1. 데이터 확보

1-1 위키피디아

위키피디아 링크 (https://dumps.wikimedia.org/kowiki/)에서 xml 파일, gensim 패키지 다운로드 받는다

In [ ]:
```
from gensim.corpora import WikiCorpus,Dictionary
from gensim.utils import to_unicode
import re
```

In [ ]:
```
in_f = "notebooks/embedding/data/raw/kowiki-20200620-pages-articles-multistream1.xml-p1p76864.bz2"
out_f = "notebooks/embedding/data/processed/processed_wiki_ko.txt"
```

Wikicorpus는 다운받은 bz2파일에서 텍스트를 추출하고 텍스트 파일로 저장

In [6]:
```
WIKI_REMOVE_CHARS = re.compile("'+|(=+.{2,30}=+)|__TOC__|(ファイル:).+|:(en|de|it|fr|es|kr|zh|no|fi):|₩n", re.UNICODE)
WIKI_SPACE_CHARS = re.compile("(₩₩s㎖|  )+", re.UNICODE)
EMAIL_PATTERN = re.compile("(^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+₩.[a-zA-Z0-9-.]+$)", re.UNICODE)
#whdbfla6@gmail.com
URL_PATTERN = re.compile("(ftp|http|https)?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*₩(₩),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+", re.UNICODE)
WIKI_REMOVE_TOKEN_CHARS = re.compile("(₩₩*$|:$|^파일:.+|^;)", re.UNICODE)
MULTIPLE_SPACES = re.compile(' +', re.UNICODE)
```

In [7]:
```
EMAIL_PATTERN.match('whdbfla6@gmail.com')
```

Out[7]:
```
<re.Match object; span=(0, 18), match='whdbfla6@gmail.com'>
```

In [4]:
```
import re
p = re.compile('파이썬') #파이썬을 컴파일 해준다
p.match('파이썬')
```

Out[4]:
```
<re.Match object; span=(0, 3), match='파이썬'>
```

In [5]:
```
re.sub('₩d{4}','xxxx','010-4416-3880')
```

Out[5]:
```
'010-xxxx-xxxx'
```

In [ ]:
```
def tokenize(content, token_min_len=2, token_max_len=100, lower=True):
    content = re.sub(EMAIL_PATTERN, ' ', content)
    # email_pattern인 문자열들을 공백으로 바꿔주는 역할
    content = re.sub(URL_PATTERN, ' ', content)
    content = re.sub(WIKI_REMOVE_CHARS, ' ', content)
    content = re.sub(WIKI_SPACE_CHARS, ' ', content)
    content = re.sub(MULTIPLE_SPACES, ' ', content)
    tokens = content.replace(", )", "").split(" ")
    result = []
    for token in tokens:
        if not token.startswith('_'):
            token_candidate = to_unicode(re.sub(WIKI_REMOVE_TOKEN_CHARS, '', token))
        else:
            token_candidate = ""
        if len(token_candidate) > 0:
            result.append(token_candidate)
    return result
```

In [ ]:
```
def make_corpus(in_f, out_f):
    output = open(out_f, 'w', encoding = "utf-8")
    wiki = WikiCorpus(in_f, tokenizer_func=tokenize, dictionary=Dictionary())
    i = 0
    for text in wiki.get_texts():
        output.write(bytes(' '.join(text), 'utf-8').decode('utf-8') + '₩n')
        i = i + 1
    output.close()
```

1-2 JSON 파일

JSON 파일 읽는 법

"data": [
  { "paragraphs":
    [{ "qas": [
      { "answers" : [{ "text" : "교향곡", "answer_start" : 54 }],
       "id": "656904",
       "question" : "바그너는 괴테의 파우스트를 읽고 무엇을 쓰고자 했는가?"},

