딥 러닝을 이용한 자연어 처리 입문

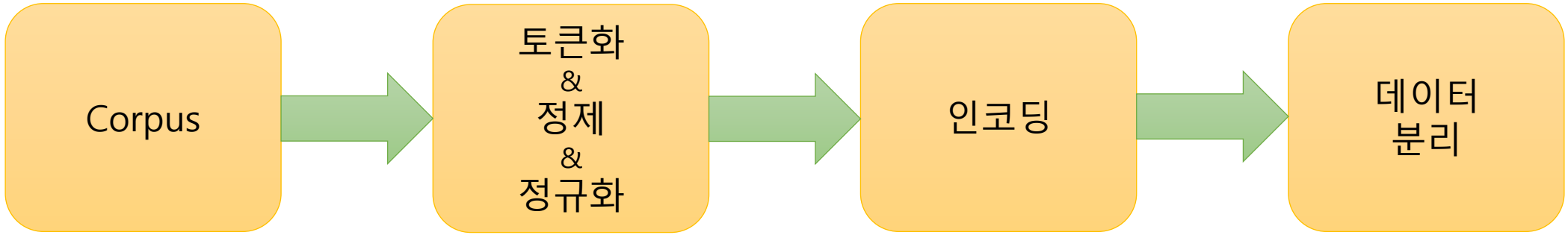
2장 텍스트 전처리

발표자 : 김성윤

목차

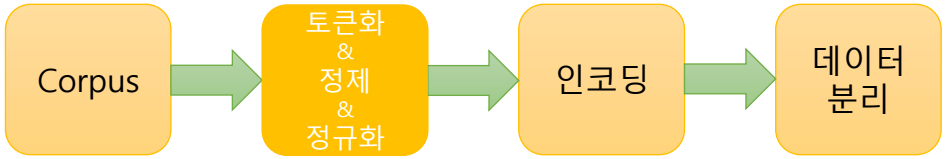
1. 텍스트 전처리

:풀고자 하는 문제의 용도에 맞게 텍스트를 사전에 처리하는 작업



*Corpus(코퍼스) : 가공하지 않은 데이터

2. 토큰화&정제&정규화



- 토큰화 종류

- 단어 토큰화 : 토큰의 기준을 단어(word)로 하는 경우

입력 : Time is an illusion. Lunchtime double so!

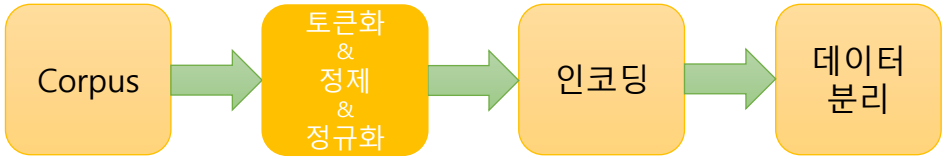
출력 : "Time", "is", "an", "illustion", "Lunchtime", "double", "so"

- 문장 토큰화 : 토큰의 단위가 문장(sentence)일 경우

```
import kss

text = '딥 러닝 자연어 처리가 재미있기는 합니다. 그런데 문제는 영어보다 한국어로 할 때 너무 어렵습니다. 이제 해보면 알걸요?'
print('한국어 문장 토큰화 :',kss.split_sentences(text))
```

```
한국어 문장 토큰화 : ['딥 러닝 자연어 처리가 재미있기는 합니다.', '그런데 문제는 영어보다 한국어로 할 때 너무 어렵습니다.',
'이제 해보면 알걸요?']
```

2. **토큰화**&정제&정규화



- 토큰화 고려사항

1. 구두점이나 특수 문자를 단순 제외해서는 안된다.

2. 줄임말과 단어 내에 띄어쓰기가 있는 경우

3. 품사

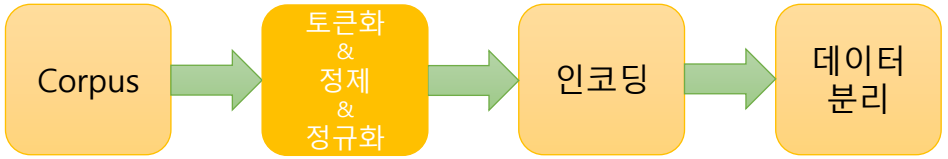
   예시 3.
   Fly : (명) 파리, (동) 날다

예시 1.
Ph.D, AT&T, $45.55, 01/02/06, 123,456,789

예시 2.

