이블(label) – 원하는 답'이 포함됨



"어떻게 새 메일이 '스팸'으로 분류되는가?"

4. 데이터의 분리(Splitting Data)

(예시) 20,000개의 행을 가진 데이터의 분리

텍스트(메일의 내용) x	레이블(스팸 여부) y
당신에게 드리는 마지막 혜택! ...	스팸 메일
내일 뵐 수 있을지 확인 부탁...	정상 메일
...	...
(광고) 멋있어질 수 있는...	스팸 메일

분리

<훈련 데이터> [학습용: 18,000개]

X_train : 문제지 데이터

y_train : 문제지에 대한 정답 데이터

<테스트 데이터> [시험용: 2,000개]

X_test : 시험지 데이터.

y_test : 시험지에 대한 정답 데이터.

- 문제지를 다 공부한 후 실력을 평가하기 위해 테스트용으로 일부 문제와 정답지를 빼놓는 것! (여기에서는 2,000개를 분리한다고 가정)

  - 분리시 x(문제지)와 y(정답지)의 맵핑 관계를 유지해야 함

- 기계는 훈련 데이터를 보면서 어떤 내용일 때 정상/스팸 메일인지 규칙 도출 → 학습 종료 후 X_test에 대해 정답 예측

- 기계가 예측한 답 vs. 실제 정답인 y_test 비교 → 기계의 **정확도(Accuracy)**

4. 데이터의 분리(Splitting Data)

◆ X와 Y 분리하기

1) zip 함수를 이용하는 방법

```
x,y = zip(['a',1], ['b', 2], ['c', 3])
```

```
print(x)
```

```
('a', 'b', 'c')
```

```
print(y)
```

```
(1, 2, 3)
```

- zip(): 동일한 개수를 갖는 시퀀스 자료형에서 각 순서에 등장하는 원소들끼리 묶어주는 역할
  → 리스트의 리스트 구성에서 x와 y 분리시 유용

4. 데이터의 분리(Splitting Data)

2) 데이터 프레임을 이용하는 방법

```
import pandas as pd
```

```
values = [['당신에게 드리는 마지막 혜택!', 1],
['내일 뵐 수 있을지 확인 부탁드...', 0],
['도연씨. 잘 지내시죠? 오랜만입...', 0],
['(광고) AI로 주가를 예측할 수 있다!', 1]]
columns = ['메일 본문', '스팸 메일 유무']
```

```
df = pd.DataFrame(values, columns=columns)
```

```
df
```

	메일 본문	스팸 메일 유무
0	당신에게 드리는 마지막 혜택!	1
1	내일 뵐 수 있을지 확인 부탁드...	0
2	도연씨. 잘 지내시죠? 오랜만입...	0
3	(광고) AI로 주가를 예측할 수 있다!	1

분리

```
x=df['메일 본문']
```

```
y=df['스팸 메일 유무']
```

```
print(x)
```

```
0          당신에게 드리는 마지막 혜택!
1      내일 뵐 수 있을지 확인 부탁드...
2      도연씨. 잘 지내시죠? 오랜만입...
3    (광고) AI로 주가를 예측할 수 있다!
Name: 메일 본문, dtype: object
```

```
print(y)
```

```
0    1
1    0
2    0
3    1
Name: 스팸 메일 유무, dtype: int64
```

- 데이터 프레임(df)은 열의 이름을 통해 각 열에 접근이 가능함을 이용

4. 데이터의 분리(Splitting Data)

3) Numpy를 이용하는 방법

```
import numpy as np
ar = np.arange(0,16).reshape((4,4))
print(ar)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
```

```
x=ar[:, :3]
print(x)
```

```
[[ 0  1  2]
 [ 4  5  6]
 [ 8  9 10]
 [12 13 14]]
```

```
y=ar[:,3]
print(y)
```

```
[ 3  7 11 15]
```

4. 데이터의 분리(Splitting Data)

◆ 테스트 데이터 분리하기

1) 사이킷 런을 이용하는 방법

```
import numpy as np
from sklearn.model_selection import train_test_split
x, y = np.arange(10).reshape((5, 2)), range(5) # 실습을 위해 임의로 x와 y가 이미 분리된 데이터 생성
print(x)
print(list(y)) # 레이블 데이터
```

```
[[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
[0, 1, 2, 3, 4]
```

1/3만 테스트 데이터로 지정(test_size, train_size 중 1개만 기재 가능)

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=1234)
print(x_train)
print(x_test)
```

x : 배열 또는 데이터프레임

랜덤하게 훈련 데이터와 테스트 데이터 분리