In [ ]:
```
def process_korQuAD(corpus_fname, output_fname):
    with open(corpus_fname) as f1, open(output_fname, 'w', encoding='utf-8') as f2:
        dataset_json = json.load(f1)
        dataset = dataset_json['data']
        for article in dataset:
            w_lines = []
            for paragraph in article['paragraphs']:
                w_lines.append(paragraph['context'])
                for qa in paragraph['qas']:
                    q_text = qa['question']
                    for a in qa['answers']:
                        a_text = a['text']
                        w_lines.append(q_text + " " + a_text)
            for line in w_lines:
                f2.writelines(line + "₩n")
```

전처리 완료된 데이터셋 일부

바그너는 괴테의 파우스트를 읽고 무엇을 쓰고자 했는가? 교양곡
바그너는 교향곡 작곡을 어디까지 쓴 뒤 중단했는가? 1악장

1-3 네이버 영화 리뷰 말뭉치

데이터 형태

id document label
9976970 아 더빙.. 진짜 짜증나네요 01

In [ ]:
```
with open(corpus_path, 'r', encoding='utf-8') as f1, ₩
            open(output_fname, 'w', encoding='utf-8') as f2:
    next(f1)  # skip head line
    for line in f1:
        _, sentence, label = line.strip().split('₩t')
        if not sentence: continue
        if with_label:
            f2.writelines(sentence + "₩u241E" + label + "₩n")
        else:
            f2.writelines(sentence + "₩n")
```

전처리 완료된 데이터 셋

document에만 있는 내용을 가져온다

2 지도학습 기반 형태소 분석

한국어는 조사와 어미가 발달한 교착어다.

- 교착어?
  어근과 접사가 결합해서 문장 내에서의 각 단어의 기능을 나타낸다. 어간에서는 어형 교체가 일어나지 않는다는 특징을 가짐.

*가겠다 가더라 가겠더라 가다*와 같이 활용형을 모두 어휘집합에 넣으면 어휘집합이 무한히 늘어날 수 있으며 매우 비효율적이다.

가겠다 > 가, 겠, 다
가더라 > 가, 더라

다음과 같이 형태소를 분석한 뒤에 형태소들을 단어로 취급하면 *가겠더라*라는 활용형이 말뭉치에 추가되더라도 어휘집합을 수정하지 않아도 처리가 가능하다

한국어는 한정된 종류의 조사와 어미를 자주 이용하기 때문에 각각에 대응하는 명사,용언,어간만 어휘집합에 추가하면 어휘집합을 줄일 수 있다

- 태깅?
  입력과 출력 쌍을 만드는 작업
  입력: 아버지가방에들어가신다
  출력: 아버지, 가, 방, 에, 들어가, 신다

2-1 KoNLPy

- konlpy 설치방법 링크 (https://konlpy-ko.readthedocs.io/ko/v0.4.4/install/#id1)에 상세히 설명되어 있음 참조!

In [ ]:
```
from konlpy.tag import Okt, Komoran, Mecab, Hannanum, Kkma

def get_tokenizer(tokenizer_name):
    if tokenizer_name == "komoran":
        tokenizer = Komoran()
    elif tokenizer_name == "okt":
        tokenizer = Okt()
    elif tokenizer_name == "mecab":
        tokenizer = Mecab()
    elif tokenizer_name == "hannanum":
        tokenizer = Hannanum()
    elif tokenizer_name == "kkma":
        tokenizer = Kkma()
    elif tokenizer_name == "khaiii":
        tokenizer = KhaiiiApi()
    else:
        tokenizer = Mecab()
    return tokenizer
```

In [ ]:
```
tokenizer = get_tokenizer("komoran")
tokenizer.morphs("아버지가방에들어가신다")
tokenizer.pos("아버지가방에들어가신다") #품사 태그 내용까지 확인 가능
```

- 은전한닢이 다른 분석기 대비 속도가 빠르다

2-2 Khaii

카카오가 공개한 오픈소스 한국어 형태소 분석기로, 국립국어원이 구축한 세종코퍼스를 이용해 CNN모델을 적용해 학습했다. 입력문장을 문자 단위로 일어 들인 후에 Convolution filter가 문자들의 정보를 추출한다. output layer에서 형태소의 경계와 품사태그를 예측한다