- Don't
- Don t
- Dont
- Do n't
- Jone's
- Jone s
- Jone
- Jones

2. 토큰화&정제&정규화

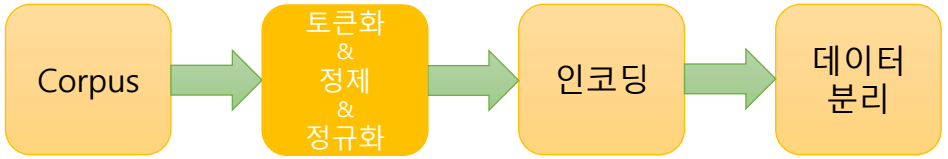


- 품사 태깅 - NLTK

```
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag

text = "I am actively looking for Ph.D. students. and you are a Ph.D. student."
tokenized_sentence = word_tokenize(text)

print('단어 토큰화 :',tokenized_sentence)
print('품사 태깅 :',pos_tag(tokenized_sentence))
```

```
단어 토큰화 : ['I', 'am', 'actively', 'looking', 'for', 'Ph.D.', 'students', '.', 'and', 'you', 'are', 'a', 'Ph.D.',
'student', '.']
품사 태깅 : [('I', 'PRP'), ('am', 'VBP'), ('actively', 'RB'), ('looking', 'VBG'), ('for', 'IN'), ('Ph.D.', 'NNP'), ('s
tudents', 'NNS'), ('.', '.'), ('and', 'CC'), ('you', 'PRP'), ('are', 'VBP'), ('a', 'DT'), ('Ph.D.', 'NNP'), ('studen
t', 'NN'), ('.', '.')]
```

2. 토큰화&정제&정규화



- 품사 태깅 - KoNLPy

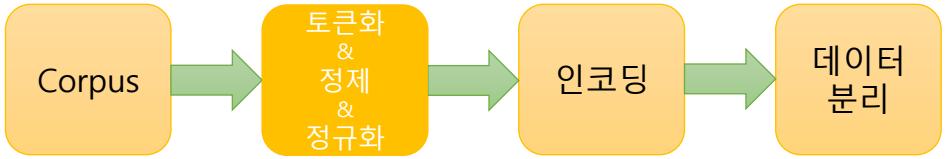
```
from konlpy.tag import Okt
from konlpy.tag import Kkma

okt = Okt()
kkma = Kkma()

print('OKT 형태소 분석 :',okt.morphs("열심히 코딩한 당신, 연휴에는 여행을 가봐요"))
print('OKT 품사 태깅 :',okt.pos("열심히 코딩한 당신, 연휴에는 여행을 가봐요"))
print('OKT 명사 추출 :',okt.nouns("열심히 코딩한 당신, 연휴에는 여행을 가봐요"))
```

```
OKT 형태소 분석 : ['열심히', '코딩', '한', '당신', ',', '연휴', '에는', '여행', '을', '가봐요']
OKT 품사 태깅 : [('열심히', 'Adverb'), ('코딩', 'Noun'), ('한', 'Josa'), ('당신', 'Noun'), (',', 'Punctuation'), ('연
휴', 'Noun'), ('에는', 'Josa'), ('여행', 'Noun'), ('을', 'Josa'), ('가봐요', 'Verb')]
OKT 명사 추출 : ['코딩', '당신', '연휴', '여행']
```

```
print('꼬꼬마 형태소 분석 :',kkma.morphs("열심히 코딩한 당신, 연휴에는 여행을 가봐요"))
print('꼬꼬마 품사 태깅 :',kkma.pos("열심히 코딩한 당신, 연휴에는 여행을 가봐요"))
print('꼬꼬마 명사 추출 :',kkma.nouns("열심히 코딩한 당신, 연휴에는 여행을 가봐요"))
```

```
꼬꼬마 형태소 분석 : ['열심히', '코딩', '하', 'ㄴ', '당신', ',', '연휴', '에', '는', '여행', '을', '가보', '아요']
꼬꼬마 품사 태깅 : [('열심히', 'MAG'), ('코딩', 'NNG'), ('하', 'XSV'), ('ㄴ', 'ETD'), ('당신', 'NP'), (',', 'SP'), ('연
휴', 'NNG'), ('에', 'JKM'), ('는', 'JX'), ('여행', 'NNG'), ('을', 'JKO'), ('가보', 'VV'), ('아요', 'EFN')]
꼬꼬마 명사 추출 : ['코딩', '당신', '연휴', '여행']
```

2. 토큰화&정제&정규화



영어	한국어
• 띄어쓰기 엄격	• 띄어쓰기 자율적 • *교착어

EX1) 제가이렇게띄어쓰기를전혀하지않고글을썼다고하더라도글을이해할수있습니다.