```
[[2 3]
 [4 5]
 [6 7]]
[[8 9]
 [0 1]]
```

```
print(y_train)
print(y_test)
```

y : 레이블

```
[1, 2, 3]
[4, 0]
```

- **Train_test_split():** 학습용 데이터와 테스트용 데이터 분리 함수

4. 데이터의 분리(Splitting Data)

2) 수동으로 분리하는 방법

```
import numpy as np
x, y = np.arange(0,24).reshape((12,2)), range(12)
print(x)
print(list(y))
```

```
[[ 0  1]
 [ 2  3]
 [ 4  5]
 [ 6  7]
 [ 8  9]
 [10 11]
 [12 13]
 [14 15]
 [16 17]
 [18 19]
 [20 21]
 [22 23]]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

```
n_of_train = int(len(x) * 0.8) # 데이터의 전체 길이의 80%에 해당하는 길이값 산출
n_of_test = int(len(x) - n_of_train) # 전체 길이에서 80%에 해당하는 길이를 뺀 값 산출
                                     # (소수점 처리로 인한 데이터 누락 방지 위해 0.2를 곱하지 않음에 유의)
print(n_of_train)
print(n_of_test)
```

```
9
3
```

분리

```
print(x_test)
print(list(y_test))
```

```
[[18 19]
 [20 21]
 [22 23]]
[9, 10, 11]
```

```
print(x_train)
print(list(y_train))
```

```
[[ 0  1]
 [ 2  3]
 [ 4  5]
 [ 6  7]
 [ 8  9]
 [10 11]
 [12 13]
 [14 15]
 [16 17]]
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

5. 한국어 전처리 패키지(Text Preprocessing Tools for Korean Text)

◆ PyKoSpacing

- 딥러닝 모델을 활용한 한국어 띄어쓰기 패키지
- Spacing() 활용

```
!pip install git+https://github.com/haven-jeon/PyKoSpacing.git
```

```
sent = '김철수는 극중 두 인격의 사나이 이광수 역을 맡았다. 철수는 한국 유일의 태권도 전승자를 가리는 결전의 날을 앞두고 10년간 함께 훈련한 사형인 유연재(김광수 분)를 찾으러 속세로 내려온 인물이다.'
```

```
new_sent = sent.replace(" ", '') # 띄어쓰기가 없는 문장 임의로 만들기
print(new_sent)
```

```
김철수는극중두인격의사나이이광수역을맡았다.철수는한국유일의태권도전승자를가리는결전의날을앞두고10년간함께훈련한사형인유연재(김광수분)를찾으러속세로내려온인물이다.
```

```
from pykospacing import Spacing
spacing = Spacing()
kospacing_sent = spacing(new_sent)

print(sent)
print(kospacing_sent)
```

```
김철수는 극중 두 인격의 사나이 이광수 역을 맡았다. 철수는 한국 유일의 태권도 전승자를 가리는 결전의 날을 앞두고 10년간 함께 훈련한 사형인 유연재(김광수 분)를 찾으러 속세로 내려온 인물이다.
김철수는 극중 두 인격의 사나이 이광수 역을 맡았다. 철수는 한국 유일의 태권도 전승자를 가리는 결전의 날을 앞두고 10년간 함께 훈련한 사형인 유연재(김광수 분)를 찾으러 속세로 내려온 인물이다.
```

5. 한국어 전처리 패키지(Text Preprocessing Tools for Korean Text)

◆ Py-Hanspell

- 네이버 한글 맞춤법 검사기를 활용한 한국어 맞춤법 +띄어쓰기 교정 패키지
- spell_checker.check() 활용

```
!pip install git+https://github.com/ssut/py-hanspell.git
```

```
from hanspell import spell_checker

sent = "맞춤법 틀리면 외 않되? 쓰고싶은대로쓰면돼지 "
spelled_sent = spell_checker.check(sent)

hanspell_sent = spelled_sent.checked
print(hanspell_sent)
```

맞춤법 틀리면 왜 안돼? 쓰고 싶은 대로 쓰면 되지

