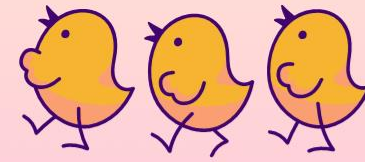


LangCon 2020



# KoNLPy 를 이용하여 Huggingface Transformers 학습하기

김현중

1. Hugging Face
2. Hugging Face: tokenizers (p8)
  1. 사전 기반 토큰나이저 vs subword tokenizer (p9)
  2. Hugging Face tokenizers (p19)
  3. tokenizers.BertWordPieceTokenizer (p25)
  4. tokenizers.SentencePieceBPETokenizer (p35)
  5. tokenizers.CharBPETokenizer (p38)
  6. tokenizers.ByteLevelBPETokenizer (p40)
3. KoNLPy 를 pre-tokenizer 로 이용하기 (p48)
4. KoNLPy 를 WordPiece 로 이용하기 (p61)
5. KoNLPy + BertTokenizer 로 BERT 학습하기 (p72)

# KoNLPy 를 이용하여 Huggingface Transformers 학습하기

김현중

soy.lovit@gmail.com

# Hugging Face

---

- NLP 기술을 많은 이들이 편히 이용할 수 있도록 기술민주화를 하는 그룹



## HUGGING FACE

On a mission to solve NLP,  
one commit at a time.

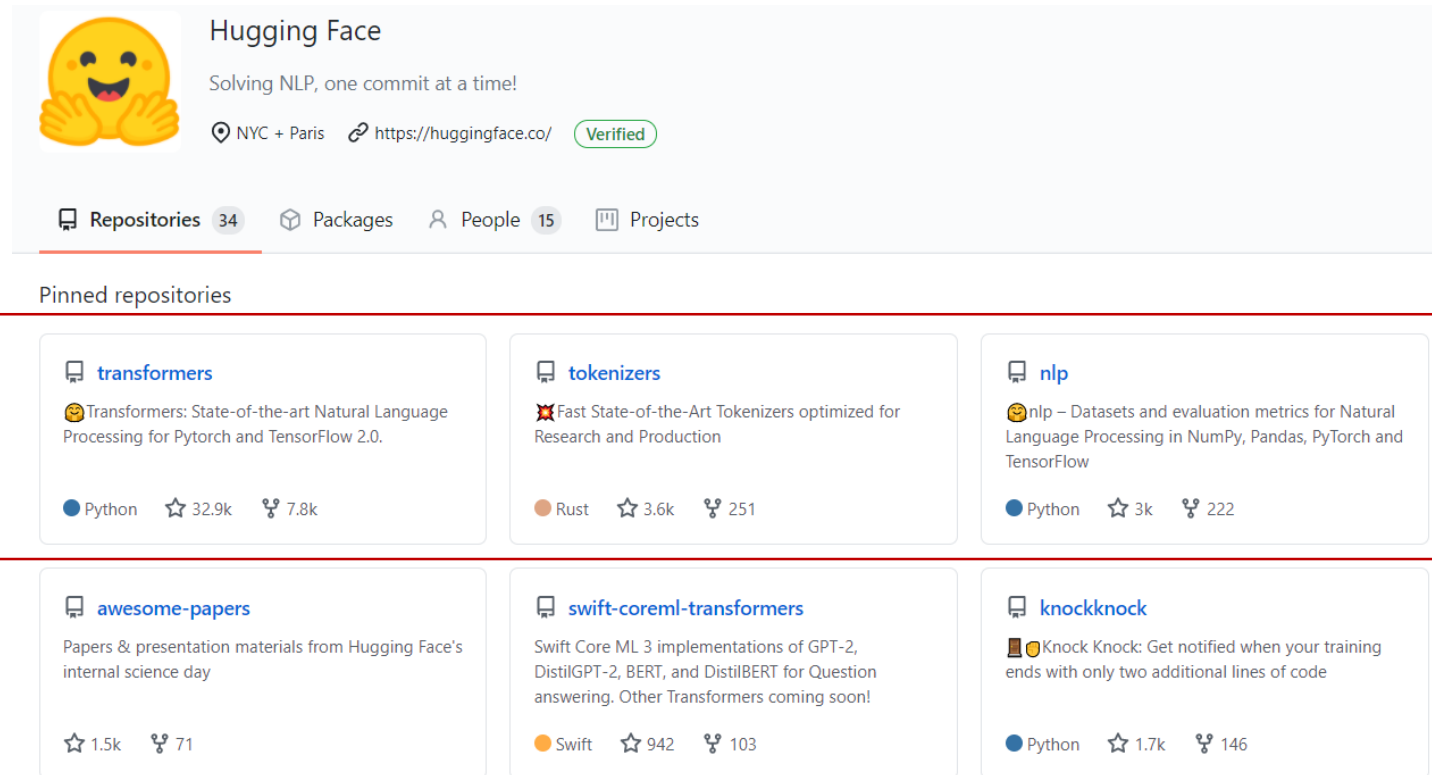
 TECH & SCIENCE

### Our science contributions

We're on a journey to advance and democratize NLP for everyone. Along the way, we contribute to the development of technology for the better.

# Hugging Face

- NLP 와 관련된 다양한 패키지를 제공하고 있습니다.



The screenshot displays the GitHub profile for Hugging Face. At the top, the profile name 'Hugging Face' is shown with a yellow hugging face emoji as the avatar. Below the name, the bio reads 'Solving NLP, one commit at a time!'. The location is listed as 'NYC + Paris' and the website is 'https://huggingface.co/'. A 'Verified' badge is present. Below the profile information, there are statistics for 'Repositories' (34), 'Packages', 'People' (15), and 'Projects'. The 'Pinned repositories' section is highlighted with a red border and contains six items:

- transformers**: Transformers: State-of-the-art Natural Language Processing for Pytorch and TensorFlow 2.0. Python, 32.9k stars, 7.8k forks.
- tokenizers**: Fast State-of-the-Art Tokenizers optimized for Research and Production. Rust, 3.6k stars, 251 forks.
- nlp**: nlp – Datasets and evaluation metrics for Natural Language Processing in NumPy, Pandas, PyTorch and TensorFlow. Python, 3k stars, 222 forks.
- awesome-papers**: Papers & presentation materials from Hugging Face's internal science day. 1.5k stars, 71 forks.
- swift-coreml-transformers**: Swift Core ML 3 implementations of GPT-2, DistilGPT-2, BERT, and DistilBERT for Question answering. Other Transformers coming soon! Swift, 942 stars, 103 forks.
- knockknock**: Knock Knock: Get notified when your training ends with only two additional lines of code. Python, 1.7k stars, 146 forks.

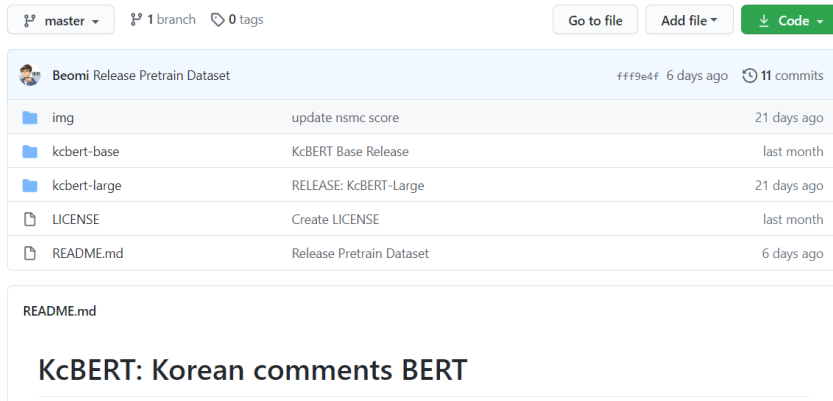
# Hugging Face

Package	Goal
<a href="#">transformers</a>	<ul style="list-style-type: none"><li>Transformer 기반 (masked) language model 알고리즘 제공</li><li>기학습된 (pretrained) 모델 배포</li></ul>
<a href="#">tokenizers</a>	<ul style="list-style-type: none"><li>transformers package 에서 사용할 수 있는 subword 토큰나이저 학습</li><li>transformers 와 분리되어 다른 목적에도 이용할 수 있습니다.</li></ul>
<a href="#">nlp</a>	<ul style="list-style-type: none"><li>데이터셋 및 NLP tasks 평가 척도 (evaluation metrics) 제공</li></ul>

- dataset → tokenizer → models 의 언어 모델 학습에 필요한 전 과정을 지원합니다.

