HW_4_POS_Tagging_with_HuggingFace_for_student

January 30, 2025

1 HW 4 - POS Tagging with Hugging Face

In this exercise, you will create a part-of-speech (POS) tagging system for Thai text using NECTEC's ORCHID corpus. Instead of building your own deep learning architecture from scratch, you will leverage a pretrained tokenizer and a pretrained token classification model from Hugging Face.

We have provided some starter code for data cleaning and preprocessing in this notebook, but feel free to modify those parts to suit your needs. You are welcome to use additional libraries (e.g., scikit-learn) as long as you incorporate the pretrained Hugging Face model. Specifically, you will need to:

1. Load a pretrained tokenizer and token classification model.
2. Fine-tune it on the ORCHID corpus for POS tagging.
3. Evaluate and report the performance of your model on the test data.

1.0.1 Don't forget to change hardware accelrator to GPU in runtime on Google Colab

1.1 1. Setup and Preprocessing

[283]:
```
# Install transformers and thai2transformers
!pip install wandb
!pip install -q transformers==4.30.1 datasets evaluate thaixtransformers
!pip install -q emoji pythainlp sefr_cut tinydb seqeval sentencepiece pydantic
  ↪jsonlines
!pip install peft==0.10.0
```

```
Requirement already satisfied: wandb in /usr/local/lib/python3.10/dist-packages
(0.19.1)
Requirement already satisfied: click!=8.0.0,>=7.1 in
/usr/local/lib/python3.10/dist-packages (from wandb) (8.1.7)
Requirement already satisfied: docker-pycreds>=0.4.0 in
/usr/local/lib/python3.10/dist-packages (from wandb) (0.4.0)
Requirement already satisfied: gitpython!=3.1.29,>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from wandb) (3.1.43)
Requirement already satisfied: platformdirs in /usr/local/lib/python3.10/dist-
packages (from wandb) (4.3.6)
Requirement already satisfied: protobuf!=4.21.0,!=5.28.0,<6,>=3.19.0 in
/usr/local/lib/python3.10/dist-packages (from wandb) (3.20.3)
Requirement already satisfied: psutil>=5.0.0 in /usr/local/lib/python3.10/dist-
```

```
packages (from wandb) (5.9.5)
Requirement already satisfied: pydantic<3,>=2.6 in
/usr/local/lib/python3.10/dist-packages (from wandb) (2.10.3)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages
(from wandb) (6.0.2)
Requirement already satisfied: requests<3,>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from wandb) (2.32.3)
Requirement already satisfied: sentry-sdk>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from wandb) (2.19.2)
Requirement already satisfied: setproctitle in /usr/local/lib/python3.10/dist-
packages (from wandb) (1.3.4)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-
packages (from wandb) (75.1.0)
Requirement already satisfied: typing-extensions<5,>=4.4 in
/usr/local/lib/python3.10/dist-packages (from wandb) (4.12.2)
Requirement already satisfied: six>=1.4.0 in /usr/local/lib/python3.10/dist-
packages (from docker-pycreds>=0.4.0->wandb) (1.17.0)
Requirement already satisfied: gitdb<5,>=4.0.1 in
/usr/local/lib/python3.10/dist-packages (from gitpython!=3.1.29,>=1.0.0->wandb)
(4.0.11)
Requirement already satisfied: annotated-types>=0.6.0 in
/usr/local/lib/python3.10/dist-packages (from pydantic<3,>=2.6->wandb) (0.7.0)
Requirement already satisfied: pydantic-core==2.27.1 in
/usr/local/lib/python3.10/dist-packages (from pydantic<3,>=2.6->wandb) (2.27.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.0.0->wandb) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests<3,>=2.0.0->wandb) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.0.0->wandb) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.0.0->wandb)
(2024.12.14)
Requirement already satisfied: smmap<6,>=3.0.1 in
/usr/local/lib/python3.10/dist-packages (from
gitdb<5,>=4.0.1->gitpython!=3.1.29,>=1.0.0->wandb) (5.0.1)
Requirement already satisfied: peft==0.10.0 in /usr/local/lib/python3.10/dist-
packages (0.10.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-
packages (from peft==0.10.0) (1.26.4)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from peft==0.10.0) (24.2)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages
(from peft==0.10.0) (5.9.5)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages
(from peft==0.10.0) (6.0.2)
Requirement already satisfied: torch>=1.13.0 in /usr/local/lib/python3.10/dist-
packages (from peft==0.10.0) (2.5.1+cu121)
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-
packages (from peft==0.10.0) (4.30.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from peft==0.10.0) (4.67.1)
Requirement already satisfied: accelerate>=0.21.0 in
/usr/local/lib/python3.10/dist-packages (from peft==0.10.0) (1.2.1)
Requirement already satisfied: safetensors in /usr/local/lib/python3.10/dist-
packages (from peft==0.10.0) (0.4.5)
Requirement already satisfied: huggingface-hub>=0.17.0 in
/usr/local/lib/python3.10/dist-packages (from peft==0.10.0) (0.27.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from huggingface-hub>=0.17.0->peft==0.10.0) (3.16.1)
Requirement already satisfied: fsspec>=2023.5.0 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub>=0.17.0->peft==0.10.0) (2024.9.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from huggingface-hub>=0.17.0->peft==0.10.0) (2.32.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub>=0.17.0->peft==0.10.0) (4.12.2)
Requirement already satisfied: mkl_fft in /usr/local/lib/python3.10/dist-
packages (from numpy>=1.17->peft==0.10.0) (1.3.8)
Requirement already satisfied: mkl_random in /usr/local/lib/python3.10/dist-
packages (from numpy>=1.17->peft==0.10.0) (1.2.4)
Requirement already satisfied: mkl_umath in /usr/local/lib/python3.10/dist-
packages (from numpy>=1.17->peft==0.10.0) (0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.10/dist-packages
(from numpy>=1.17->peft==0.10.0) (2025.0.1)
Requirement already satisfied: tbb4py in /usr/local/lib/python3.10/dist-packages
(from numpy>=1.17->peft==0.10.0) (2022.0.0)
Requirement already satisfied: mkl-service in /usr/local/lib/python3.10/dist-
packages (from numpy>=1.17->peft==0.10.0) (2.4.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-
packages (from torch>=1.13.0->peft==0.10.0) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
(from torch>=1.13.0->peft==0.10.0) (3.1.4)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-
packages (from torch>=1.13.0->peft==0.10.0) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.10/dist-packages (from
sympy==1.13.1->torch>=1.13.0->peft==0.10.0) (1.3.0)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.10/dist-packages (from transformers->peft==0.10.0)
(2024.11.6)
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in
/usr/local/lib/python3.10/dist-packages (from transformers->peft==0.10.0)
(0.13.3)
Requirement already satisfied: MarkupSafe>=2.0 in
```

```
/usr/local/lib/python3.10/dist-packages (from
jinja2->torch>=1.13.0->peft==0.10.0) (3.0.2)
Requirement already satisfied: intel-openmp>=2024 in
/usr/local/lib/python3.10/dist-packages (from mkl->numpy>=1.17->peft==0.10.0)
(2024.2.0)
Requirement already satisfied: tbb==2022.* in /usr/local/lib/python3.10/dist-
packages (from mkl->numpy>=1.17->peft==0.10.0) (2022.0.0)
Requirement already satisfied: tcmlib==1.* in /usr/local/lib/python3.10/dist-
packages (from tbb==2022.*->mkl->numpy>=1.17->peft==0.10.0) (1.2.0)
Requirement already satisfied: intel-cmplr-lib-rt in
/usr/local/lib/python3.10/dist-packages (from
mkl_umath->numpy>=1.17->peft==0.10.0) (2024.2.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->huggingface-
hub>=0.17.0->peft==0.10.0) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests->huggingface-hub>=0.17.0->peft==0.10.0) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->huggingface-
hub>=0.17.0->peft==0.10.0) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->huggingface-
hub>=0.17.0->peft==0.10.0) (2024.12.14)
Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in
/usr/local/lib/python3.10/dist-packages (from intel-
openmp>=2024->mkl->numpy>=1.17->peft==0.10.0) (2024.2.0)
```

1.2 Setup

1. Register Wandb account (and confirm your email)
2. `wandb login` and copy paste the API key when prompt

[284]:
```
import wandb
# import os
# wandb.login(key=os.environ.get("WANDB_API_KEY"))
# wandb.login(key='')
```