```
spelled_sent = spell_checker.check(new_sent)

hanspell_sent = spelled_sent.checked
print(hanspell_sent)
print(kospacing_sent) # 앞서 사용한 kospacing 패키지에서 얻은 결과
```

김철수는 극 중 두 인격의 사나이 이광수 역을 맡았다. 철수는 한국 유일의 태권도 전승자를 가리는 결전의 날을 앞두고 10년간 함께 훈련한 사형인 유연제(김광수 분)를 찾으러 속세로 내려온 인물이다.
김철수는 극중 두 인격의 사나이 이광수 역을 맡았다. 철수는 한국 유일의 태권도 전승자를 가리는 결전의 날을 앞두고 10년간 함께 훈련한 사형인 유연재(김광수 분)를 찾으러 속세로 내려온 인물이다.

5. 한국어 전처리 패키지(Text Preprocessing Tools for Korean Text)

◆ SOYNLP

- **비지도 학습**으로 데이터에 **자주 등장하는 단어를 단어로 추출**하는 단어 토크나이저
  - 기존 형태소 분석기가 미등록 단어(예: 신조어)를 제대로 분리하지 못하는 점 보완
  - **단어 점수 표**(응집 확률, 브랜칭 엔트로피 활용)를 바탕으로 동작

```
!pip install soynlp
```

<훈련 데이터 활용하여 학습한 후, 응집 확률 및 브랜칭 엔트로피 계산>

```
import urllib.request
from soynlp import DoublespaceLineCorpus
from soynlp.word import WordExtractor
```

```
urllib.request.urlretrieve("https://raw.githubusercontent.com/lovit/soynlp/master/tutorials/2016-10-20.txt", filename="2016-10-20.txt")
# 학습에 필요한 한국어 문서 다운로드
```

```
('2016-10-20.txt', <http.client.HTTPMessage at 0x7f6414117890>)
```

```
corpus = DoublespaceLineCorpus("2016-10-20.txt") # 훈련 데이터를 다수의 문서로 분리
len(corpus)
```

```
30091
```

5. 한국어 전처리 패키지(Text Preprocessing Tools for Korean Text)

- wordExtractor.extract() 활용: 전체 코퍼스에 대한 응집 확률, 브랜칭 엔트로피 **단어 점수표** 산출

```
word_extractor = WordExtractor()
word_extractor.train(corpus)
word_score_table = word_extractor.extract()
```

```
training was done. used memory 1.467 Gb
all cohesion probabilities was computed. # words = 223348
all branching entropies was computed # words = 361598
all accessor variety was computed # words = 361598
```

- 응집 확률(cohension probability): 문자열을 문자 단위로 분리하여 하위 문자열을 만드는 과정에서, 왼쪽부터 순서대로 문자를 추가하면서 각 문자열이 주어졌을 때 그 다음 문자가 나올 확률을 계산하여 누적곱을 한 값

$$\text{cohesion}(n) = \left(\prod_{i=1}^{n-1} P(c_{1:i+1} | c_{1:i}) \right)^{\frac{1}{n-1}}$$

→ **이 값이 클수록** 전체 코퍼스에서 **단어로 등장할 가능성이 높음**

5. 한국어 전처리 패키지(Text Preprocessing Tools for Korean Text)

$$\text{cohesion}(n) = \left(\prod_{i=1}^{n-1} P(c_{1:i+1} | c_{1:i}) \right)^{\frac{1}{n-1}}$$

- 예시) '반포한강공원에'(길이: 7)에 대해 각 하위 문자열의 응집 확률 구하기

- $cohesion(2) = P(\text{반포}|\text{반})$
- $cohesion(3) = \sqrt[2]{(P(\text{반포}|\text{반}) \cdot P(\text{반포한}|\text{반포}))}$
- $cohesion(4) = \sqrt[3]{(P(\text{반포}|\text{반}) \cdot P(\text{반포한}|\text{반포}) \cdot P(\text{반포한강}|\text{반포한}))}$
- $cohesion(5) = \sqrt[4]{(P(\text{반포}|\text{반}) \cdot P(\text{반포한}|\text{반포}) \cdot P(\text{반포한강}|\text{반포한}) \cdot P(\text{반포한강공}|\text{반포한강}))}$
- $cohesion(6) = \sqrt[5]{(P(\text{반포}|\text{반}) \cdot P(\text{반포한}|\text{반포}) \cdot P(\text{반포한강}|\text{반포한}) \cdot P(\text{반포한강공}|\text{반포한강}) \cdot P(\text{반포한강공원}|\text{반포한강공}))}$
- $cohesion(7) = \sqrt[5]{(P(\text{반포}|\text{반}) \cdot P(\text{반포한}|\text{반포}) \ldots \text{중략} \ldots \cdot P(\text{반포한강공원}|\text{반포한강공}) \cdot P(\text{반포한강공원에}|\text{반포한강공원}))}$