# KcBERT: Korean comments BERT

- 이준범 (<https://github.com/Beomi>)님께서 최근에 댓글을 이용하여 학습한 BERT 모델과 학습에 이용한 데이터를 Kaggle 에 공개하였습니다.
- 데이터의 추가와 토큰라이저의 설정으로 다양한 언어 모델을 학습할 수 있습니다.



The screenshot shows a GitHub repository page for 'Beomi Release Pretrain Dataset'. The repository is on the 'master' branch and has 1 branch and 0 tags. It contains 11 commits. The file list includes:

File	Description	Last Commit
img	update nsmc score	21 days ago
kcbert-base	KcBERT Base Release	last month
kcbert-large	RELEASE: KcBERT-Large	21 days ago
LICENSE	Create LICENSE	last month
README.md	Release Pretrain Dataset	6 days ago

The README.md file content is visible below the file list:

```
README.md

KcBERT: Korean comments BERT
```



The screenshot shows a Kaggle dataset page for 'KcBERT Pre-Training Corpus (Korean News Comments)'. The dataset is a 'Korean Naver News Comments Corpus (cleaned)' and was updated 6 days ago (Version 1). The page includes navigation tabs for Data, Tasks, Notebooks (1), Discussion, Activity, and Metadata. There are buttons for 'Download (12 GB)' and 'New Notebook'.

# Hugging Face: tokenizers



# 사전 기반 토큰라이저 vs subword tokenizer

토큰라이저	토큰 단위	vocab size	예시	미등록단어 가정
사전 기반	알려진 단어/형태소 및 이들의 결합	unlimited	KoNLPy	<ul style="list-style-type: none"><li>알려진 단어/형태소의 결합이라 가정</li><li>필요시 형태소 분석 (형태 변형 가능)</li></ul>
subword	알려진 글자 및 subword	fixed	Hugging Face tokenizers	<ul style="list-style-type: none"><li>알려진 subwords 로 분해</li><li>e.g) appaer → app + ear</li></ul>

```
sent = '코로나 사태가 심각했습니다'

print(bert_tokenizer.tokenize(sent))
print(komoran.pos(sent, join=True))
print(komoran.morphs(sent))
```

['코로나', '사태', '##가', '심각', '##했', '##습니다']

['코로나/NNP', '사태/NNG', '가/JKS', '심각/XR', '하/XSA', '었/EP', '습니다/EC']

['코로나', '사태', '가', '심각', '하', '었', '습니다']

# 사전 기반 토크나이저

- 알려진 단어/형태소의 조합으로 문장을 토큰화 합니다
  - 후보들 중에서 확률 (점수)가 가장 큰 후보를 최종 토큰으로 선택합니다.

sent = "코로나가 심하다"

후보1 = [(코/명사), (로/조사), (나가/동사), ...]

후보2 = [(코로나/명사), (가/조사), (심하/형용사), ...]

$$P(\text{후보 1} \mid \text{sent}) <? P(\text{후보 2} \mid \text{sent})$$

(코, 명사),  
(코로나, 명사),  
(로, 조사)  
(나가, 동사)  
(나, 조사)  
...

단어/형태소 사전

# 사전 기반 토큰나이저

- 알려지지 않은 단어의 품사를 추정할 수도 있습니다  
(확률이 크면 "후보2"를 선택합니다)

sent = "코로나가 심하다"

후보1 = [(코/명사), (로/조사), (나가/동사), ...]

후보2 = [(Unk/명사), (가/조사), (심하/형용사), ...]

$$P(\text{후보 1} \mid \text{sent}) <? P(\text{후보 2} \mid \text{sent})$$

(코, 명사),  
~~(코로나, 명사),~~  
(로, 조사)  
(나가, 동사)  
(나, 조사)  
...

단어/형태소 사전

# 사전 기반 토큰나이저

- 여러 어절의 문맥 정보를 활용하는 경우가 많습니다.
  - 그러나 어절만으로도 문맥이 명확하다면 반드시 여러 어절을 이용해야 하는 것도 아닙니다.

sent = "코로나가 심하다"

$P(w_i = \text{"Unk/명사 + 가/조사"} \mid w_{i+1} = \text{심하/형용사 + 다/어미})$

>

$P(w_i = \text{"코/명사 + 로/조사 + 나/명사 + 가/조사"} \mid w_{i+1} = \text{심하/형용사 + 다/어미})$

(코, 명사),  
(~~코로나~~, 명사),  
(로, 조사)  
(나가, 동사)  
(나, 조사)  
...

단어/형태소 사전

# 사전 기반 토큰나이저

- 확률이 크지 않으면 알려진 단어/형태소의 조합으로 잘못 분해될 가능성이 있습니다.

sent = "코로나가 심하다"

후보1 = [(코/명사), (로/조사), (나/명사), (가/조사), ...]

후보2 = [(Unk/명사), (가/조사), (심하/형용사), ...]

$$P(\text{후보 1} \mid \text{sent}) > P(\text{후보 2} \mid \text{sent})$$

(코, 명사),  
~~(코로나, 명사),~~  
(로, 조사)  
(나가, 동사)  
(나, 조사)  
...

단어/형태소 사전

# Subword tokenizer

---

- 알려진 글자/서브워드의 조합으로 문장을 토큰화 합니다
  - 데이터를 잘 설명하는 (적은 수의 서브워드로 많은 다수의 문장을 조합하는) 서브워드를 직접 학습합니다.
  - (대표적인 학습법은 뒤에서 간단히 살펴봅니다)

sent = "코로나가 심하다"

tokens = [코로나, 가</w>, 심하, 다</w>]

코, 로, 나, 가, 심, 하, 다,  
코로, 코로나, 로나, ...,  
가</w>, 다</w>,  
...

서브워드 사전

# Subword tokenizer

---

- 이 방법의 목적은
  - 1) 고정된 개수의 토큰으로 문장을 표현하고
  - 2) 언어적 지식을 이용하지 않음으로써 다수의 언어에 공통으로 적용하기 위함 입니다.

sent = "코로나가 심하다"

tokens = [코로나, 가</w>, 심하, 다</w>]

코, 로, 나, 가, 심, 하, 다,  
코로, 코로나, 로나, ...,  
가</w>, 다</w>,  
...

서브워드 사전

# 사전 기반 토크나이저 vs subword tokenizer

- 사전 기반 토크나이저는 미등록단어를 맥락에 따라 잘못 분해할 가능성이 존재합니다.
- Subword tokenizer 는 학습데이터에 자주 등장하는 substring 이라면 단어로 보존될 가능성이 높습니다.

어절	Mecab	Komoran	Bert + Covid news
코로나가	코 로 나가	코로나 가	코로나 ##가
코로나는	코로나 는	코로나 는	코로나 ##는
코로나를	코로나 를	코로나 를	코로나 ##를
코로나의	코로 나의	코로나 의	코로나 ##의



# 사전 기반 토큰라이저 vs subword tokenizer

- 반대로 학습데이터에 자주 등장하지 않은 단어 **심각** 은 잘 인식되지 않습니다.
- 활용된 용언 **하다**가 다양한 형태로 표현됩니다 (**합니다**, **했다**, **하다 + 고**, **하며**)
  - 반드시 원형으로 처리해야 하는 것은 아니나, 이를 원할 경우 방법이 없습니다.