다운로드는 리눅스 환경에서만 지원하기 때문에 도커파일로 다운받자

In [ ]:
```
from khaiii import KhaiiApi
tokenizer = KhaiiApi()
```

In [ ]:
```
data = tokenizer.analyze("아버지가방에들어가신다")
tokens = []
for word in data:
    tokens.extend([str(m) for m in word.morphs])
```

'아버지/NNG' 형식으로 출력되기 때문에 품사 정보 외에 형태소 분석 결과만 확인하고 싶으면 `str(m).split("/")[0]`

3. 비지도 학습 기반 형태소 분석

비지도 학습 기법은 데이터의 패턴을 모델 스스로가 학습하게 해서 형태소를 분석하는 방법으로 데이터에 자주 등장하는 단어들을 형태소로 인식

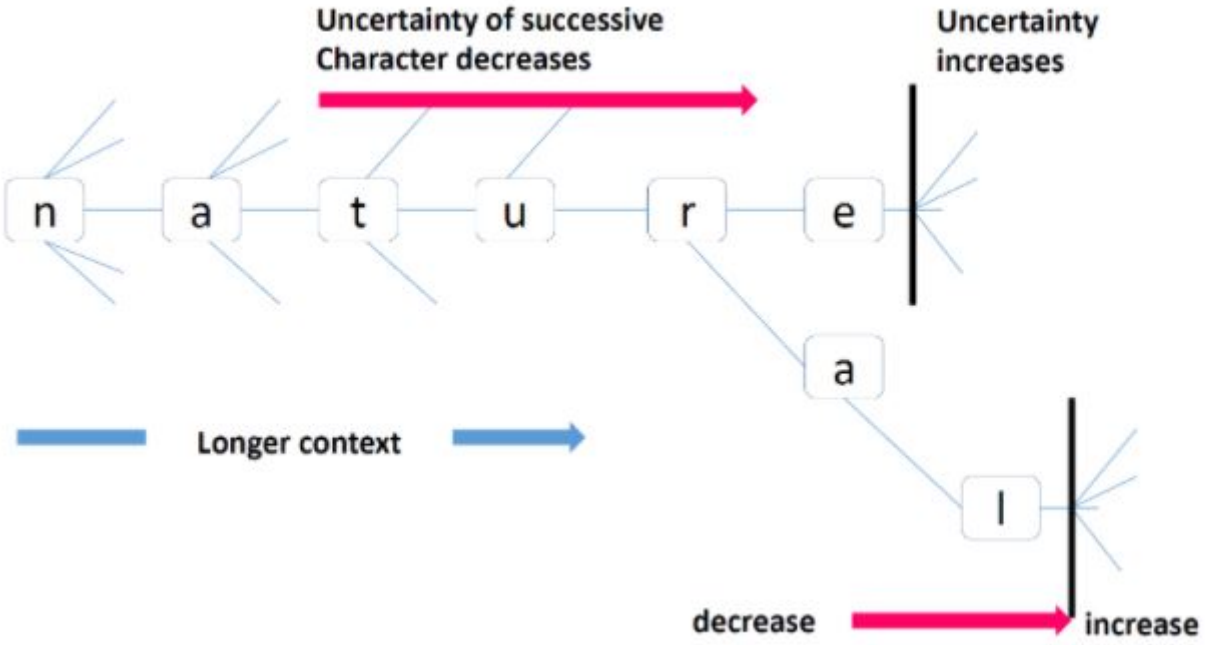
3-1 soynlp

soynlp는 형태소 분석, 품사 판별 등을 지원하는 파이썬 기반 한국어 자연어 처리 패키지다. 데이터 패턴을 스스로 학습해야하기 때문에 동질적인 문서집합에서 잘 작동함.