We encourage you to login to your `Hugging Face` account so you can upload and share your model with the community. When prompted, enter your token to login

[285]:
```
# from huggingface_hub import notebook_login

# notebook_login()
```

Download the dataset from Hugging Face

[286]:
```
from datasets import load_dataset
```

```
orchid = load_dataset("Thichow/orchid_corpus", trust_remote_code=True)
```

[287]:
```
orchid
```

[287]:
```
DatasetDict({
    train: Dataset({
        features: ['id', 'label_tokens', 'pos_tags', 'sentence'],
        num_rows: 18500
    })
    test: Dataset({
        features: ['id', 'label_tokens', 'pos_tags', 'sentence'],
        num_rows: 4625
    })
})
```

[288]:
```
orchid['train'][0]
```

[288]:
```
{'id': '0',
 'label_tokens': ['  ', '   ', '  ', '    ', ' ', '  ', '  1'],
 'pos_tags': [21, 39, 26, 26, 37, 4, 18],
 'sentence': '              1'}
```

[289]:
```
orchid['train'][0]["sentence"]
```

[289]:
```
'              1'
```

[290]:
```
''.join(orchid['train'][0]['label_tokens'])
```

[290]:
```
'              1'
```

[291]:
```
label_list = orchid["train"].features[f"pos_tags"].feature.names
print('total type of pos_tags :', len(label_list))
print(label_list)
```

```
total type of pos_tags : 47
['ADVI', 'ADVN', 'ADVP', 'ADVS', 'CFQC', 'CLTV', 'CMTR', 'CMTR@PUNC', 'CNIT',
'CVBL', 'DCNM', 'DDAC', 'DDAN', 'DDAQ', 'DDBQ', 'DIAC', 'DIAQ', 'DIBQ', 'DONM',
'EAFF', 'EITT', 'FIXN', 'FIXV', 'JCMP', 'JCRG', 'JSBR', 'NCMN', 'NCNM', 'NEG',
'NLBL', 'NONM', 'NPRP', 'NTTL', 'PDMN', 'PNTR', 'PPRS', 'PREL', 'PUNC', 'RPRE',
'VACT', 'VATT', 'VSTA', 'XVAE', 'XVAM', 'XVBB', 'XVBM', 'XVMM']
```

[292]:
```
import numpy as np
import numpy.random
import torch

from tqdm.auto import tqdm
from functools import partial
```

```
#transformers
from transformers import (
    CamembertTokenizer,
    AutoTokenizer,
    AutoModel,
    AutoModelForMaskedLM,
    AutoModelForSequenceClassification,
    AutoModelForTokenClassification,
    TrainingArguments,
    Trainer,
    pipeline,
)

#thaixtransformers
from thaixtransformers import Tokenizer
from thaixtransformers.preprocess import process_transformers
```

Next, we load a pretrained tokenizer from Hugging Face. In this work, we utilize WangchanBERTa, a Thai-specific pretrained model, as the tokenizer.

2 Choose Pretrained Model

In this notebook, you can choose from 5 versions of WangchanBERTa, XLMR and mBERT to perform downstream tasks on Thai datasets. The datasets are:

- `wangchanberta-base-att-spm-uncased` (recommended) - Largest WangchanBERTa trained on 78.5GB of Assorted Thai Texts with subword tokenizer SentencePiece
- `xlm-roberta-base` - Facebook's XLMR trained on 100 languages
- `bert-base-multilingual-cased` - Google's mBERT trained on 104 languages
- `wangchanberta-base-wiki-newmm` - WangchanBERTa trained on Thai Wikipedia Dump with PyThaiNLP's word-level tokenizer `newmm`
- `wangchanberta-base-wiki-syllable` - WangchanBERTa trained on Thai Wikipedia Dump with PyThaiNLP's syllabel-level tokenizer `syllable`
- `wangchanberta-base-wiki-sefr` - WangchanBERTa trained on Thai Wikipedia Dump with word-level tokenizer `SEFR`
- `wangchanberta-base-wiki-spm` - WangchanBERTa trained on Thai Wikipedia Dump with subword-level tokenizer SentencePiece

In the first part, we require you to select the wangchanberta-base-att-spm-uncased.