어절	Mecab	Komoran	Bert + Covid news
심각합니다	심각 합니다	심각 하 ㅂ니다	심 ##각 ##합니다
심각했다	심각 했 다	심각 하 었 다	심 ##각 ##했다
심각하다고	심각 하 다고	심각 하 다고	심 ##각 ##하다 ##고
심각하며	심각 하 며	심각 하 며	심 ##각 ##하며

# 사전 기반 토크나이저 vs subword tokenizer

---

- 두 종류의 토크나이저는 각자의 장단점도, 사용 목적도 서로 다릅니다.
  - Subword tokenizers 는 자주 등장하는 단어를 제대로 인식할 가능성이 높지만,
  - 빈번하지 않은 단어는 사전 기반 토크나이저가 잘 인식합니다.
  - 전체 단어의 개수가 제한된 상황에서는 사전 기반 방식으로 인식된 다수의 단어를 unk 로 처리해야 하지만, subword tokenizers 는 알려진 글자로 분해하여 unk 의 개수를 줄입니다.
- 목적에 맞게 선택하여 사용하면 됩니다.

# Hugging Face tokenizers

- Subwords 사전을 학습하는 방법과 단어를 subwords 로 조합하는 방법에 따라 몇 가지 subword tokenizers 가 제안되었습니다.
- Hugging Face tokenizers 에서 대부분을 제공하고 있습니다.

tokenizer (class name)	unit	trainer	normalizer	boundary symbol	outputs
Byte-level BPE (ByteLevelBPETokenizer)	byte	BPE	[Unicode, Lowercase]	어절 앞 Ġ	vocab.json / merges.txt
Character-level BPE (CharBPETokenizer)	char	BPE	[Unicode, BertNormalizer, Lowercase]	어절 뒤 </w>	vocab.json / merges.txt
SentencePiece (SentencePieceBPETokenizer)	char	BPE	NFKC	어절 앞 _	vocab.json / merges.txt
Bert tokenizer (BertWordPieceTokenizer)	char	WordPiece	BertNormalizer	어절 중간 서브워드 앞에 ## 부착	vocab.txt

# BPE vs WordPiece

---

- tokenizers 에서는 BPE 와 WordPiece trainer 를 제공합니다.

```
from tokenizers import trainers

class SentencePieceBPETokenizer(BaseTokenizer):
    # ...
    def train(...):
        trainer = trainers.BpeTrainer(
            vocab_size=vocab_size,
            ...
        )

class BertWordPieceTokenizer(BaseTokenizer):
    # ...
    def train(...):
        trainer = trainers.WordPieceTrainer(
            vocab_size=vocab_size, ...
        )
```

# Byte-Pair Encoding (BPE)

```
vocab = {'l o w </w>' : 5,  
        'l o w e r </w>' : 2,  
        'n e w e s t </w>':6,  
        'w i d e s t </w>': 3}  
num_merges = 10  
  
# training units  
for i in range(num_merges):  
    pairs = get_stats(vocab)  
    best = max(pairs, key=pairs.get)  
    vocab = merge_vocab(best, vocab)
```

```
# subword tokens  
vocab = {  
    'low</w>': 5,  
    'low e r </w>': 2,  
    'newest</w>': 6,  
    'wi d est</w>': 3  
}
```

```
# merging  
( 'e', 's' )  
( 'es', 't' )  
( 'est', '</w>' )  
( 'l', 'o' )  
( 'lo', 'w' )  
( 'n', 'e' )  
( 'ne', 'w' )  
( 'new', 'est</w>' )  
( 'low', '</w>' )  
( 'w', 'i' )
```

```
# final units
```

```
{'low</w>': 5,  
 'low': 2,  
 'e': 2,  
 'r': 2,  
 '</w>': 2,  
 'newest</w>': 6,  
 'wi': 3,  
 'd': 3,  
 'est</w>': 3}
```

# Hugging Face tokenizers

---

- `transformers` 에서는 토크나이저의 학습 기능을 제공하지 않습니다.
  - 모든 토크나이저는 pretrained tokenizer 라 가정하기 때문에
  - `tokenizers` 를 이용하여 토크나이저를 학습해야 합니다.
- `tokenizers` 에서 제공하는 토크나이저들의 기능을 먼저 이해해 봅니다.
  - `0.9.0dev` 가 나왔으나 dev 버전이기에 `0.8.1` 로 노트를 만들었습니다.

# tokenizers.BaseTokenizer

---

- 네 가지 토크나이저 모두 BaseTokenizer 를 상속하며, 아래의 기능이 구현되어 있습니다.
  - get\_vocab()
  - add\_tokens(), add\_special\_tokens()
  - ~~normalize()~~ # 0.9.0dev 에서 제거되었습니다.
  - encode() / encode\_batch()
  - decode() / decode\_batch()
  - save() vs save\_model()
- tokenize() 의 기능이 없고 대신 encode() 를 이용해야 합니다.

# tokenizers

---

- 6글자 / 2줄로 이뤄진 매우 작은 텍스트를 이용하여 각 토큰라이저의 기능을 살펴봅니다.

```
( very_small_corpus.txt )
```

```
ABCDE ABC AC ABD  
DE AB ABC AF
```



# tokenizers.BertWordPieceTokenizer

- vocab\_size 와 min\_frequency만 변경하였습니다

```
from tokenizers import BertWordPieceTokenizer

bert_wordpiece_tokenizer = BertWordPieceTokenizer()
bert_wordpiece_tokenizer.train(
    files = [small_corpus],
    vocab_size = 10,
    min_frequency = 1,
    limit_alphabet = 1000,
    initial_alphabet = [],
    special_tokens = ["[PAD]", "[UNK]", "[CLS]", "[SEP]", "[MASK]"],
    show_progress = True,
    wordpieces_prefix = "##",
)
vocab = bert_wordpiece_tokenizer.get_vocab()
sorted(vocab, key=lambda x: vocab[x])
```

```
['[PAD]', '[UNK]', '[CLS]', '[SEP]', '[MASK]', 'a', 'b', 'c', 'd', 'e', 'f', '##e', '##b',
'##c', '##d', '##f']
```

# tokenizers.BertWordPieceTokenizer

---

- `tokenize()` 대신 `encode()` 를 이용하며, `tokens` 와 `ids` 를 가져올 수 있습니다.
- `ids` 는 `vocab` 의 순서대로의 인덱스입니다.

```
encoding = bert_wordpiece_tokenizer.encode('ABCDE')  
print(encoding.tokens)  
print(encoding.ids)
```

```
['a', '##b', '##c', '##d', '##e']  
[5, 11, 12, 13, 14]
```

# tokenizers.BertWordPieceTokenizer

---

- `lowercase=False` 를 하면 대문자를 유지합니다.
- `train(files=)` 에는 하나의 파일을 넣을 수도, 여러 개의 파일을 넣을 수도 있습니다.

```
from tokenizers import BertWordPieceTokenizer

bert_wordpiece_tokenizer = BertWordPieceTokenizer(
    lowercase = False)
bert_wordpiece_tokenizer.train(files=[small_corpus], vocab_size=10)
encoding = bert_wordpiece_tokenizer.encode('ABCDE')
print(encoding.tokens)
print(encoding.ids)
```

```
['A', '##B', '##C', '##D', '##E']
[5, 11, 14, 12, 15]
```

# tokenizers.BertWordPieceTokenizer

- vocab\_size 를 늘리면 더 많은 subwords 가 vocab 으로 학습됩니다.

```
bert_wordpiece_tokenizer.train(  
    files = [small_corpus],  
    vocab_size = 20,  
    min_frequency = 1,  
    initial_alphabet = ['g'],  
)  
vocab = bert_wordpiece_tokenizer.get_vocab()  
print(sorted(vocab, key=lambda x: vocab[x]))
```

```
['[PAD]', '[UNK]', '[CLS]', '[SEP]', '[MASK]', 'a', 'b', 'c', 'd', 'e', 'f', 'g', '##b', '##  
d', '##c', '##e', '##f', 'ab', 'abc', 'ac']
```

```
encoding = bert_wordpiece_tokenizer.encode('ABCDE')  
print(encoding.tokens)  
print(encoding.ids)
```

```
['abc', '##d', '##e']  
[18, 15, 12]
```

