# Solutions for Assignment 2

# Exercise 1 - Implementing a tokenizer

Implement a basic whitespace tokenizer in Python from scratch without the use of any NLP libraries. This tokenizer should drop whitespaces and create tokens for the following cases:

(a) End-of-sentence (EOS) symbols, brackets and separators

(b) Abbreviations - Assume those are only one of the following: Ph.D., Dr., M.Sc.

(c) Special characters as in prices separated (i.e. $45.55)

(d) Dates - Assume that they follow the format dd/mm/yy (i.e. 01/02/06)

(e) URLs - Assume that they follow the format:
`http[s]://[...]`, (i.e. https://www.stanford.edu)

(f) Hashtags separated (i.e. #nlproc)

(g) Email addresses - Assume that they follow the format:
name@domain.xyz (i.e. someOne@brown.edu)

Apply your code on the test example below, which should yield the specified tokens:

## Exercise 2 - Implementing a BPE tokenizer

Implement a Byte Pair Encoder (BPE) tokenizer as shown in the lecture and apply it to a sample text. You are free with your choice of libraries. You can assume that the corpus only consists of a list of words.

# Exercise 2 – BPE Learner

## BPE token learner algorithm (Lecture 2 – Slide 48)

**function** BYTE-PAIR ENCODING(strings $C$, number of merges $k$) **returns** vocab $V$

$V \leftarrow$ all unique characters in $C$ # initial set of tokens is characters
**for** $i = 1$ **to** $k$ **do** # merge tokens til $k$ times
$t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in $C$
$t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
$V \leftarrow V + t_{NEW}$ # update the vocabulary
Replace each occurrence of $t_L, t_R$ in $C$ with $t_{NEW}$ # and update the corpus
**return** $V$

# Exercise 2 – BPE Learner

BPE token learner algorithm (Lecture 2 – Slide 48)

```
def BPE_learner(corpus, num_merges):
```

**returns** vocab $V$

$V \leftarrow$ all unique characters in $C$ # initial set of tokens is characters
**for** $i = 1$ **to** $k$ **do** # merge tokens til $k$ times
$t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in $C$
$t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
date the vocabulary
ith $t_{NEW}$ # and update the corpus
**return** $V$

Comes with a few notes!

# Exercise 2 – BPE Learner

Notes (Lecture 2 - Slide 49):

1. Most subword algorithms are run inside space separated tokens.
2. So we commonly first add a special end of word symbol '_' before space in training corpus

# Exercise 2 – BPE Learner

**<u>Step 1 - Prepare the vocab:</u>**

1. space separation

2. Add EOW-symbol for each word

```python
def BPE_learner(corpus, num_merges):
    words = [word + "_" for word in corpus.split(" ")]
    vocab = list(set("".join(words).lower()))
```

3. Initial vocab:
all unique characters in corpus
(incl. EOW-symbol)

# Exercise 2 – BPE Learner

## BPE token learner algorithm

```
def BPE_learner(corpus, num_merges):
```
**returns** vocab $V$

$V \leftarrow$ all unique characters in $C$ # initial set of tokens is characters
**for** $i = 1$ **to** $k$ **do** # merge tokens til $k$ times
    $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in $C$
    $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
    $V \leftarrow V + t_{NEW}$ # update the vocabulary
    Replace each occurrence of $t_L, t_R$ in $C$ with $t_{NEW}$ # and update the corpus
**ret**

The actual longest part of the code!

# Exercise 2 – BPE Learner

**<u>Step 2 – Initialize the tokens:</u>**

Inital tokens should look like this (L2 Slide 48):

**representation**

1. Represent corpus as set of words

2. Compute word counts

| corpus | |
|---|---|
| 5 | l o w _ |
| 2 | l o w e s t _ |
| 6 | n e w e r _ |
| 3 | w i d e r _ |
| 2 | n e w _ |

3. Split each word into letters

# Exercise 2 – BPE Learner

**<u>Step 2 – Initialize the tokens:</u>**

1. Represent corpus as set of words

```python
    tokens = [list(token) for token in set(words)]
    vocab_count = collections.Counter(words)
```