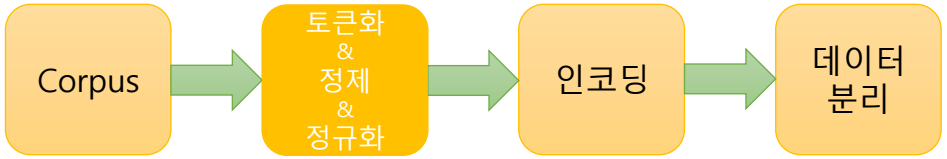
EX2) Tobeornottobethatisthequestion

문장 : 에디가 책을 읽었다

자립 형태소 : 에디, 책
의존 형태소 : -가, -을, 읽-, -었, -다

*교착어 : 어근에 접사가 결합하여 의미가 변화하는 형태의 언어

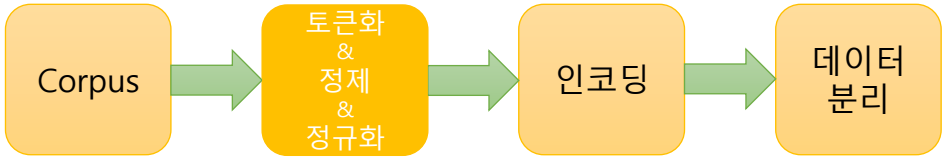
2. **토큰화**&정제&정규화



- 토큰화 유용한 도구 – 정규 표현식(Regular Expression)

특수 문자	설명
.	한 개의 임의의 문자를 나타냅니다. (줄바꿈 문자인 \n 는 제외)
?	앞의 문자가 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 또는 1개)
*	앞의 문자가 무한개로 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 이상)
+	앞의 문자가 최소 한 개 이상 존재합니다. (문자가 1개 이상)
^	뒤의 문자열로 문자열이 시작됩니다.
$	앞의 문자열로 문자열이 끝납니다.

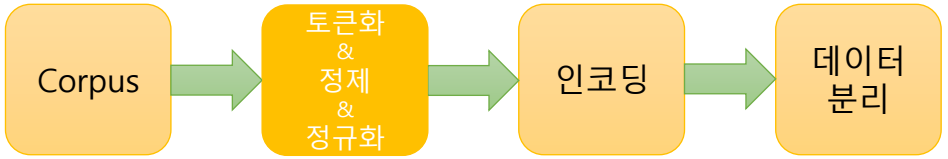
2. 토큰화&정제&정규화



- 토큰화 유용한 도구 – 정규 표현식(Regular Expression)

`{숫자}`	숫자만큼 반복합니다.
`{숫자1, 숫자2}`	숫자1 이상 숫자2 이하만큼 반복합니다. `?`, `*`, `+`를 이것으로 대체할 수 있습니다.
`{숫자,}`	숫자 이상만큼 반복합니다.
`[ ]`	대괄호 안의 문자들 중 한 개의 문자와 매치합니다. [amk]라고 한다면 a 또는 m 또는 k 중 하나라도 존재하면 매치를 의미합니다. `[a-z]`와 같이 범위를 지정할 수도 있습니다. `[a-zA-Z]`는 알파벳 전체를 의미하는 범위이며, 문자열에 알파벳이 존재하면 매치를 의미합니다.
`[^문자]`	해당 문자를 제외한 문자를 매치합니다.
`|`	`A|B`와 같이 쓰이며 A 또는 B의 의미를 가집니다.

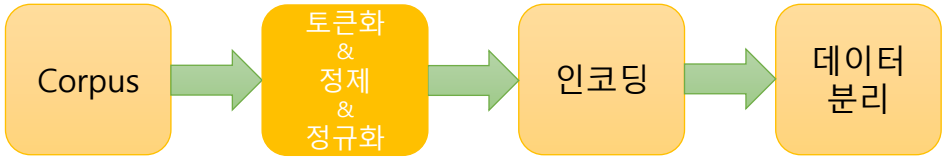
2. 토큰화&정제&정규화



- 토큰화 유용한 도구 – 정규 표현식(Regular Expression)

문자 규칙	설명
`\\\`	역 슬래쉬 문자 자체를 의미합니다
`\\d`	모든 숫자를 의미합니다. `[0-9]`와 의미가 동일합니다.
`\\D`	숫자를 제외한 모든 문자를 의미합니다. `[^0-9]`와 의미가 동일합니다.
`\\s`	공백을 의미합니다. `[ \t\n\r\f\v]`와 의미가 동일합니다.
`\\S`	공백을 제외한 문자를 의미합니다. `[^ \t\n\r\f\v]`와 의미가 동일합니다.
`\\w`	문자 또는 숫자를 의미합니다. `[a-zA-Z0-9]`와 의미가 동일합니다.
`\\W`	문자 또는 숫자가 아닌 문자를 의미합니다. `[^a-zA-Z0-9]`와 의미가 동일합니다.

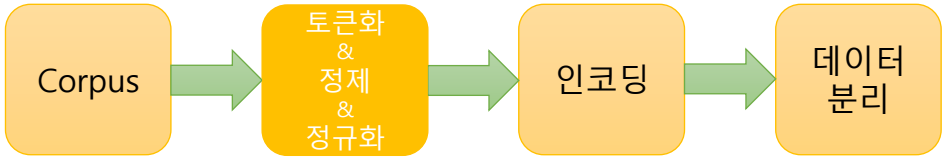
2. 토큰화&정제&정규화



- 토큰화 유용한 도구 – 정규 표현식(Regular Expression)