# tokenizers.BertWordPieceTokenizer

---

- `encode_batch` 는 `list of str` 을 입력받아 `list of Encoding` 을 return 합니다.

```
encodings = bert_wordpiece_tokenizer.encode_batch(['ABCDE', 'abcd'])  
print(encodings[0].tokens)  
print(encodings[1].tokens)
```

```
['abc', '##d', '##e']  
['abc', '##d']
```

# tokenizers.BertWordPieceTokenizer

---

- `save_model` 을 이용하여 `{directory}/{name}-vocab.txt` 파일로 vocab 을 저장합니다.
- BertWordPieceTokenizer 는 `vocab.txt` 파일만 저장합니다.

```
bert_wordpiece_tokenizer.save_model(  
    directory = './',  
    name = 'very_small_bertwordpiece'  
)
```

```
 ['./very_small_bertwordpiece-vocab.txt']
```

# tokenizers.BertWordPieceTokenizer

---

- vocab\_file 을 입력하면 학습한 vocab 을 이용할 수 있습니다.
- 문장 앞, 뒤로 [CLS], [SEP] 토큰이 부착되며, 이는 선택할 수 있습니다.

```
bert_wordpiece_tokenizer = BertWordPieceTokenizer(  
    vocab_file = './very_small_bertwordpiece-vocab.txt'  
)  
bert_wordpiece_tokenizer.encode('ABCDE').tokens
```

```
['[CLS]', 'abc', '##d', '##e', '[SEP]']
```

```
bert_wordpiece_tokenizer.encode('ABCDE', add_special_tokens=False).tokens
```

```
['abc', '##d', '##e']
```

# tokenizers.BertWordPieceTokenizer

---

- BERT 는 두 문장을 [SEP] 으로 구분하며 학습하기 때문에 pair 기능을 제공합니다.

```
bert_wordpiece_tokenizer.encode(  
    sequence = 'abcde',  
    pair = 'abcd'  
).tokens
```

```
['[CLS]', 'abc', '##d', '##e', '[SEP]', 'abc', '##d', '[SEP]']
```



# tokenizers.BertWordPieceTokenizer

- 사용자가 직접 단어를 입력할 수 있습니다.
  - 하지만 `0.8.1`에서는 저장에 제대로 되지 않습니다.

```
bert_wordpiece_tokenizer.add_tokens(['lovit'])
bert_wordpiece_tokenizer.save_model(
    directory = './',
    name = 'very_small_bertwordpiece_lovit')
bert_wordpiece_tokenizer = BertWordPieceTokenizer(
    vocab_file = './very_small_bertwordpiece_lovit-vocab.txt'
)
vocab = bert_wordpiece_tokenizer.get_vocab()
print(sorted(vocab, key=lambda x: vocab[x]))
```

```
['[PAD]', '[UNK]', '[CLS]', '[SEP]', '[MASK]', 'a', 'b', 'c', 'd', 'e', 'f', 'g', '##b', '##c', '##d', '##e', '##f', 'ab', 'abc', 'ac']
```

# tokenizers.BertWordPieceTokenizer

---

- 하지만 `0.8.1` 에서는 저장이 제대로 되지 않습니다.
- 대신 `vocab.txt` 파일에 직접 단어를 추가하면 정상적으로 작동합니다.

```
bert_wordpiece_tokenizer = BertWordPieceTokenizer(  
    vocab_file = './very_small_bertwordpiece_lovit2-vocab.txt')  
bert_wordpiece_tokenizer.encode('ABCDE abg lovit').tokens
```

```
['[CLS]', 'abcde', '[UNK]', 'lovit', '[SEP]']
```

# tokenizers.SentencePieceBPETokenizer

- SentencePiece 는 공백 뒤에 등장하는 단어 앞에 `_` 를 붙여 실제 공백과 subwords 의 경계를 구분합니다.

```
sentencepiece_tokenizer = SentencePieceBPETokenizer(
    add_prefix_space = True)
sentencepiece_tokenizer.train(
    files = [small_corpus],
    vocab_size = 20,
    min_frequency = 1,
    special_tokens = ['<unk>'],)
vocab = sentencepiece_tokenizer.get_vocab()
print(sorted(vocab, key=lambda x: vocab[x]))
```

```
['<unk>', 'A', 'B', 'C', 'D', 'E', 'F', '_', '_A', '_AB', '_ABC', 'DE', '_DE', '_AC', '_AF',
 '_ABD', '_ABCDE']
```

# tokenizers.SentencePieceBPETokenizer

- `add_prefix_space=True` 이면 문장의 맨 앞 단어에도 공백을 부여, `False` 이면 공백없이 시작하는 단어에는 `_` 를 붙이지 않습니다.

```
sentencepiece_tokenizer = SentencePieceBPETokenizer(  
    add_prefix_space = False  
)  
sentencepiece_tokenizer.train(  
    files = [small_corpus],  
    vocab_size = 20,  
    min_frequency = 1,  
    special_tokens = ['<unk>', 'lovit'],  
)  
vocab = sentencepiece_tokenizer.get_vocab()  
print(sorted(vocab, key=lambda x: vocab[x]))
```

```
['<unk>', 'lovit', 'A', 'B', 'C', 'D', 'E', 'F', '_', '_A', '_AB', 'DE', '_ABC', 'AB', 'CD  
E', '_AC', '_AF', '_ABD', 'ABCDE']
```

# tokenizers.SentencePieceBPETokenizer

- BPE 를 이용하는 토큰나이저들은 `vocab.json` 과 `merges.txt` 두 개의 파일을 저장합니다.
- 학습된 토큰나이저를 이용할 때에도 두 개의 파일을 모두 입력해야 합니다.

```
sentencepiece_tokenizer.save_model('./', 'very_small_sentencepiece')
```

```
['./very_small_sentencepiece-vocab.json',  
 './very_small_sentencepiece-merges.txt']
```

```
sentencepiece_tokenizer = SentencePieceBPETokenizer(  
    vocab_file = './very_small_sentencepiece-vocab.json',  
    merges_file = './very_small_sentencepiece-merges.txt'  
)  
sentencepiece_tokenizer.encode('ABCDE').tokens
```

```
['_ABC', 'DE']
```

# tokenizers.CharBPETokenizer

- Character-level BPE 는 단어 수준에서 BPE 를 이용하여 subwords 를 학습하며, 단어에 suffix 로 </w> 를 부착하여 공백을 표현합니다.

```
charbpe_tokenizer = CharBPETokenizer(  
    suffix='</w>',  
    split_on_whitespace_only = True  
)  
charbpe_tokenizer.train(  
    files = [small_corpus],  
    vocab_size = 15,  
    min_frequency = 1  
)  
charbpe_tokenizer.encode('ABCDE.ABC').tokens
```

```
['AB', 'C', 'D', 'E', 'ABC</w>']
```

# tokenizers.CharBPETokenizer

---

- CharBPETokenizer 의 기본값은 공백/구두점으로 pre-tokenization 을 수행합니다.

```
charbpe_tokenizer = CharBPETokenizer(suffix='</w>')
charbpe_tokenizer.train(
    files = [small_corpus],
    vocab_size = 15,
    min_frequency = 1
)
charbpe_tokenizer.encode('ABCDE.ABC').tokens
```

```
['AB', 'C', 'DE</w>', 'ABC</w>']
```