2. Compute word counts

3. Split each word into letters

# Exercise 2 – BPE Learner

**Step 3 – Merge/Iterate k times:**

```
    for iteration in range(num_merges):
```

# Exercise 2 – BPE Learner

**Step 3.1 - Compute pair frequencies:**

| corpus | |
|---|---|
| 5 | l o w _ |
| 2 | l o w e s t _ |
| 6 | n e w e r _ |
| 3 | w i d e r _ |
| 2 | n e w _ |

**Pair-counts**

| | |
|---|---|
| 7 | lo |
| 5 | ow |
| 5 | w_ |

# Exercise 2 – BPE Learner

1. Create a counter for the pairs

**Step 3.1 - Compute pair frequencies:**

**corpus**

| | |
|---|---|
| 5 | l o w _ |
| 2 | l o w e s t _ |
| 6 | n e w e r _ |
| 3 | w i d e r _ |
| 2 | n e w _ |

**Pair-counts**

| | |
|---|---|
| 7 | lo |
| 5 | ow |
| 5 | w_ |

2. Iterate through each word in set

3. Iterate through each pair in word

4. Update counter for every pair

# Exercise 2 – BPE Learner

**<u>Step 3.1 - Compute pair frequencies:</u>**

```python
counts = {}
for token in tokens:
    word = "".join(token)
    for t_l, t_r in zip(token[:-1], token[1:]):
        pair = t_l + t_r
        if pair in counts.keys():
            counts[pair] += vocab_count[word]
        else:
            counts[pair] = vocab_count[word]
```

1. Create a counter for the pairs

2. Iterate through each word

3. Iterate through each pair

4. Update counter

# Exercise 2 – BPE Learner

**Intermission – exit condition**

All words are represented by a single token
-> Nothing can be merged anymore!

```
if len(counts) == 0:
    print("No more pairings possible")
    break
```

# Exercise 2 – BPE Learner

## BPE token learner algorithm

```
3       def BPE_learner(corpus, num_merges):
```

**returns** vocab $V$

$V \leftarrow$ all unique characters in $C$ # initial set of tokens is characters
**for** $i = 1$ **to** $k$ **do** # merge tokens til $k$ times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in $C$
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of $t_L, t_R$ in $C$ with $t_{NEW}$ # and update the corpus
**return** $V$

# Exercise 2 – BPE Learner

**Step 3.2 - Find most frequent pair:**

**corpus**

| | |
|---|---|
| 5 | l o w _ |
| 2 | l o w e s t _ |
| 6 | n e w e r _ |
| 3 | w i d e r _ |
| 2 | n e w _ |

**Pair-counts**

| | |
|---|---|
| 9 | er |
| 9 | r_ |
| 8 | ne |
| ⋮ | |
| 2 | es |

**Step 3.3: Append to vocab:**

**vocabulary**

_, d, e, i, l, n, o, r, s, t, w, er

# Exercise 2 – BPE Learner

**<u>Step 3.2 + 3.3:</u>**

1. Find most frequent pair

```
else:
    most_common_pair = max(counts, key=counts.get)
    print(f"most common pair: {most_common_pair}")
    vocab.append(most_common_pair)
```

2. Append to vocab

# Exercise 2 – BPE Learner

## BPE token learner algorithm

```
def BPE_learner(corpus, num_merges):
```
**returns** vocab $V$

$V \leftarrow$ all unique characters in $C$ # initial set of tokens is characters
**for** $i = 1$ **to** $k$ **do** # merge tokens til $k$ times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in $C$
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of $t_L, t_R$ in $C$ with $t_{NEW}$ # and update the corpus
**return** $V$

# Exercise 2 – BPE Learner

**<u>Step 3.4 - Merge all occurences of most frequent pair:</u>**

most frequent pair: er

**corpus**

| | |
|---|---|
| 5 | l o w _ |
| 2 | l o w e s t _ |
| 6 | n e w e r _ |
| 3 | w i d e r _ |
| 2 | n e w _ |

→

**corpus**

| | |
|---|---|
| 5 | l o w _ |
| 2 | l o w e s t _ |
| 6 | n e w er _ |
| 3 | w i d er _ |
| 2 | n e w _ |

# Exercise 2 – BPE Learner

**Step 3.4 - Merge all occurences of most frequent pair:**

most frequent pair:  er

**corpus**

| | |
|---|---|
| 5 | l o w _ |
| 2 | l o w e s t _ |
| 6 | n e w e r _ |
| 3 | w i d e r _ |
| 2 | n e w _ |