Learn more about using wangchanberta at wangchanberta_getting_started_ai_reseach

- You need to set the transformers version to transformers==4.30.1.

`In the first part, we require you to select the wangchanberta-base-att-spm-uncased.`

[293]:
```
model_names = [
    'airesearch/wangchanberta-base-att-spm-uncased',
    'airesearch/wangchanberta-base-wiki-newmm',
    'airesearch/wangchanberta-base-wiki-ssg',
```

```
    'airesearch/wangchanberta-base-wiki-sefr',
    'airesearch/wangchanberta-base-wiki-spm',
]

#@title Choose Pretrained Model
model_name = "airesearch/wangchanberta-base-att-spm-uncased"

#create tokenizer
tokenizer = Tokenizer(model_name).from_pretrained(
                f'{model_name}',
                revision='main',
                model_max_length=416,)
```

```
The tokenizer class you load from this checkpoint is not the same type as the
class this function is called from. It may result in unexpected tokenization.
The tokenizer class you load from this checkpoint is 'CamembertTokenizer'.
The class this function is called from is 'WangchanbertaTokenizer'.
The tokenizer class you load from this checkpoint is not the same type as the
class this function is called from. It may result in unexpected tokenization.
The tokenizer class you load from this checkpoint is 'CamembertTokenizer'.
The class this function is called from is 'WangchanbertaTokenizer'.
```

Let's try using a pretrained tokenizer.

[294]:
```
text = '                  '
print('text :', text)
tokens = []
for i in tokenizer([text], is_split_into_words=True)['input_ids']:
  tokens.append(tokenizer.decode(i))
print('tokens :', tokens)
```

```
text :
tokens : ['<s>', '', '   ', '   ', '    ', '  ', '<_>', '  ',
'   ', '  ', '  ', '</s>']
```

model : * `wangchanberta-base-att-spm-uncased`

First, we print examples of label tokens from our dataset for inspection.

[295]:
```
example = orchid["train"][0]
for i in example :
    print(i, ':', example[i])
```

```
id : 0
label_tokens : ['  ', '   ', '  ', '    ', ' ', '  ', '  1']
pos_tags : [21, 39, 26, 26, 37, 4, 18]
sentence :               1
```

Then, we use the sentence '   1' to be tokenized by the pretrained tokenizer model.

[296]:
```
text = '                1'
tokenizer(text)
```

[296]:
```
{'input_ids': [5, 10, 882, 8222, 8, 10, 1014, 8, 10, 59, 6], 'attention_mask':
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

These are already mapped into discrete values. We can uncover the original token text from the tokens by.

[297]:
```
for i in tokenizer(text)['input_ids']:
  print(tokenizer.convert_ids_to_tokens(i))
```

```
<s>




<_>


<_>

1
</s>
```

Now let's look at another example.

[298]:
```
example = orchid["train"][1899]
print('sentence :', example["sentence"])
tokenized_input = tokenizer([example["sentence"]], is_split_into_words=True)
tokens = tokenizer.convert_ids_to_tokens(tokenized_input["input_ids"])
print('tokens :',tokens)
print('label tokens :', example["label_tokens"])
print('label pos :', example["pos_tags"])
```

```
sentence :                (Bilingual transfer dictionary)
tokens : ['<s>', '  ', '     ', '    ', '  ', ' ', '<_>', ' (',
'<unk>', 'i', 'ling', 'ual', '<_>', ' ', 'trans', 'fer', '<_>', ' ', 'di',
'ction', 'ary', ')', '</s>']
label tokens : ['  ', '    ', '  ', '     ', '  ', '', ' ', '(',
'Bilingual transfer dictionary', ')']
label pos : [25, 39, 38, 26, 26, 5, 37, 37, 26, 37]
```

Notice how `B` becomes an `<unk>` token. This is because this is an uncased model, meaning it only handles small English characters.

3 #TODO 0

Convert the dataset to lowercase.