# tokenizers.ByteLevelBPETokenizer

- Byte-level BPE 는 글자가 아닌 byte 기준으로 BPE 를 적용하기 때문에 1 byte 로 표현되는 글자 (알파벳, 숫자, 기호) 만 형태가 보존됩니다.

```
bytebpe_tokenizer = ByteLevelBPETokenizer()
bytebpe_tokenizer.train(files = [small_corpus],
                        vocab_size = 1000, min_frequency = 1)
vocab = bytebpe_tokenizer.get_vocab()
print(sorted(vocab, key=lambda x: vocab[x]))
```

```
[ '!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.', '/', '0', '1', '2',
'3', '4', '5', '6', '7', '8', '9', ':', ';', '<', '=', '>', '?', '@', 'A', 'B', 'C', 'D',
'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V',
'W', 'X', 'Y', 'Z', '[', '\\', ']', '^', '_', '`', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
'{', '|', '}', '~', 'ı', 'ç', '£', '¤', '¥', '¦', '§', '¨', '©', 'ª', «, ¬, ®, ¯,
°', ±, ², ³, ´, µ, ¶, ·, ¸, ¹, º, » ¼, ½, ¾, ¿, À, Á,
Â, Ã, Ä, Å, Æ, Ç, È, É, Ê, Ë, Ì, Í, Î, Ï, Ð, Ñ, Ò, Ó,
Ô, Õ, Ö, ×, Ø, Ù, Ú, Û, Ü, Ý, Þ, ß, à, á, â, ã, ä, å,
æ, ç, è, é, ê, ë, ì, í, î, ï, ð, ñ, ò, ó, ô, õ, ö, ÷,
ø, ù, ú, û, ü, ý, þ, ÿ, Ā, ā, Ă, ă, Ą, ą, Ć, ć, Ĉ, ĉ,
Ċ, ċ, Č, č, Ď, ď, Đ, đ, Ē, ē, Ĕ, ĕ, Ė, ė, Ę, ę, Ě, ě,
Ĝ, ĝ, Ğ, ğ, Ġ, ġ, Ģ, ģ, Ĥ, ĥ, Ħ, ħ, Ĩ, ĩ, Ī, ī, Ĭ, ĭ,
```



# tokenizers.ByteLevelBPETokenizer

---

- 띄어쓰기로 시작하는 단어 앞에 Ġ 를 prefix 로 부착합니다.

```
bytebpe_tokenizer.encode('ABCDE ABC').tokens
```

```
['ĠABCDE', 'ĠABC']
```

# tokenizers + COVID19 뉴스 학습

---

- 네 종류의 토크나이저 모두 다음처럼 7만 문장의 코로나 관련 뉴스로부터 3000 개의 vocab 을 학습합니다.

```
bert_wordpiece_tokenizer = BertWordPieceTokenizer()  
bert_wordpiece_tokenizer.train(  
    files=[corpus_path], vocab_size=vocab_size)  
bert_wordpiece_tokenizer.save_model(  
    directory='./tokenizers/BertWordPieceTokenizer/', name='covid')
```

```
['./tokenizers/BertWordPieceTokenizer/covid-vocab.txt']
```

# tokenizers + COVID19 뉴스 학습

- 한 문장에 대하여 각각의 토큰라이저를 적용한 결과를 살펴봅니다.

```
sent_ko = '신종 코로나바이러스 감염증(코로나19) 사태가 심각합니다'
tokenizers = [bert_wordpiece_tokenizer,
               sentencepiece_bpe_tokenizer,
               char_bpe_tokenizer,
               byte_level_bpe_tokenizer]

for tokenizer in tokenizers:
    encode_single = tokenizer.encode(sent_ko)
    print(f'\n{tokenizer.__class__.__name__}')
    print(f'tokens = {encode_single.tokens}')
```

BertWordPieceTokenizer

tokens = ['신종', '코로나바이러스', '감염증', '(', '코로나19', ')', '사태', '##가', '심', '##각', '##합니다']

SentencePieceBPETokenizer

tokens = ['\_신종', '\_코로나바이러스', '\_감염증(코로나19)', '\_사태', '\_가', '\_심', '\_각', '\_합', '\_니다']

CharBPETokenizer

tokens = ['신종</w>', '코로나바이러스</w>', '감염증</w>', '(</w>', '코로나19</w>', ')</w>', '사태', '가</w>', '심', '각', '합니다</w>']

ByteLevelBPETokenizer

tokens = ['iĩiĩçh', 'Ġi½kë;IëĤÍë°kİl'ëł-iĩı', 'Ġë°Uİı½i;L', '(', 'i½kë;IëĤÍ', '19', ')', 'ĠiĤ-iĤI', 'ë°ğ', 'Ġiĩ-', 'ë°ğ', 'iķ@ëiĩiııı']

# 학습한 토큰나이를 transformers 에서 이용하기

- 학습을 완료한 vocab 을 transformers 의 PreTrainedTokenizer 에서 이용합니다.
  - transformers 에서 모델들과 함께 이용되는 토큰나이저 (BertTokenizer, GPT2Tokenizer, ...) 는 PreTrainedTokenizer 를 상속합니다.

```
from transformers import BertTokenizer

transformers_bert_tokenizer = BertTokenizer(
    vocab_file = './tokenizers/BertWordPieceTokenizer/covid-vocab.txt'
)
print(f'tokenizers : {bert_wordpiece_tokenizer.encode(sent_ko).tokens}')
print(f'transformers: {transformers_bert_tokenizer.tokenize(sent_ko)}')
```

```
tokenizers : ['신종', '코로나바이러스', '감염증', '(', '코로나19', ')', '사태', '##가', '심', '##각', '##합니다']
transformers: ['신종', '코로나바이러스', '감염증', '(', '코로나19', ')', '사태', '##가', '심', '##각', '##합니다']
```

# unicodedata.normalize

---

- 'NFKD' 는 한글의 자/모를 분해, 'NFKC' 는 자/모를 한글로 조합합니다.

```
from unicodedata import normalize

print(normalize('NFKD', '가감')) # 출력 시 글자를 재조합해서 보여줌
print(len(normalize('NFKD', '가감')))
print(normalize('NFKC', normalize('NFKD', '가감')))
print(len(normalize('NFKC', normalize('NFKD', '가감'))))
```

```
가감
5
가감
2
```

# unicodedata.normalize

---

- 화면에서 한글로 보이지만 실제로는 자/모 분해가 된 글자입니다

```
for token in bert_wordpiece_tokenizer.encode(sent_ko).tokens[:3]:  
    print(f'len({token}) = {len(token)}')
```

len(신종) = 6

len(코로나바이러스) = 14

len(감염증) = 9

# unicodedata.normalize

---

- 매번 NFKC 로 후처리를 반복해야 한다면 간단한 함수를 만들어 뒀도 좋습니다.

```
from unicodedata import normalize

def compose(tokens):
    return [normalize('NFKC', token) for token in tokens]
```

KoNLPy 를 pre-tokenizer 로 이용하기



- 앞서 subword tokenizers 는 형태소 분석기능을 이용할 수 없음을 확인했습니다.
- 활용된 용언 하다는 다양한 표현을 (합니다→ 하+ㅂ니다, 했다→ 하+았다) 로 복원하며 subword tokenizers 를 학습하고 싶습니다.