1. Iteratively find occurences...

**corpus**

| | |
|---|---|
| 5 | l o w _ |
| 2 | l o w e s t _ |
| 6 | n e w er _ |
| 3 | w i d er _ |
| 2 | n e w _ |

2. ... and merge them

# Exercise 2 – BPE Learner

**<u>Step 3.4 - Merge all occurences of most frequent pair:</u>**

```
for token in tokens:
    ind = 1
    while ind < len(token):
        if token[ind - 1] + token[ind] == most_common_pair:
            token[ind - 1] = most_common_pair
            token.pop(ind)
        else:
            ind += 1
```

Iterate over each word

1. find occurences

2. Merge them

# Exercise 2 – BPE Learner

## BPE token learner algorithm

```
def BPE_learner(corpus, num_merges):
```

**returns** vocab $V$

$V \leftarrow$ all unique characters in $C$ # initial set of tokens is characters
**for** $i = 1$ **to** $k$ **do** # merge tokens til $k$ times
$t_L, t_R \leftarrow$ Most fr
$t_{NEW} \leftarrow t_L + t_R$ en by concatenating
$V \leftarrow V + t_{NEW}$ lary
Replace each occurrence of $t_L, t_R$ in $C$ with $t_{NEW}$ # and update the corpus
**return** $V$

That‘s it!

# Exercise 2 – BPE segmenter

Lecture 2 – Slide 55:

On the test data, run each merge learned from the training data:

- Greedily
- In the order we learned them
- (test frequencies don't play a role)

# Exercise 2 – BPE segmenter

Algorithm:

1. Replace spaces with EOW-symbol
2. Split input into letters
3. Merge letters in learned order

# Exercise 2 – BPE segmenter

Input : low low

Vocab : {..., lo, low, newer_, low_, ...}

1. Replace spaces : low_low_
2. Split : l, o, w, _, l, o, w, _
3. Merge : lo, w, _, lo, w, _

   : low, _, low, _

   : low_, low_

# Exercise 2 – BPE segmenter

```
def BPE_tokenizer(text, vocab):  2 usages
    tokens = list(text.lower().replace(" ", "_"))
    i = 0
    while i < len(tokens):
        if not tokens[i] in vocab:
            tokens[i] = "UNK"
        i += 1
    for entry in vocab:
        if len(entry) == 1:
            continue
        else:
            i = 1
            while i < len(tokens):
                if tokens[i - 1] + tokens[i] == entry:
                    tokens[i - 1] = entry
                    tokens.pop(i)
                i += 1
    return tokens
```