```
[299]: # Create a lowercase dataset for uncased BERT
def lower_case_sentences(examples: dict):
    lower_cased_examples = examples
    # TODO: fill code here to lower case the "sentence" and "label_tokens"
    sentence = lower_cased_examples["sentence"].lower()
    label_tokens = [token.lower() for token in␣
 ↪lower_cased_examples["label_tokens"]]
    lower_cased_examples["sentence"] = sentence
    lower_cased_examples["label_tokens"] = label_tokens
    return lower_cased_examples
```

```
[300]: orchidl = orchid.map(lower_case_sentences)
```

```
[301]: orchidl
```

```
[301]: DatasetDict({
    train: Dataset({
        features: ['id', 'label_tokens', 'pos_tags', 'sentence'],
        num_rows: 18500
    })
    test: Dataset({
        features: ['id', 'label_tokens', 'pos_tags', 'sentence'],
        num_rows: 4625
    })
})
```

```
[302]: orchidl["train"][1899]
```

```
[302]: {'id': '1899',
 'label_tokens': ['  ',
  '    ',
  '  ',
  '     ',
  '   ',
  ' ',
  ' ',
  '(',
  'bilingual transfer dictionary',
  ')'],
 'pos_tags': [25, 39, 38, 26, 26, 5, 37, 37, 26, 37],
 'sentence': '              (bilingual transfer dictionary)'}
```

Now let's examine the labels again.

```
[303]: example = orchidl["train"][1899]
print('sentence :', example["sentence"])
tokenized_input = tokenizer([example["sentence"]], is_split_into_words=True)
tokens = tokenizer.convert_ids_to_tokens(tokenized_input["input_ids"])
```

```
print('tokens :',tokens)
print('label tokens :', example["label_tokens"])
print('label pos :', example["pos_tags"])
```

```
sentence :                (bilingual transfer dictionary)
tokens : ['<s>', '  ', '     ', '     ', '  ', ' ', '<_>', ' (',
'bi', 'ling', 'ual', '<_>', ' ', 'trans', 'fer', '<_>', ' ', 'di', 'ction',
'ary', ')', '</s>']
label tokens : ['  ', '   ', '  ', '    ', '  ', '', ' ', '(',
'bilingual transfer dictionary', ')']
label pos : [25, 39, 38, 26, 26, 5, 37, 37, 26, 37]
```

[304]:
```
example = orchidl["train"][0]
print('sentence :', example["sentence"])
tokenized_input = tokenizer([example["sentence"]], is_split_into_words=True)
tokens = tokenizer.convert_ids_to_tokens(tokenized_input["input_ids"])
print('tokens :',tokens)
print('label tokens :', example["label_tokens"])
print('label pos :', example["pos_tags"])
```

```
sentence :               1
tokens : ['<s>', ' ', '    ', '     ', '<_>', ' ', '  ', '<_>',
' ', '1', '</s>']
label tokens : ['  ', '   ', '  ', '    ', ' ', '  ', '  1']
label pos : [21, 39, 26, 26, 37, 4, 18]
```

In the example above, tokens refer to those tokenized using the pretrained tokenizer, while label tokens refer to tokens tokenized from our dataset.

Do you see something?

Yes, the tokens from the two tokenizers do not match.

- sentence : 1
- tokens : ['<s>', ' ', '   ', '   ', '<_>', ' ', ' ', '<_>', ' ', '1', '</s>']
- label tokens : [' ', '   ', ' ', '   ', ' ', ' ', ' 1']
- label pos : [21, 39, 26, 26, 37, 4, 18]

You can see that in our label tokens, ‘ ’ has a POS tag of 21, and ‘ ’ has a POS tag of 39. However, when we tokenize the sentence using WangchanBERTa, we get the token ‘ ’. What POS tag should we assign to this new token?

What should we do ?

Based on this example, we found that the tokens from the WangchanBERTa do not directly align with our label tokens. This means we cannot directly use the label POS tags. Therefore, we need to reassign POS tags to the tokens produced by WangchanBERTa tokenization. The method we will use is majority voting: - If a token from the WangchanBERTa matches a label token exactly, we will directly assign the POS tag from the label POS. - If the token generated overlaps or combines multiple label tokens, we assign the POS tag based on the number of characters in each token: If the token contains the most characters from any label token, we assign the POS tag from that label token.

Example :

```
# "     " (9 chars) is formed from "  " (3 chars) + "   " (6 chars).
# "  " has a POS tag of 21,
# and "   " has a POS tag of 39.
# Therefore, the POS tag for "     " is 39,
# as "     " is derived more from the "   " part than from the "  " part.