어절	Mecab	Komorani	Bert + Covid news
심각합니다	심각 합니다	심각 하 ㅂ니다	심 ##각 ##합니다
심각했다	심각 했 다	심각 하 었 다	심 ##각 ##했다
심각하다고	심각 하 다고	심각 하 다고	심 ##각 ##하다 ##고
심각하며	심각 하 며	심각 하 며	심 ##각 ##하며

# failure case

---

- 첫번째 생각할 수 있는 방법은 형태소분석기로 pre-tokenization 을 수행한 뒤 그 결과를 tokenizers 의 input 으로 입력하는 것입니다.

```
print(f'input : {sent_ko}')
print(f'mecab pretok : {pretokenized}')
print(f'mecab + bert : {split_tokens}')
```

```
input : 신종 코로나바이러스 감염증(코로나19) 사태가 심각합니다
mecab pretok : 신종 코로나 바이러스 감염증 ( 코로나 19 ) 사태 가 심각 합니다
mecab + bert : 신종 코로나 바이러스 감염증 ( 코로나 19 ) 사태 가 심 ##각 합 ##니다
```

```
from tokenizers import BaseTokenizer

class BertWordPieceTokenizer(BaseTokenizer):
    def __init__(self, ...):
        # ...
        tokenizer.normalizer = BertNormalizer(...)
        tokenizer.pre_tokenizer = BertPreTokenizer()
```

# failure case

---

- transformers 는 Rust 로 구현된 토크나이저를 Python 에서 사용할 수 있도록 도와주는 패키지입니다. Normalizer 와 PreTokenizer 는 Rust 는 상속이 되지 않습니다.

```
from tokenizers.normalizers import Normalizer

class KomoranNormalizer(Normalizer):
    def __init__(self):
        print('success')
"type 'tokenizers.normalizers.Normalizer' is not an acceptable base type"

from tokenizers.pre_tokenizers import PreTokenizer
class KomoranPreTokenizer(PreTokenizer):
    def __init__(self):
        print('success')
"type 'tokenizers.pre_tokenizers.PreTokenizer' is not an acceptable base type"
```

## failure case

---

- transformers 는 Rust 로 구현된 토크나이저를 Python 에서 사용할 수 있도록 도와주는 패키지입니다. Normalizer 와 PreTokenizer 는 Rust 는 상속이 되지 않습니다.
- normalizer 와 pretokenizer 를 수정하지 않으면 train 과정에 영향을 줄 수 없습니다.

# KoNLPy 로 학습데이터에 pre-tokenization 수행

- `str` → `str` 로 처리하는 `pretokenizer` 를 만듭니다.

```
class KoNLPyPreTokenizer:
    def __init__(self, konlpy_tagger):
        self.konlpy_tagger = konlpy_tagger
    def __call__(self, sentence):
        return self.pre_tokenize(sentence)
    def pre_tokenize(self, sentence):
        return ' '.join(self.konlpy_tagger.morphs(sentence))

mecab_pretok = KoNLPyPreTokenizer(Mecab())
print(f'input : {sent_ko}\npretok : {mecab_pretok(sent_ko)}')
```

input : 신종 코로나바이러스 감염증(코로나19) 사태가 심각합니다

pretok : 신종 코로나 바이러스 감염증 ( 코로나 19 ) 사태 가 심각 합니다

# KoNLPy 로 학습데이터에 pre-tokenization 수행

---

- Mecab 을 이용하여 학습데이터의 어절을 형태소 단위로 구분합니다.

```
def prepare_pretokenized_corpus(raw_path, pretokenized_path, pretok):
    with open(raw_path, encoding='utf-8') as f:
        lines = [line.strip() for line in f]
    with open(pretokenized_path, 'w', encoding='utf-8') as f:
        for line in lines:
            f.write(f'{pretok(line)}\n')

prepare_pretokenized_corpus(
    raw_path = '../data/2020-07-29_covid_news_sents.txt',
    pretokenized_path = '../data/2020-07-29_covid_news_sents.mecab.txt',
    KoNLPyPreTokenizer(Mecab())
)
```

# KoNLPy 로 학습데이터에 pre-tokenization 수행

---

- 학습은 BertWordPieceTokenizer 를 이용할 수 있습니다.

```
from tokenizers import BertWordPieceTokenizer

bert_wordpiece_tokenizer = BertWordPieceTokenizer()
bert_wordpiece_tokenizer.train(
    files = ['./data/2020-07-29_covid_news_sents.mecab.txt'],
    vocab_size = 3000
)
bert_wordpiece_tokenizer.save_model(
    directory='./tokenizers/MecabBertWordPieceTokenizer/',
    name='covid'
)
```

```
['./tokenizers/MecabBertWordPieceTokenizer/covid-vocab.txt']
```

# tokenizers.BaseTokenizer 에 pretok 적용

- BertWordPieceTokenizer 를 상속받아 encode(), encode\_batch() 함수를 수정합니다.

```
class KoNLPyPretokBertWordPieceTokenizer(BertWordPieceTokenizer):
    def __init__(self, konlpy_pretok, vocab_file, ...):
        super().__init__(...)
        self.konlpy_pretok = konlpy_pretok

    def encode(self, sequence, ...):
        # ...
        sequence = self.konlpy_pretok(sequence)
        encoding = self._tokenizer.encode(
            sequence, pair, is_pretokenized, add_special_tokens)
        return encoding

    def encode_batch(self, inputs, ...):
        # ...
        input_iterator = tqdm(inputs, desc='konlpy pretok', total=len(inputs))
        konlpy_pretok_inputs = [
            self.konlpy_pretok(sequence) for sequence in input_iterator]
        encodings = self._tokenizer.encode_batch(
            konlpy_pretok_inputs, is_pretokenized, add_special_tokens)
        return encodings
```



# transformers.PreTrainedTokenizer 에 pretok 적용

- BertTokenizer 를 상속받아 `_tokenize()` 를 변경

```
from transformers import BertTokenizer

class KoNLPyPretokBertTokenizer(BertTokenizer):
    def __init__(self, konlpy_pretok, vocab_file, ...):
        super().__init__(...)
        self.konlpy_pretok = konlpy_pretok

    def _tokenize(self, text):
        text = self.konlpy_pretok(text)
        split_tokens = []
        if self.do_basic_tokenize:
            for token in self.basic_tokenizer.tokenize(
                text, never_split=self.all_special_tokens):
                # If the token is part of the never_split set
                if token in self.basic_tokenizer.never_split:
                    split_tokens.append(token)
                else:
                    split_tokens += self.wordpiece_tokenizer.tokenize(token)
        else:
            split_tokens = self.wordpiece_tokenizer.tokenize(text)
        return split_tokens
```

# Hugging Face + KoNLPy (pretok)

---

- 앞의 코드들을 패키지로 정리해 두었습니다. [https://github.com/lovit/huggingface\\_konlpy](https://github.com/lovit/huggingface_konlpy)

```
from huggingface_konlpy.tokenizers_konlpy import KoNLPyPreTokenizer
from konlpy.tag import Komoran

sent_ko = '신종 코로나바이러스 감염증(코로나19) 사태가 심각합니다'
komoran_pretok = KoNLPyPreTokenizer(Komoran())
print(komoran_pretok(sent_ko))
```

신종 코로나바이러스 감염증 ( 코로나 19 ) 사태 가 심각 하 ㅂ니다

# Hugging Face + KoNLPy (pretok)

```
from huggingface_konlpy.tokenizers_konlpy import KoNLPyPretokBertWordPieceTokenizer
from huggingface_konlpy.transformers_konlpy import KoNLPyPretokBertTokenizer
from huggingface_konlpy import compose

komoran_bertwordpiece_tokenizer = KoNLPyPretokBertWordPieceTokenizer(
    konlpy_pretok = komoran_pretok)

komoran_bertwordpiece_tokenizer.train(
    files = ['./data/2020-07-29_covid_news_sents.txt'],
    vocab_size = 3000)
komoran_bertwordpiece_tokenizer.save_model(
    directory='./tokenizers/KomoranBertWordPieceTokenizer/',
    name='covid')

komoran_pretok_berttokenizer = KoNLPyPretokBertTokenizer(
    konlpy_pretok = komoran_pretok,
    vocab_file = './tokenizers/KomoranBertWordPieceTokenizer/covid-vocab.txt')

indices = komoran_pretok_berttokenizer.encode(sent_ko)
tokens = [komoran_pretok_berttokenizer.ids_to_tokens[ids] for ids in indices]
print(' '.join(compose(tokens)))
```

# Custom 기능이 제공되지 않는가?

---

- 2020.4 에 `custom_pre_tokenizer` [예시 코드](#)가 업로드 되었습니다.
  - 하지만 정상적으로 작동하지 않습니다.
  - `tokenizers` 가 계속 버전업 중이니 기능에 추가 될 것이라 기대하고 있습니다.

```
from tokenizers import pre_tokenizers

class GoodCustom:
    def pre_tokenize(self, sentence):
        return sentence.split(" ")

    def decode(self, tokens):
        return ", ".join(tokens)

good_custom = GoodCustom()
good_pretok = pre_tokenizers.PreTokenizer.custom(good_custom)
```

KoNLPy 를 WordPiece 로 이용하기

## Input

신종 코로나바이러스 감염증(코로나19) 사태가 **심각합니다**

## KoNLPy (pretok) + Bert Tokenizer

[CLS] 신종 코로나 ##바 ##이 ##러스 감염증 ( 코로나 19 ) 사태 가 심각 하 [UNK] [SEP]

어절의 중간에 위치하기에  
"##하" 로 표시하고 싶습니다

KoNLPy 의 품사 정보도 함께  
이용하고 싶습니다.