Insert EOW and split

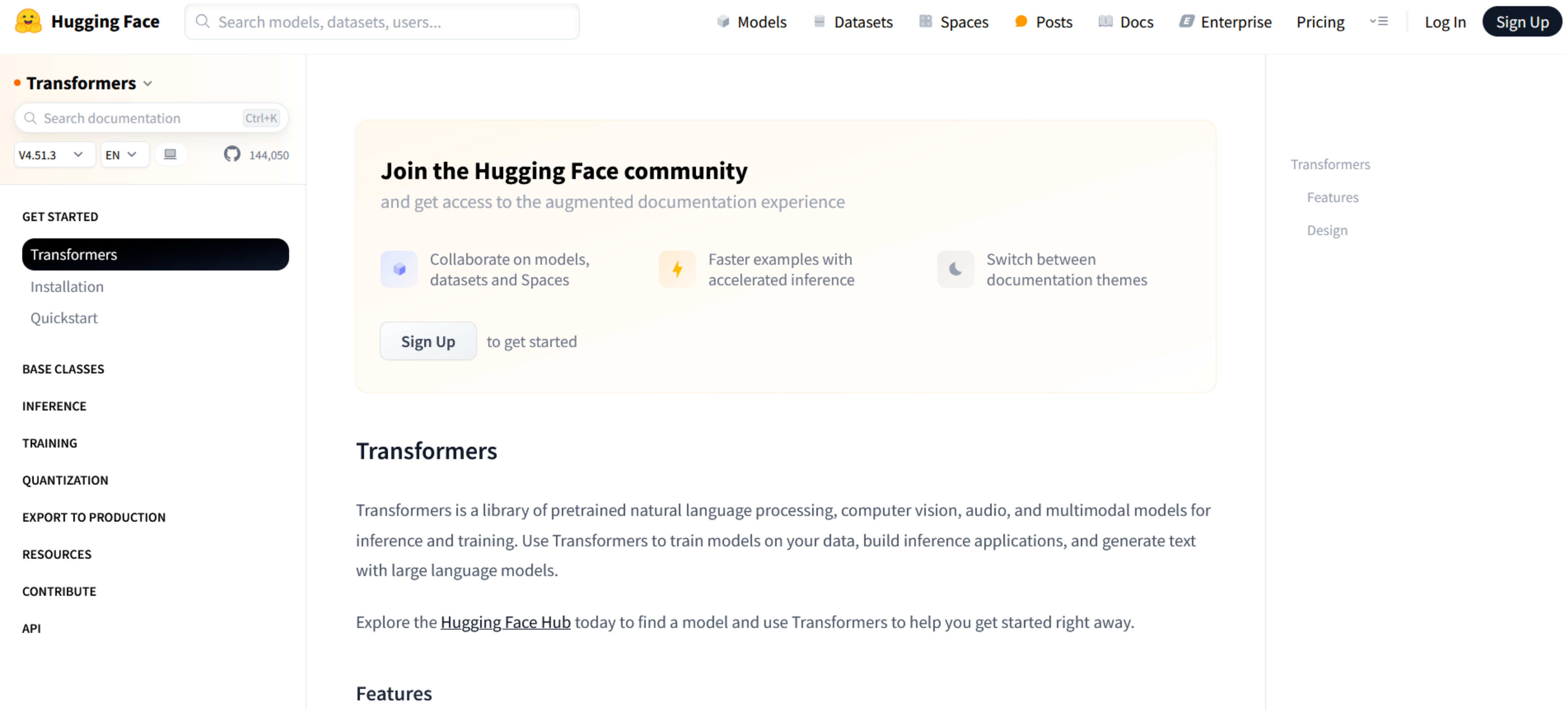
Deal with unknown letters (in general unneccessary)

Iteratively merge

## Exercise 3 - Using pre-implemented tokenizers

Use an existing tokenizer from the T5 Transformer or any other tokenizer of choice from the HuggingFace library. Apply the tokenizer to a text sample of choice. Compare the output of this tokenizer with the two tokenizers you implemented in the previous questions and explain the similarities and differences.

# Transformers Library



# Transformers Library

- Installation:

https://huggingface.co/docs/transformers/en/installation

- Quickstart

https://huggingface.co/docs/transformers/en/quicktour#pretrained-models

```
from transformers import AutoModelForCausalLM, AutoTokenizer

model = AutoModelForCausalLM.from_pretrained("meta-llama/Llama-2-7b-hf",
tokenizer = AutoTokenizer.from_pretrained("meta-llama/Llama-2-7b-hf")
```

# Transformers Library

- We want the „T5“ transformer

**from_pretrained** <source>

```
( pretrained_model_name_or_path: typing.Union[str, os.PathLike, NoneType], *model_args, config: typing.Union[transformers.configuration_utils.PretrainedConfig, str, os.PathLike, NoneType] = None, cache_dir: typing.Union[str, os.PathLike, NoneType] = None, ignore_mismatched_sizes: bool = False, force_download: bool = False, local_files_only: bool = False, token: typing.Union[str, bool, NoneType] = None, revision: str = 'main', use_safetensors: typing.Optional[bool] = None, weights_only: bool = True, **kwargs )
```

**Parameters**

- **pretrained_model_name_or_path** (`str` or `os.PathLike`, *optional*) — Can be either:
  - A string, the *model id* of a pretrained model hosted inside a model repo on huggingface.co.
  - A path to a *directory* containing model weights saved using save_pretrained(), e.g., `./my_model_directory/`.

# Transformers Library

- Need T5 transformer

**pretrained_model_name_or_path** (`str` or `os.PathLike`, *optional*) — Can be either:

- A string, the *model id* of a pretrained model hosted inside a model repo on huggingface.co.