모듈 함수	설명
re.compile()	정규표현식을 컴파일하는 함수입니다. 다시 말해, 파이썬에게 전해주는 역할을 합니다. 찾고자 하는 패턴이 빈번한 경우에는 미리 컴파일해놓고 사용하면 속도와 편의성면에서 유리합니다.
re.search()	문자열 전체에 대해서 정규표현식과 매치되는지를 검색합니다.
re.match()	문자열의 처음이 정규표현식과 매치되는지를 검색합니다.
re.split()	정규 표현식을 기준으로 문자열을 분리하여 리스트로 리턴합니다.
re.findall()	문자열에서 정규 표현식과 매치되는 모든 경우의 문자열을 찾아서 리스트로 리턴합니다. 만약, 매치되는 문자열이 없다면 빈 리스트가 리턴됩니다.
re.finditer()	문자열에서 정규 표현식과 매치되는 모든 경우의 문자열에 대한 이터레이터 객체를 리턴합니다.
re.sub()	문자열에서 정규 표현식과 일치하는 부분에 대해서 다른 문자열로 대체합니다.

2. 토큰화&**정제**&정규화



- 정제

: 갖고 있는 코퍼스로부터 노이즈 데이터를 제거한다. (=*불용어를 제거한다.)

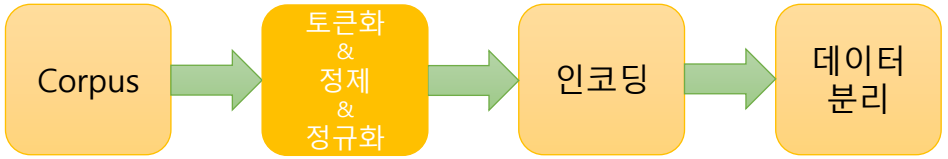
1. 등장 빈도가 적은 단어

2. 길이가 짧은 단어

: 영어권 언어에서 길이가 짧은 단어들은 대부분 불용어(영어 단어의 평균 길이는 6~7 정도)

*불용어 : 자주 등장하지만 실제 의미 분석을 하는데는 거의 기여하는 바가 없는 단어

2. 토큰화&정제&**정규화**



- 정규화

: 표현 방법이 다른 단어들을 통합시켜서 같은 단어로 만들어준다.

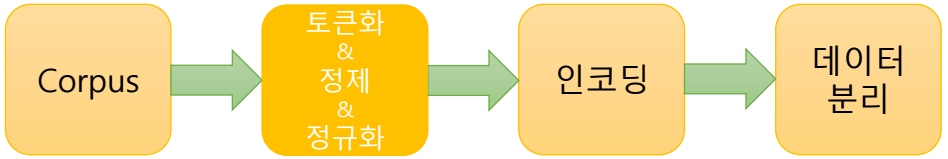
1. 규칙에 기반한 표기가 다른 단어들의 통합

: 어간 추출 & 표제어 추출

2. 대•소문자 통합

: 문장의 맨 앞에서 나오는 단어의 대문자만 소문자로 바꾸고, 다른 단어들은 전부 대문자인 상태로 놔두는 것 (US – us 구분)

2. 토큰화&정제&**정규화**

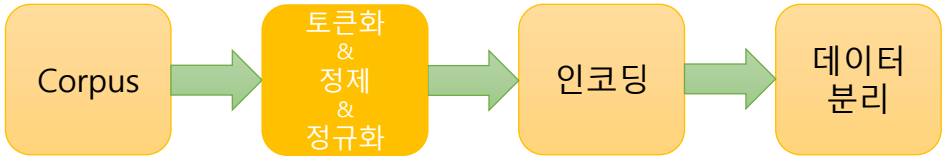


- 정규화 – 어간 추출
  - 형태학적 분석을 단순화한 버전
  - 정해진 규칙만 보고 단어의 어미를 자르는 어림짐작의 작업
  - 어간 추출 후에 나오는 결과 단어는 사전에 존재하지 않는 단어일 수도 있음 (organization -> organ)

```
from nltk.stem import PorterStemmer
from nltk.stem import LancasterStemmer

porter_stemmer = PorterStemmer()
lancaster_stemmer = LancasterStemmer()

words = ['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched', 'has', 'start
ing']
print('어간 추출 전 :', words)
print('포터 스테머의 어간 추출 후:',[porter_stemmer.stem(w) for w in words])
print('랭커스터 스테머의 어간 추출 후:',[lancaster_stemmer.stem(w) for w in words])
```

```
어간 추출 전 : ['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched', 'has',
'starting']
포터 스테머의 어간 추출 후: ['polici', 'do', 'organ', 'have', 'go', 'love', 'live', 'fli', 'die', 'watch', 'ha', 'star
t']
랭커스터 스테머의 어간 추출 후: ['policy', 'doing', 'org', 'hav', 'going', 'lov', 'liv', 'fly', 'die', 'watch', 'has',
'start']
```