학습되지 않은 단어는 글자로  
분해하고 싶습니다

- 
- 어절 단위로 KoNLPy 를 적용하면 "##하"를 표현할 수 있습니다.

KoNLPy (pretok) + Bert Tokenizer

[CLS] 신종 코로나 ##바 ##이 ##러스 감염증 ( 코로나 19 ) 사태 가 심각 **하** [UNK] [SEP]

어절의 중간에 위치하기에  
"##하" 로 표시하고 싶습니다



- 
- 어절 단위로 KoNLPy 를 적용한 뒤, 알려지지 않은 형태소를 글자로 분리합니다.
    - 어절 단위로 KoNLPy 를 적용하면 문맥 정보를 이용하지 못하지만, 그렇지 않더라도 충분히 토큰라이징이 잘 이뤄집니다 (subword tokenizer 도 문맥 정보를 이용하지 않습니다)

KoNLPy (pretok) + Bert Tokenizer

[CLS] 신종 코로나 ##바 ##이 ##러스 감염증 ( 코로나 19 ) 사태 가 심각 하 [UNK] [SEP]

학습되지 않은 단어는 글자로  
분해하고 싶습니다



# KoNLPy as Word Piece Tokenizer

- 어절 단위로 KoNLPy 를 적용한 뒤, 두번째 morph 부터 prefix ## 을 부착합니다.

```
class KoNLPyWordPieceTokenizer:
    def __init__(self, konlpy_tagger, wordpieces_prefix="##"):
        self.konlpy_tagger = konlpy_tagger

    def pretokenize(eojeol):
        return self.konlpy_tagger.morphs(eojeol)

    self.pretokenize = pretokenize
    self.prefix = wordpieces_prefix

    def tokenize(self, sent):
        def _tokenize(eojeol):
            split_tokens = self.pretokenize(eojeol)
            if len(split_tokens) <= 1:
                return split_tokens
            return [split_tokens[0]] + [f'{self.prefix}{sub}' for sub in split_tokens[1:]]

        split_tokens = []
        for eojeol in sent.split():
            split_tokens += _tokenize(eojeol)
        return split_tokens
```

# KoNLPy as Word Piece Tokenizer

- 품사 정보를 이용하고 싶다면 `morphs()` 대신 `pos(eojeol, join=True)` 를 이용합니다.

```
class KoNLPyWordPieceTokenizer:
    def __init__(self, konlpy_tagger, wordpieces_prefix="##"):
        self.konlpy_tagger = konlpy_tagger

    def pretokenize(eojeol):
        return self.konlpy_tagger.pos(eojeol, join=True)

    self.pretokenize = pretokenize
    self.prefix = wordpieces_prefix

    def tokenize(self, sent):
        def _tokenize(eojeol):
            split_tokens = self.pretokenize(eojeol)
            if len(split_tokens) <= 1:
                return split_tokens
            return [split_tokens[0]] + [f'{self.prefix}{sub}' for sub in split_tokens[1:]]

        split_tokens = []
        for eojeol in sent.split():
            split_tokens += _tokenize(eojeol)
        return split_tokens
```

# KoNLPy as Word Piece Tokenizer

---

- prefix ## 을 제거하고 KoNLPy 의 출력 morph[/tag] 를 글자열로 만들 준비도 합니다.

```
class KoNLPyWordPieceTokenizer:
    # ...
    def token_to_alphabets(self, token):
        if token[:2] == self.prefix:
            token = token[2:]
        if self.use_tag:
            return list(token.rsplit('/')[0])
        return list(token)
```

# KoNLPy as Word Piece Tokenizer

- tokenizers.BaseTokenizer 를 상속하지 않았으니 train() 함수를 추가로 구현해 봅니다.

```
class KoNLPyBertWordPieceTrainer:
    def __init__(self, konlpy_tagger, wordpieces_prefix="##", use_tag=False):
        konlpy_wordpiece = KoNLPyWordPieceTokenizer(
            konlpy_tagger, wordpieces_prefix, use_tag)
        self.tokenizer = konlpy_wordpiece

    def train(
        self,
        files: Union[str, List[str]],
        vocab_size: int = 30000,
        min_frequency: int = 2,
        limit_alphabet: int = 1000,
        initial_alphabet: List[str] = [],
        special_tokens: List[str] = ["[PAD]", "[UNK]", "[CLS]", "[SEP]", "[MASK]"],
        show_progress: bool = True,
    ):
        if isinstance(files, str):
            files = [files]

        alphabets = initialize_alphabet(files, limit_alphabet,
            initial_alphabet, special_tokens, show_progress)
        self.vocab = train_vocab(files, vocab_size, min_frequency,
            show_progress, alphabets, self.tokenizer.tokenize)
```

# KoNLPy as Word Piece Tokenizer

---

- transformers.PreTrainedTokenizer 가 학습된 `vocab.txt` 와 KoNLPy 를 WordPiece 대신 이용할 수 있도록 만듭니다. base\_tokenizer 만 KoNLPyWordPiece 로 교체합니다.

```
class KoNLPyBertTokenizer(BertTokenizer):
    def __init__(self, konlpy_wordpiece, vocab_file, ...):
        super().__init__(...)
        self.konlpy_wordpiece = konlpy_wordpiece

    def _tokenize(self, text):
        base_tokens = self.konlpy_wordpiece.tokenize(text)
        split_tokens = []
        for token in base_tokens:
            if token in self.vocab:
                split_tokens.append(token)
            else:
                split_tokens += self.konlpy_wordpiece.token_to_alphabets(token)
        return split_tokens
```

# KoNLPy as Word Piece Tokenizer

- 정리된 클래스를 이용해 봅니다.

```
from huggingface_konlpy.tokenizers_konlpy import KoNLPyBertWordPieceTrainer

mecab_wordpiece_notag_trainer = KoNLPyBertWordPieceTrainer(
    Mecab(), use_tag=False)
mecab_wordpiece_notag_trainer.train(
    files = ['./data/2020-07-29_covid_news_sents.txt'])
mecab_wordpiece_notag_trainer.save_model('./tokenizers/BertStyleMecab/', 'notag')
```

```
Initialize alphabet 1/1: 100%|██████████| 70964/70964 [00:00<00:00, 83278.38it/s]
Train vocab 1/1: 100%|██████████| 70964/70964 [00:14<00:00, 4853.12it/s]
```

```
from huggingface_konlpy.transformers_konlpy import KoNLPyBertTokenizer

konlpy_bert_notag = KoNLPyBertTokenizer(
    konlpy_wordpiece = KoNLPyWordPieceTokenizer(Mecab(), use_tag=False),
    vocab_file = './tokenizers/BertStyleMecab/notag-vocab.txt'
)
print(' '.join(konlpy_bert_notag.tokenize(sent_ko)))
```