- Head to https://huggingface.co/google-t5

# Transformers Library

**Models** 5 | Sort: Recently updated

**google-t5/t5-base**
Translation • Updated Feb 14, 2024 • 4.3M • • 708

**google-t5/t5-3b**
Translation • Updated Jan 29, 2024 • 357k • 46

**google-t5/t5-small**
Translation • Updated Jun 30, 2023 • 3.02M • • 455

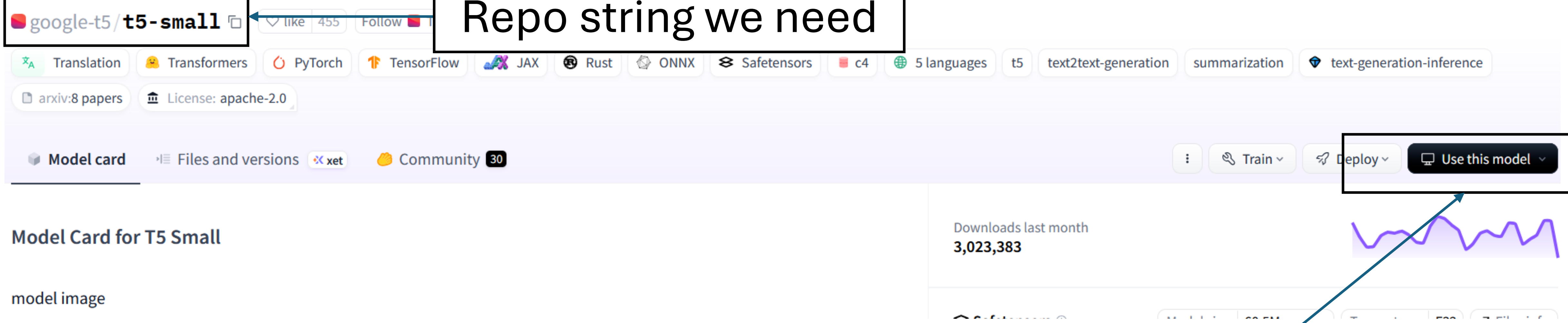
**google-t5/t5-large**
Translation • Updated Apr 6, 2023 • 436k • • 205

**google-t5/t5-11b**

Choose any

# Transformers Library

Repo string we need




Click for usage reference

https://huggingface.co/google-t5/t5-small

# Exercise 3

```
from transformers import T5Tokenizer

tokenizer = T5Tokenizer.from_pretrained("google-t5/t5-small")
test_text = "He has a M.Sc. in Math and a Ph.D. in NLP. A sess
#print(tokenizer.convert_ids_to_tokens(tokenizer(test_text).ir
print(tokenizer.tokenize(test_text))
```

# Exercise 3 - Similarities

…with WS-tokenizer:

- Mostly assigns (EOS-)symbols their own token
- Most splits happen on word-level

… with BPE:

- (T5 uses an extension of BPE called ‚unigram‘ or ‚SentencePiece‘)
- Learns (sub-)word splits
- Vocab size fixed before training
- Tries to minimize sequence length (token count)

# Exercise 3 - Differences

…with WS-tokenizer:

- Splits are learned and not rule-based
- Does not care about abbrevations, links, emails, dates, …
- Includes spaces in the tokens

… with BPE:

- Start-of-word instead of EOW-symbol
- Has some (model specific) special tokens (e.g. end-of-sequence </s>
- (Trims a „seed“ vocabulary (i.e. built using BPE) instead of building one)

## Exercise 4 - RegEx

Assume we have a lookup table named `lookup` storing abbreviation definitions. Using regular expressions in Python, write code that uses `lookup` to replace abbreviations in any given text with their full-text counterparts. Apply your code to the following snippet so that `example` is transformed to match `target_output`:

# Exercise 4 - RegEx

Abbr. could be followed by an apostrophe

```
def convert_to_fulltext(text):
    for abbr, ft in lookup.items():
        text = re.sub(pattern: fr"\b{abbr}'?\b", ft, text)
    return text
```

Abbr. may not be preceded or followed by word characters (match „bf“ but not „bff“)

„\b“: word boundary; matches anything but [A-Za-z0-9_]

-> excludes word letters

-> thus facilitates whole word search

# Regex

- Powerful tool
- Can solve ex. 1 in 3 lines:

```python
# Regex (optional)
import re
tokens=re.findall(r'https?://\S+|[\w.+%-]+@[\w.-]+\.\w{2,}|(?:Ph\.D\.|M\.Sc\.|Dr\.)|\d{2}/\d{2}/\d{2}|\$?\d+\.\d{2}\$?|#\w+|[.,:;!?()[\]/]|\w+',test_text
print(all([token == ref_token for token, ref_token in zip(tokens, ref)]))
```