# '     ' (10 chars) is formed from '  ' (3 chars) + '    ' (7 chars)
# "  " has a POS tag of 26,
# and "    " has a POS tag of 2.
# Therefore, the POS tag for "      " is 2,
# as "      " is derived more from the "  " part than from the "    " part.
```

4 #TODO 1

```
**Warning: Please be careful of <unk>, an unknown word token.**
```

```
**Warning: Please be careful of "   ", the 'am' vowel. WangchanBERTa's internal
preprocessing replaces all "   " to '' and ' '**
```

Assigning the label -100 to the special tokens `[<s>]` and `[</s>]` and `[_]` so they're ignored by the PyTorch loss function (see CrossEntropyLoss: ignore_index)

[305]:
```python
# example.keys(), example.values()
```

[306]:
```python
def majority_vote_pos(examples: dict):
    """
    Args:
        examples (dict):
            id: int
            label_tokens: list of str
            pos_tags: list of int
            sentence: str
            input_ids: list of int
            attention_mask: list of int
            tokens: list of str
            labels: list of int
    """
  ␣
↪##########################################################################################
    # TO DO: Since the tokens from the output of the pretrained tokenizer
    # do not match the tokens in the label tokens of the dataset,
```

```
    # the task is to create a function to determine the POS tags of the tokens generated by the pretrained tokenizer.
    # This should be done by referencing the POS tags in the label tokens. If a token partially overlaps with others,
    # the POS tag from the segment with the greater number of characters should be assigned.
    #
    # Example :
    # "     " (9 chars) is formed from "  " (3 chars) + "   " (6 chars).
    # "  " has a POS tag of 21,
    # and "   " has a POS tag of 39.
    # Therefore, the POS tag for "     " is 39,
    # as "     " is derived more from the "   " part than from the "  " part.
    #
    # '      ' (10 chars) is formed from '  ' (3 chars) + '    ' (7 chars)
    # "  " has a POS tag of 26,
    # and "    " has a POS tag of 2.
    # Therefore, the POS tag for "      " is 2,
    # as "      " is derived more from the "  " part than from the "    " part.

    # tokenize word by pretrained tokenizer
    tokenized_inputs = tokenizer([examples["sentence"]], is_split_into_words=True)
    # print(tokenized_inputs) # input_ids (list of int), attention_mask (list of int)

    # TODO: FILL CODE HERE

    new_tokens = tokenizer.convert_ids_to_tokens(tokenized_inputs["input_ids"])
    label_tokens = examples["label_tokens"]
    pos_tags = examples["pos_tags"]

    new_pos_result = []

    label_idx = 0  # Index for label_tokens
    char_idx = 0  # Pointer for matching characters within label tokens

    for token in new_tokens:
        # PyTorch special tokens
        if token in ["<s>", "</s>", "_"]:
            new_pos_result.append(-100)
            continue

        # Handle space characters and Thai character normalization
        token = token.replace(' ', " ")
        token = token.replace("<_>", " ")
        if token.startswith(" "):
```

```
            token = token[1:]

        # Ensure token is not empty
        if not token:
            new_pos_result.append(-100)
            continue

        # Move label pointer to correct position
        while label_idx < len(label_tokens) and char_idx < len(label_tokens[label_idx]):
            if label_tokens[label_idx][char_idx] == token[0]:
                break
            # Move to next label token
            char_idx += 1
            if char_idx >= len(label_tokens[label_idx]):
                label_idx += 1
                char_idx = 0

        # Align token with label tokens
        accumulated_text = ""
        pos_weight = {}
        # Accumulate characters and determine correct POS tag
        while accumulated_text != token and label_idx < len(label_tokens):
            accumulated_text += label_tokens[label_idx][char_idx]
            pos_tag = pos_tags[label_idx]

            if pos_tag not in pos_weight:
                pos_weight[pos_tag] = 0
            pos_weight[pos_tag] += 1

            # Move idx
            char_idx += 1
            if char_idx >= len(label_tokens[label_idx]):
                label_idx += 1
                char_idx = 0

        # Assign POS tag based on majority voting
        best_pos_tag = max(pos_weight, key=pos_weight.get) if pos_weight else -100
        new_pos_result.append(best_pos_tag)