2. 토큰화&정제&**정규화**



- 정규화 – 표제어 추출

- 표제어(Lemma) : 한글로는 '표제어' 또는 '기본 사전형 단어' 정도의 의미 (Am, are, is의 표제어 : be)
- 단어의 형태학적 파싱을 먼저 진행
- 형태소 : 1) 어간 2) 접사 ex. Cats = Cat + s
- 어간 추출과 달리 단어 형태가 적절히 보존

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

words = ['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched', 'has', 'start
ing']

print('표제어 추출 전 :',words)
print('표제어 추출 후 :',[lemmatizer.lemmatize(word) for word in words])
```

```
표제어 추출 전 : ['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched', 'ha
s', 'starting']
표제어 추출 후 : ['policy', 'doing', 'organization', 'have', 'going', 'love', 'life', 'fly', 'dy', 'watched', 'ha', 's
tarting']
```

```
lemmatizer.lemmatize('dies', 'v')
```

```
'die'
```

```
lemmatizer.lemmatize('watched', 'v')
```

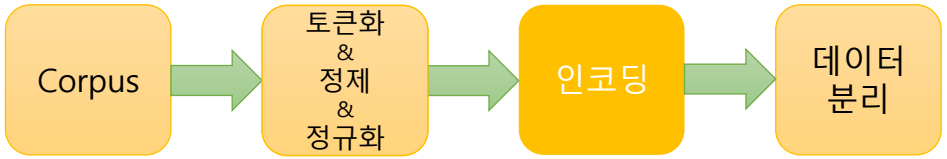
```
'watch'
```

```
lemmatizer.lemmatize('has', 'v')
```

```
'have'
```

품사 입력 시, 품사 정보 보존하면서
정확한 Lemma 값 출력

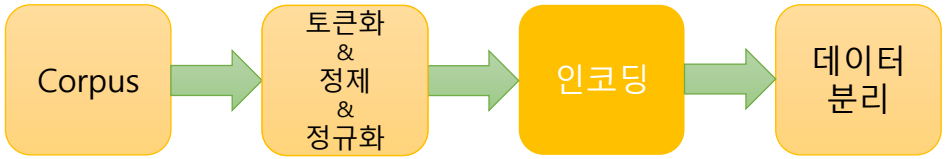
3. 인코딩



- 정수 인코딩(Integer Encoding) - 개념

- 각 단어를 고유한 정수에 맵핑(mapping)시키는 전처리 작업
- 단어를 빈도수 순으로 정렬한 단어 집합(vocabulary)을 만들고 빈도수가 높은 순서대로 차례로 낮은 숫자부터 정수를 부여하는 방법

3. 인코딩



- 정수 인코딩(Integer Encoding) - dictionary 사용하기

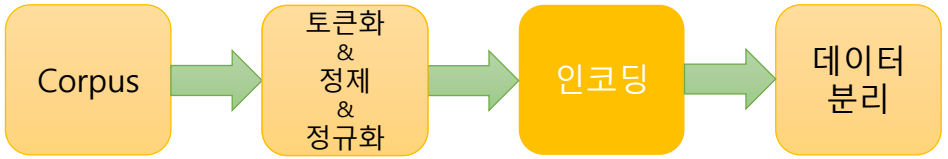
```
from nltk.tokenize import sent_tokenize
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
```

```
raw_text = "A barber is a person. a barber is good person. a barber is huge person. he Knew A Secret! The Secret He K
ept is huge secret. Huge secret. His barber kept his word. a barber kept his word. His barber kept his secret. But ke
eping and keeping such a huge secret to himself was driving the barber crazy. the barber went up a huge mountain."
```

```
# 문장 토큰화
sentences = sent_tokenize(raw_text)
print(sentences)
```

```
['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 kept his word.', 'His barber kept his s
ecret.', 'But keeping and keeping such a huge secret to himself was driving the barber crazy.', 'the barber went up a
huge mountain.']
```

3. 인코딩

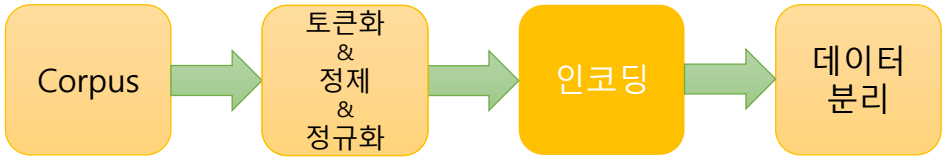


- 정수 인코딩(Integer Encoding) - dictionary 사용하기

```
vocab = {}
preprocessed_sentences = []
stop_words = set(stopwords.words('english'))

for sentence in sentences:
    # 단어 토큰화
    tokenized_sentence = word_tokenize(sentence)
    result = []
    #정제&정규화&빈도수 저장
    for word in tokenized_sentence:
        word = word.lower() # 모든 단어를 소문자화하여 단어의 개수를 줄인다.
        if word not in stop_words: # 단어 토큰화 된 결과에 대해서 불용어를 제거한다.
            if len(word) > 2: # 단어 길이가 2이하인 경우에 대하여 추가로 단어를 제거한다.
                result.append(word)
                if word not in vocab:
                    vocab[word] = 0
                vocab[word] += 1
    preprocessed_sentences.append(result)
print(preprocessed_sentences)
```