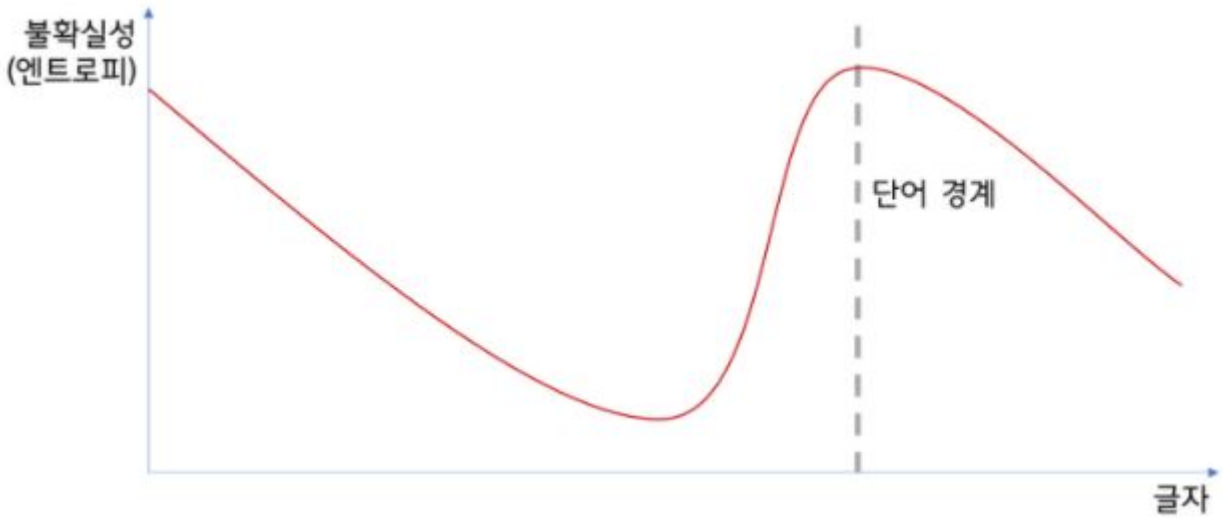
- 작동 원리: 데이터의 통계량을 확인해 응집확률과 브랜칭 엔트로피를 통해 단어 점수 생성, 이 점수표로 작동한다.

응집확률 : '영화 너무 꿀잼 ㅎㅎㅎ' 'ㄷ라마 정말 꿀잼 ㅋㅋㅋ' 등 말뭉치에서 *꿀잼*이라는 단어가 연결돼 자주 나타났다면 *꿀잼*의 응집 확률이 높아짐

브랜칭 엔트로피 : 단어 내부에서는 앤트로피가 줄어들고 경계에서는 증가하는 현상



알파벳 n 한글자만 주어질 때 이 정보만으로 어떤 다음 단어가 등장할지 알기가 어려움(앤트로피가 높은 상태) 하지만 단어가 natur까지 등장했다면 nature이 오거나 natural 두가지 경우만 존재(앤트로피가 낮은 상태) nature이 다 등장하고 나면 다음 글자 또한 예측하기 어려워 진다.



$$H(X|X_n) = -\sum_x p(x|x_n) log(p(x|x_n))$$

구분	빈도수
아직까지	4714
아직까진	632
아직까니	3
총합	**5349**

$$
\begin{aligned}
H(X|\text{아직까}) = &- P(\text{아직까지}|\text{아직까}) \times \log\left(P(\text{아직까지}|\text{아직까})\right) \\
&- P(\text{아직까진}|\text{아직까}) \times \log\left(P(\text{아직까진}|\text{아직까})\right) \\
&- P(\text{아직까니}|\text{아직까}) \times \log\left(P(\text{아직까니}|\text{아직까})\right)
\end{aligned}
$$

$$
= - \frac{4714}{5349} \times \log \frac{4714}{5349} - \frac{632}{5349} \times \log \frac{632}{5349} - \frac{3}{5349} \times \log \frac{3}{5349}
$$