    # Update example dictionary with new tokens and labels
    tokenized_inputs['tokens'] = new_tokens
    tokenized_inputs['labels'] = new_pos_result

    return tokenized_inputs
```

[307]:
```python
tokenized_orchid = orchidl.map(majority_vote_pos)
```

[308]:
```python
tokenized_orchid
```

[308]:
```
DatasetDict({
    train: Dataset({
        features: ['id', 'label_tokens', 'pos_tags', 'sentence', 'input_ids',
'attention_mask', 'tokens', 'labels'],
        num_rows: 18500
    })
    test: Dataset({
        features: ['id', 'label_tokens', 'pos_tags', 'sentence', 'input_ids',
'attention_mask', 'tokens', 'labels'],
        num_rows: 4625
    })
})
```

[309]:
```python
tokenized_orchid['train'][0]
```

[309]:
```
{'id': '0',
 'label_tokens': ['  ', '   ', ' ', '   ', ' ', ' ', ' 1'],
 'pos_tags': [21, 39, 26, 26, 37, 4, 18],
 'sentence': '              1',
 'input_ids': [5, 10, 882, 8222, 8, 10, 1014, 8, 10, 59, 6],
 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 'tokens': ['<s>',
  ' ',
  '    ',
  '     ',
  '<_>',
  ' ',
  '  ',
  '<_>',
  ' ',
  '1',
  '</s>'],
 'labels': [-100, -100, 39, 26, 37, -100, 4, 18, -100, 18, -100]}
```

[310]:
```python
example = tokenized_orchid["train"][0]
for i in example :
    print(i, ":", example[i])
```

```
id : 0
label_tokens : ['  ', '   ', ' ', '   ', ' ', ' ', ' 1']
pos_tags : [21, 39, 26, 26, 37, 4, 18]
sentence :              1
input_ids : [5, 10, 882, 8222, 8, 10, 1014, 8, 10, 59, 6]
attention_mask : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

```
tokens : ['<s>', ' ', '      ', '       ', '<_>', ' ', '   ', '<_>',
' ', '1', '</s>']
labels : [-100, -100, 39, 26, 37, -100, 4, 18, -100, 18, -100]
```

This is the result after we realigned the POS based on the majority vote. - label_tokens : ['   ', '   ', '   ', '   ', ' ', '   ', ' 1'] - pos_tags : [21, 39, 26, 26, 37, 4, 18] - tokens : ['<s>', ' ', '   ', '   ', '<_>', ' ', '   ', '<_>', ' ', '1', '</s>'] - labels : [-100, -100, 39, 26, 37, -100, 4, 18, -100, 18, -100]

```
['<s>', ' ', '</s>'] : -100
```

Check :

"   " (9 chars) is formed from "   " (3 chars) + "   " (6 chars).

"   " has a POS tag of 21,

and "   " has a POS tag of 39.

Therefore, the POS tag for "   " is 39,

as "   " is derived more from the "   " part than from the "   " part.

[311]:
```python
# hard test case
example = tokenized_orchid["train"][1899]
for i in example :
    print(i, ":", example[i])

# test
print(f"id: {example['id']=='1899'}")
print(f"label_tokens: {example['label_tokens']==['   ', '     ', '   ', '      ',
 ↪'   ', ' ', ' ', '(', 'bilingual transfer dictionary', ')']}")
print(f"pos_tags: {example['pos_tags']==[25, 39, 38, 26, 26, 5, 37, 37, 26,
 ↪37]}")
print(f"sentence: {example['sentence']=='                (bilingual transfer
 ↪dictionary)'}")
print(f"input_ids: {example['input_ids']==[5, 489, 15617, 19737, 958, 493, 8,
 ↪1241, 4906, 11608, 12177, 8, 10, 11392, 9806, 8, 10, 2951, 15779, 8001, 29,
 ↪6]}")
print(f"attention_mask: {example['attention_mask']==[1, 1, 1, 1, 1, 1, 1, 1, 1,
 ↪1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}")
print(f"tokens: {example['tokens']==['<s>', '   ', '      ', '      ', '   ', ' ',
 ↪'<_>', ' (', 'bi', 'ling', 'ual', '<_>', ' ', 'trans', 'fer', '<_>', ' ',
 ↪'di', 'ction', 'ary', ')', '</s>']}")
print(f"labels: {example['labels']==[-100, 25, 39, 26, 26, 5, 37, 37, 26, 26,
 ↪26, 26, -100, 26, 26, 26, -100, 26, 26, 26, 37, -100]}")
```

```
id : 1899
label_tokens : ['   ', '    ', '   ', '      ', '   ', '', ' ', '(',
'bilingual transfer dictionary', ')']
pos_tags : [25, 39, 38, 26, 26, 5, 37, 37, 26, 37]
sentence :                (bilingual transfer dictionary)
```

```
input_ids : [5, 489, 15617, 19737, 958, 493, 8, 1241, 4906, 11608, 12177, 8, 10,
11392, 9806, 8, 10, 2951, 15779, 8001, 29, 6]
attention_mask : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1]
tokens : ['<s>', '  ', '     ', '    ', '  ', ' ', '<_>', ' (',
'bi', 'ling', 'ual', '<_>', ' ', 'trans', 'fer', '<_>', ' ', 'di', 'ction',
'ary', ')', '</s>']
labels : [-100, 25, 39, 26, 26, 5, 37, 37, 26, 26, 26, 26, -100, 26, 26, 26,
-100, 26, 26, 26, 37, -100]
id: True
label_tokens: True
pos_tags: True
sentence: True
input_ids: True
attention_mask: True
tokens: True
labels: True
```