```
[['barber', 'person'], ['barber', 'good', 'person'], ['barber', 'huge', 'person'], ['knew', 'secret'], ['secret', 'ke
pt', 'huge', 'secret'], ['huge', 'secret'], ['barber', 'kept', 'word'], ['barber', 'kept', 'word'], ['barber', 'kep
t', 'secret'], ['keeping', 'keeping', 'huge', 'secret', 'driving', 'barber', 'crazy'], ['barber', 'went', 'huge', 'mo
untain']]
```

3. 인코딩



- 정수 인코딩(Integer Encoding) - dictionary 사용하기

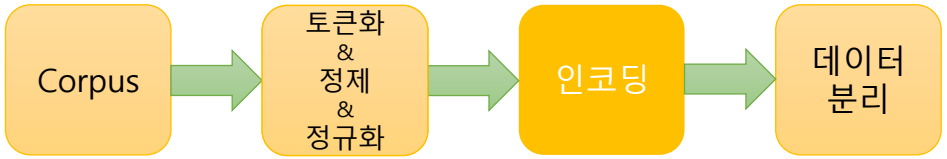
```
print('단어 집합 :',vocab)
```

```
단어 집합 : {'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}
```

```
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), ('k
new', 1), ('driving', 1), ('crazy', 1), ('went', 1), ('mountain', 1)]
```

3. 인코딩



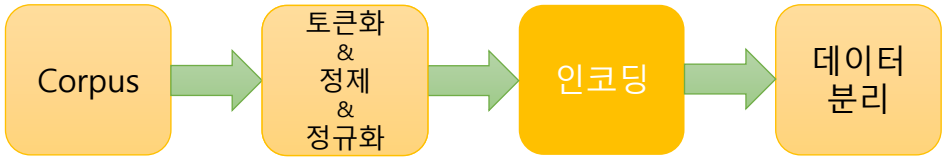
- 정수 인코딩(Integer Encoding) - dictionary 사용하기

```
vocab_size = 5

# 인덱스가 5 초과인 단어 제거
words_frequency = [word for word, index in word_to_index.items() if index >= vocab_size + 1]

# 해당 단어에 대한 인덱스 정보를 삭제
for w in words_frequency:
    del word_to_index[w]
print(word_to_index)
```

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

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

3. 인코딩



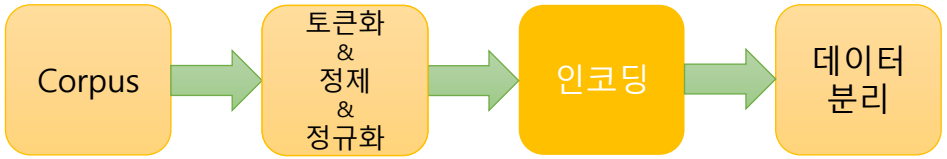
- 정수 인코딩(Integer Encoding) - dictionary 사용하기

```
['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 kept his word.', 'His barber kept his secret.', 'But keeping and keeping such a huge secret to himself was driving the barber crazy.', 'the barber went up a huge mountain.']
```

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

```
[[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]]
```

3. 인코딩



- 정수 인코딩(Integer Encoding) - Counter 사용하기

```
from collections import Counter
```

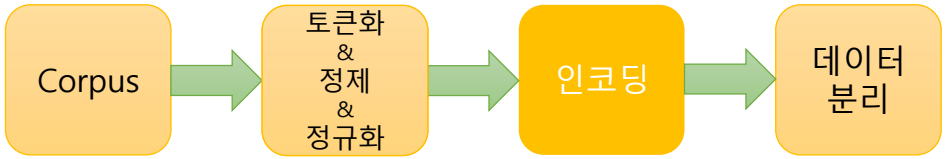
```
# words = np.hstack(preprocessed_sentences)으로도 수행 가능.
all_words_list = sum(preprocessed_sentences, [])
print(all_words_list)
```

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

```
# 파이썬의 Counter 모듈을 이용하여 단어의 빈도수 카운트
vocab = Counter(all_words_list)
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})
```

3. 인코딩



- 정수 인코딩(Integer Encoding) - NLTK의 FreqDist 사용하기

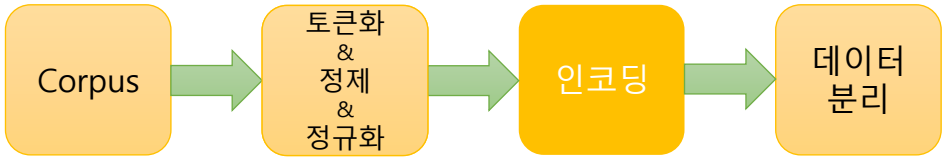
```
from nltk import FreqDist
import numpy as np
```