```
word_score_table["반포한"].cohesion_forward
```

```
0.08838002913645132
```

```
word_score_table["반포한강"].cohesion_forward
```

```
0.19841268168224552
```

```
word_score_table["반포한강공"].cohesion_forward
```

```
0.2972877884078849
```

```
word_score_table["반포한강공원"].cohesion_forward
```

```
0.37891487632839754
```

```
word_score_table["반포한강공원에"].cohesion_forward
```

```
0.33492963377557666
```

5. 한국어 전처리 패키지(Text Preprocessing Tools for Korean Text)

- 브랜칭 엔트로피(branching entropy): 주어진 문자열에서 다음 문자가 얼마나 등장할 수 있는지를 판단하는 척도
("다음 문자 예측을 위해 헷갈리는 정도")

```
word_score_table["디스"].right_branching_entropy
```

```
1.6371694761537934
```

```
word_score_table["디스플"].right_branching_entropy
```

```
-0.0
```

```
word_score_table["디스플레"].right_branching_entropy
```

```
-0.0
```

```
word_score_table["디스플레이"].right_branching_entropy
```

```
3.1400392861792916
```

'디스플레이' 뒤에 조사나 다른 단어가 다양하게 올 수 있음!

→ 전체 코퍼스에서 **하나의 단어가 끝나면 그 경계 부분부터 이 값이 증가**

5. 한국어 전처리 패키지(Text Preprocessing Tools for Korean Text)

1) L 토크나이저: 점수가 가장 높은 L 토큰을 찾아 분리

```
from soynlp.tokenizer import LTokenizer

scores = {word:score.cohesion_forward for word, score in word_score_table.items()}
l_tokenizer = LTokenizer(scores=scores)
l_tokenizer.tokenize("국제사회와 우리의 노력들로 범죄를 척결하자", flatten=False)
```

```
[('국제사회', '와'), ('우리', '의'), ('노력', '들로'), ('범죄', '를'), ('척결', '하자')]
```

2) 최대 점수 토크나이저: 점수가 가장 높은 글자 시퀀스를 순차적으로 탐색

- 띄어쓰기가 되어 있지 않는 문장 처리시 유용

```
from soynlp.tokenizer import MaxScoreTokenizer

maxscore_tokenizer = MaxScoreTokenizer(scores=scores)
maxscore_tokenizer.tokenize("국제사회와우리의노력들로범죄를척결하자")
```

```
['국제사회', '와', '우리', '의', '노력', '들로', '범죄', '를', '척결', '하자']
```

5. 한국어 전처리 패키지(Text Preprocessing Tools for Korean Text)

3) soynlp.normalizer: 반복되는 문자 정제

```
from soynlp.normalizer import *
```

```
print(emoticon_normalize('앜ㅋㅋㅋㅋ이영화존잼쓰ㅠㅠㅠㅠㅠ', num_repeats=2))
print(emoticon_normalize('앜ㅋㅋㅋㅋㅋㅋㅋㅋㅋ이영화존잼쓰ㅠㅠㅠㅠ', num_repeats=2))
print(emoticon_normalize('앜ㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋ이영화존잼쓰ㅠㅠㅠㅠㅠㅠ', num_repeats=2))
print(emoticon_normalize('앜ㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋ이영화존잼쓰ㅠㅠㅠㅠㅠㅠㅠㅠ', num_repeats=2))
```

```
아ㅋㅋ영화존잼쓰ㅠㅠ
아ㅋㅋ영화존잼쓰ㅠㅠ
아ㅋㅋ영화존잼쓰ㅠㅠ
아ㅋㅋ영화존잼쓰ㅠㅠ
```

```
print(repeat_normalize('와하하하하하하하하하핫', num_repeats=2))
print(repeat_normalize('와하하하하하하핫', num_repeats=2))
print(repeat_normalize('와하하하하핫', num_repeats=2))
```

```
와하하핫
와하하핫
와하하핫
```

5. 한국어 전처리 패키지(Text Preprocessing Tools for Korean Text)

◆ Customized KoNLPy

- 사용자 사전 추가가 매우 쉬운 패키지
- add_dictionary() 활용

```
pip install customized_konlpy
```

```
from ckonlpy.tag import Twitter
twitter = Twitter()
twitter.morphs('은경이는 사무실로 갔습니다.')
```

```
/usr/local/lib/python3.7/dist-packages/konlpy/tag/_okt.py:16: UserWarning: "Twitter" has changed to "Okt" since KoNLPy v0.4.5.
  warn('"Twitter" has changed to "Okt" since KoNLPy v0.4.5.')
['은', '경이', '는', '사무실', '로', '갔습니다', '.']
```

```
twitter.add_dictionary('은경이', 'Noun')
```

```
twitter.morphs('은경이는 사무실로 갔습니다.')
```

```
['은경이', '는', '사무실', '로', '갔습니다', '.']
```