$$
= 0.3679
$$

구분	아직	아직까	아직까지	아직까지도
BE	2.95	0.37	3.46	4.69

단어 내부에서 앤트로피(불확실성)이 가장 낮고 경계에서 증가하는 것을 확인할 수 있다

응집확률을 이용해서 형태소 추출을 하는데, 엔트로피가 높을 수록 형태소일 확률이 커지도록 알고리즘을 생성한다. 예를 들어 '엔진이' '엔진에서' '엔진을'과 같이 '*엔진*' 뒤에는 다양한 조사가 나올 수 있기 때문에 불확실성이 높다.

In [ ]:
```
from soynlp.word import WordExtractor
import math
from soynlp.tokenizer import LTokenizer

sentences = [sent.strip() for sent in open(corpus_fname, 'r').readlines()]
word_extractor = WordExtractor(min_frequency=100,
                               min_cohesion_forward=0.05,
                               min_right_branching_entropy=0.0
                               )

word_extractor.train(sentences)
word_extractor.save(model_fname)

word_extractor.load(model_fname)
scores = word_extractor.word_scores()

scores = {key:(scores[key].cohesion_forward * math.exp(scores[key].right_branching_entropy
)) for key in scores.keys()}
tokenizer = LTokenizer(scores=scores)
```

3-2 Google sentencepiece

sentencepiece는 구글에서 공개한 비지도 학습 기반 형태소 분석 패키지로 BPE(바이트 페어 인코딩 기법)을 지원한다. BPE는 말뭉치에서 가장 많이 등장한 문자열을 병합해 문자열을 압축하는 것이다

예를들어 aaabdaaabac 문자열에서 aa가 가장 많이 등장했기 때문에 Z로 치환하면 ZabdZabac로 나타낼 수 있다. ab를 Y로 치환하면 ZYdZYac로 나타낼 수 있다.

- BPE 학습 방식

전체 어휘 집합을 원하는 크기가 될 때까지 반복적으로 고빈도 문자열을 병합해 어휘집합에 추가

- BPE 예측 방식

각 어절에 어휘 집합에 있는 Subword가 포함되어 있는 경우에 해당 subword를 어절에서 분리한다. 이후 어절의 나머지에서 어휘집합에 있는 subword를 다시 찾고 분리한다. 이 과정을 어휘집합에 해당 단어가 없을 때까지 반복하고 나머지를 unknown word로 취급한다

BERT에서 BPE로 학습한 어휘집합을 쓴다고 함. BERT 모델 코드에는 BPE로 학습한 어휘집합으로 토큰을 분리할 수 있는 FullTokenizer 클래스가 포함돼 있다.

In [ ]:
```
from bert.tokenization import FullTokenizer

tokenizer = FullTokenizer(vocab_file=vocab_fname, do_lower_case=False)
tokenizer.tokenize("집에좀 가자")
```

['집에','##좀','가자']

3-3 띄어쓰기 교정

Soynlp에서 띄어쓰기 교정 모듈을 제공한다. 이 모듈은 corpus에서 띄어쓰기 패턴을 학습하고 그 패턴으로 교정을 수행한다.

하자고 앞 뒤로 공백이 다수 발견되면 예측 단계에서 *하자고* 앞 뒤를 띄어서 교정하는 방식

soynlp 형태소 분석이나 BPE 방식의 토크나이즈 기법은 띄어쓰기에 따라 결과가 크게 달라지기 때문에 교정을 먼저 적용해야 품질 개선이 된다

In [ ]:
```
from soyspacing.countbase import CountSpace

model = CountSpace()
model.train(corpus_fname)
model.save_model(model_fname,json_format=False)

model.load_model(model_fname,json_format=False)
model.correct("어릴때보고 지금다시봐도 재밌어요")
```

어릴때 보고 지금 다시봐도 재밌어요