```
# np.hstack으로 문장 구분을 제거
vocab = FreqDist(np.hstack(preprocessed_sentences))
```

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

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

3. 인코딩

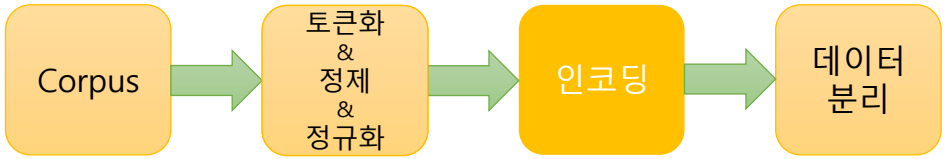


- 정수 인코딩(Integer Encoding) - enumerate 이해하기

```
test_input = ['a', 'b', 'c', 'd', 'e']
for index, value in enumerate(test_input): # 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
```

3. 인코딩



- 패딩(Padding)

- 병렬 연산을 위해서 여러 문장의 길이를 임의로 동일하게 맞춰주는 작업

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

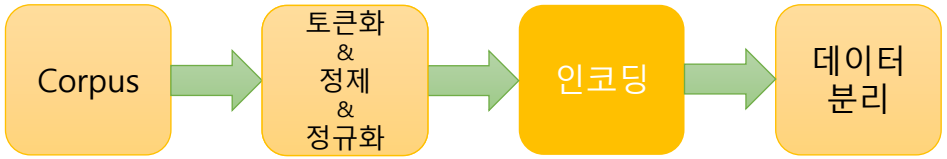
```
array([[ 1,  5,  0,  0,  0],
       [ 1,  8,  5,  0,  0],
       [ 1,  3,  5,  0,  0],
       [ 9,  2,  0,  0,  0],
       [ 2,  4,  3,  2,  0],
       [ 3,  2,  0,  0,  0],
       [ 1,  4,  6,  0,  0],
       [ 1,  4,  6,  0,  0],
       [ 1,  4,  2,  0,  0],
       [ 7,  7,  3,  2, 10],
      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

```
for sentence in encoded:
    while len(sentence) < max_len:
        sentence.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]])
```

제로 패딩

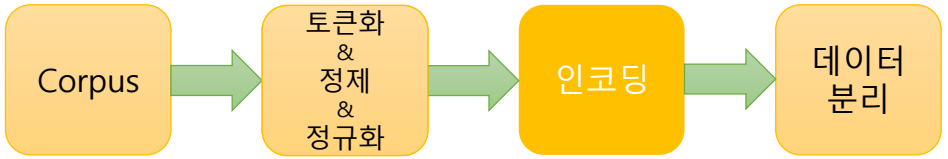
3. 인코딩



- 원-핫 인코딩(One-Hot Encoding) - 개념

- 컴퓨터 또는 기계가 더 잘 처리할 수 있도록 문자를 숫자로 바꾸는 작업
- 정수 인코딩 후, 단어 집합의 크기를 벡터의 차원으로 하고, 표현하고 싶은 단어의 인덱스에 1의 값을 부여하고, 다른 인덱스에는 0을 부여하는 단어의 벡터 표현 방식

3. 인코딩



- 원-핫 인코딩(One-Hot Encoding) - 과정

```
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"

tokenizer = Tokenizer()
tokenizer.fit_on_texts([text])
print('단어 집합 :',tokenizer.word_index)
```

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

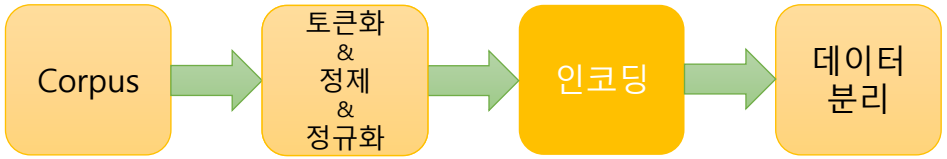
```
sub_text = "점심 먹으러 갈래 메뉴는 햄버거 최고야"
encoded = tokenizer.texts_to_sequences([sub_text])[0]
print(encoded)
```

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

```
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
```