# KoNLPy as Word Piece Tokenizer

- Mecab 의 품사 정보도 이용해 봅니다.

```
mecab_wordpiece_usetag_trainer = KoNLPyBertWordPieceTrainer(Mecab(), use_tag=True)
mecab_wordpiece_usetag_trainer.train(
    files = ['./data/2020-07-29_covid_news_sents.txt'])
mecab_wordpiece_usetag_trainer.save_model('./tokenizers/BertStyleMecab/', 'usetag')

konlpy_bert_usetag = KoNLPyBertTokenizer(
    konlpy_wordpiece = KoNLPyWordPieceTokenizer(Mecab(), use_tag=True),
    vocab_file = './tokenizers/BertStyleMecab/usetag-vocab.txt')

indices = konlpy_bert_usetag.encode(sent_ko)
tokens = [konlpy_bert_usetag.ids_to_tokens[ids] for ids in indices]
print(' '.join(compose(tokens)))
```

```
Initialize alphabet 1/1: 100%|██████████| 70964/70964 [00:00<00:00, 79650.13it/s]
Train vocab 1/1: 100%|██████████| 70964/70964 [00:14<00:00, 4824.97it/s]
```

```
[CLS] 신종/NNG 코로나/NNP ##바이러스/NNG 감염증/NNG ##(/SSO ##코로나/NNP ##19/SN ##)/SSC 사태/NN
G ##가/JKS 심각/XR 합 니 다 [SEP]
```

KoNLPy + BertTokenizer 로 BERT 학습하기



- 
- PreTrainedTokenizer 를 로딩할 함수를 만듭니다.

```
from konlpy.tag import Komoran, Mecab, Okt

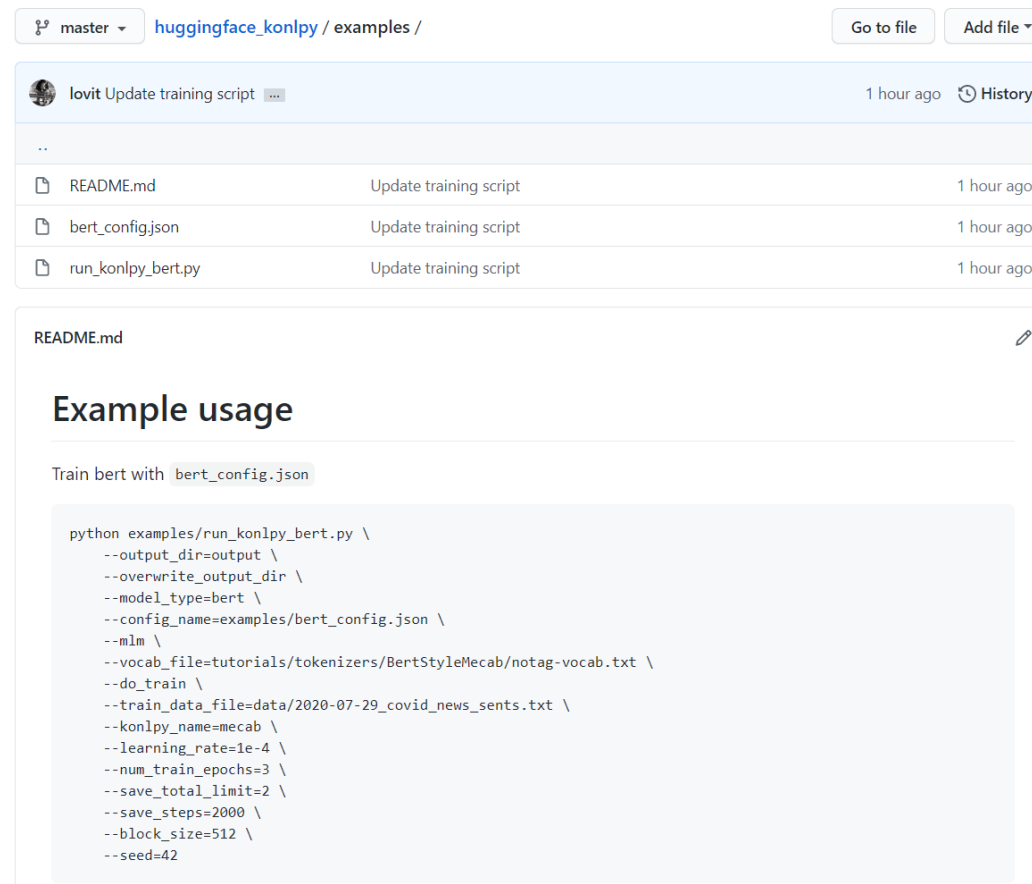
KONLPY = {'komoran': Komoran, 'mecab': Mecab, 'Okt': Okt}

def get_tokenizer(vocab_file, konlpy_name, use_tag=False):
    if konlpy_name not in KONLPY:
        raise ValueError(f'Support only {set(KONLPY.keys())}')
    konlpy_bert_tokenizer = KoNLPyBertTokenizer(
        konlpy_wordpiece = KoNLPyWordPieceTokenizer(
            KONLPY[konlpy_name](), use_tag=use_tag),
        vocab_file = vocab_file
    )
    return konlpy_bert_tokenizer
```

- transformers github 에서 다양한 tasks 의 학습 스크립트를 제공합니다.
  - > language-modeling 에 들어가시면 스크립트와 사용법이 적혀있습니다.

adversarial	Black 20 release	2 days ago
benchmarking	Black 20 release	2 days ago
bert-loses-patience	Black 20 release	2 days ago
bertology	Black 20 release	2 days ago
contrib	Black 20 release	2 days ago
deebert	Black 20 release	2 days ago
distillation	Black 20 release	2 days ago
language-modeling	Black 20 release	2 days ago
longform-qa	Black 20 release	2 days ago
movement-pruning	Black 20 release	2 days ago
multiple-choice	Black 20 release	2 days ago
question-answering	Fix it to work with BART (#6756)	yesterday
seq2seq	prepare_seq2seq_batch makes labels/ decoder_input_ids made later. (#6654)	2 hours ago
text-classification	Black 20 release	2 days ago
text-generation	Black 20 release	2 days ago
token-classification	Fix the TF Trainer gradient accumulation and the TF NER example (#6713)	yesterday

- 앞의 스크립트를 바탕으로 KoNLPy + BertTokenizer 를 이용하여 BERT 를 학습하는 스크립트를 아래의 github 에 올려두었습니다.



The screenshot shows a GitHub repository for 'huggingface\_konlpy' in the 'examples' directory. A commit by user 'lovit' titled 'Update training script' is shown, dated '1 hour ago'. The commit includes three files: 'README.md', 'bert\_config.json', and 'run\_konlpy\_bert.py', all updated '1 hour ago'. Below the commit list, the 'README.md' file is open, showing an 'Example usage' section. The usage instructions are to 'Train bert with bert\_config.json' and provide a list of command-line arguments for the 'run\_konlpy\_bert.py' script.

```
python examples/run_konlpy_bert.py \  
  --output_dir=output \  
  --overwrite_output_dir \  
  --model_type=bert \  
  --config_name=examples/bert_config.json \  
  --mlm \  
  --vocab_file=tutorials/tokenizers/BertStyleMecab/notag-vocab.txt \  
  --do_train \  
  --train_data_file=data/2020-07-29_covid_news_sents.txt \  
  --konlpy_name=mecab \  
  --learning_rate=1e-4 \  
  --num_train_epochs=3 \  
  --save_total_limit=2 \  
  --save_steps=2000 \  
  --block_size=512 \  
  --seed=42
```

- 
- tokenizers 튜토리얼/ huggingface\_konlpy / BERT 학습 스크립트는 아래의 주소에 있습니다.
    - [https://github.com/lovit/huggingface\\_konlpy](https://github.com/lovit/huggingface_konlpy)