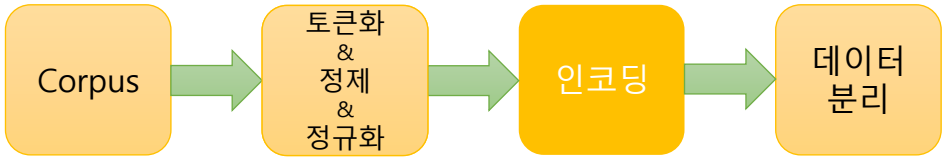
3. 인코딩



- 원-핫 인코딩(One-Hot Encoding) – 한계

- 단어의 개수가 늘어날 수록, 벡터를 저장하기 위해 필요한 공간이 계속 늘어남 (벡터 차원의 증가)
- 단어의 유사도를 표현하지 못함

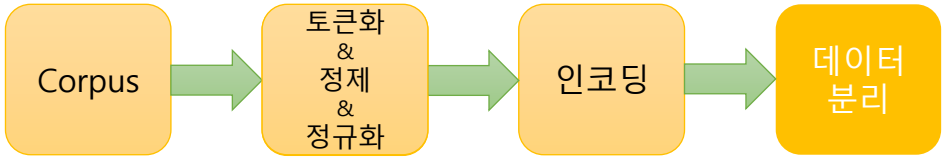
3. 인코딩



- 원-핫 인코딩(One-Hot Encoding) – 해결책

- 카운트 기반의 벡터화 방법인 LSA(잠재 의미 분석), HAL 등
- 예측 기반으로 벡터화하는 NNLM, RNNLM, Word2Vec, FastText 등
- 카운트 기반과 예측 기반 두 가지 방법을 모두 사용하는 GloVe 방법

4. 데이터 분리



- X데이터-y데이터 분리

```
X, y = zip(['a', 1], ['b', 2], ['c', 3])
print('X 데이터 :',X)
print('y 데이터 :',y)
```

```
X 데이터 : ('a', 'b', 'c')
y 데이터 : (1, 2, 3)
```

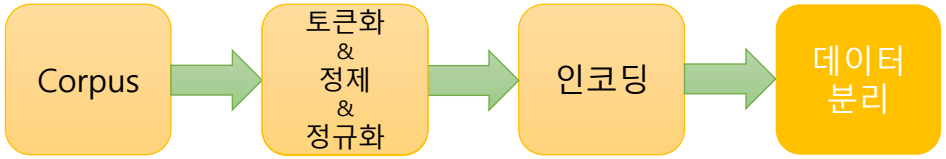
예시 1. Zip 함수 이용

```
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

예시 2. 데이터 프레임 이용

4. 데이터 분리



- X데이터-y데이터 분리

```
np_array = np.arange(0,16).reshape((4,4))
print('전체 데이터 :')
print(np_array)
```

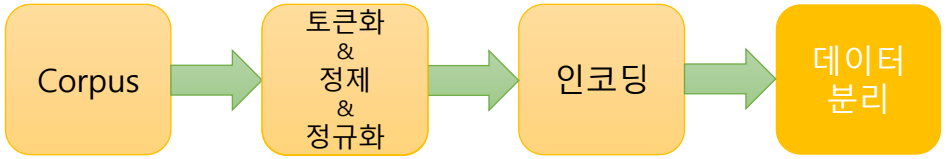
```
전체 데이터 :
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
```

예시 3. 넘파이 이용

```
X = np_array[:, :3]
y = np_array[:,3]

print('X 데이터 :')
print(X)
print('y 데이터 :',y)
```

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

4. 데이터 분리



- train데이터-test데이터 분리

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2, random_state=1234)
```

예시 1. 사이킷런 이용

```
num_of_train = int(len(X) * 0.8) # 데이터의 전체 길이의 80%에 해당하는 길이값을 구한다.
num_of_test = int(len(X) - num_of_train) # 전체 길이에서 80%에 해당하는 길이를 뺀다.
print('훈련 데이터의 크기 :',num_of_train)
print('테스트 데이터의 크기 :',num_of_test)
```

예시 2. 직접 분리

5. 한국어 전처리 패키지 소개

- **PyKoSpacing**

- 띄어쓰기가 되어있지 않은 문장을 띄어쓰기를 한 문장으로 변환해주는 패키지

```
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 소개

- **Py-Hanspell**

- 네이버 한글 맞춤법 검사기를 바탕으로 만들어진 패키지

```
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
```

5. 한국어 전처리 패키지 소개

- **SOYNLP**

- 비지도 학습으로 단어 토큰화
- 문자열이 자주 연결되어 등장한다면 한 단어라고 판단
- 응집 확률(cohesion probability)과 브랜칭 엔트로피(branching entropy)를 활용한 단어 점수표

```
from konlpy.tag import Okt
tokenizer = Okt()
print(tokenizer.morphs('에이비식스 이대휘 1월 최애돌 기부 요정'))
```

```
['에이', '비식스', '이대', '휘', '1월', '최애', '돌', '기부', '요정']
```

예시 1. 신조어 문제

5. 한국어 전처리 패키지 소개

- **SOYNLP**

- 비지도 학습으로 단어 토큰화
- 문자열이 자주 연결되어 등장한다면 한 단어라고 판단
- 응집 확률(cohesion probability)과 브랜칭 엔트로피(branching entropy)를 활용한 단어 점수표

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

응집 확률 수식

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

'반포한강공원에'라는 7의 길이를 가진 문자 시퀀스에 대해서
각 내부 문자열의 스코어를 구하는 과정
-> 스코어가 꺾이는 시점을 통해 한 단어를 판단

5. 한국어 전처리 패키지 소개

- **Customized KoNLPy**

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

```
['은', '경이', '는', '사무실', '로', '갔습니다', '.']
```

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

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

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