Expected output

```
id : 1899
label_tokens : ['  ', '    ', '  ', '     ', '  ', ' ', ' ', '(', 'bilingual transfer dictionary'
pos_tags : [25, 39, 38, 26, 26, 5, 37, 37, 26, 37]
sentence :                 (bilingual transfer dictionary)
input_ids : [5, 489, 15617, 19737, 958, 493, 8, 1241, 4906, 11608, 12177, 8, 10, 11392, 9806, 8
attention_mask : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
tokens : ['<s>', '  ', '     ', '    ', '  ', ' ', '<_>', ' (', 'bi', 'ling', 'ual', '<_>', ' '
labels : [-100, 25, 39, 26, 26, 5, 37, 37, 26, 26, 26, 26, -100, 26, 26, 26, -100, 26, 26, 26,
```

5 Train and Evaluate model

We will create a batch of examples using DataCollatorWithPadding.

Data collators are objects that will form a batch by using a list of dataset elements as input. These elements are of the same type as the elements of train_dataset or eval_dataset.

DataCollatorWithPadding will help us pad the sentences to the longest length in a batch during collation, instead of padding the whole dataset to the maximum length. This allows for efficient computation during each batch.

- DataCollatorForTokenClassification : `padding (bool, str or PaddingStrategy, optional, defaults to True)`
- `True` or `'longest'` (default): Pad to the longest sequence in the batch (or no padding if only a single sequence is provided).

[312]:
```
from transformers import DataCollatorForTokenClassification

data_collator = DataCollatorForTokenClassification(tokenizer=tokenizer)
```

For evaluating your model's performance. You can quickly load a evaluation method with the

Evaluate library. For this task, load the seqeval framework (see the Evaluate quick tour to learn more about how to load and compute a metric). Seqeval actually produces several scores: precision, recall, F1, and accuracy.

[313]:
```
import evaluate

seqeval = evaluate.load("seqeval")
```

Huggingface requires us to write a `compute_metrics()` function. This will be invoked when huggingface evalutes a model.

Note that we ignore to evaluate on -100 labels.

[314]:
```
import numpy as np
import warnings


def compute_metrics(p):
    predictions, labels = p
    predictions = np.argmax(predictions, axis=2)

    true_predictions = [
        [label_list[p] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]
    true_labels = [
        [label_list[l] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]

    with warnings.catch_warnings():
        warnings.filterwarnings("ignore")
        results = seqeval.compute(predictions=true_predictions,␣
↪references=true_labels)
    return {
        "precision": results["overall_precision"],
        "recall": results["overall_recall"],
        "f1": results["overall_f1"],
        "accuracy": results["overall_accuracy"],
    }
```

The total number of labels in our POS tag set.

[315]:
```
id2label = {
    0: 'ADVI',
    1: 'ADVN',
    2: 'ADVP',
    3: 'ADVS',
    4: 'CFQC',
```

```
    5: 'CLTV',
    6: 'CMTR',
    7: 'CMTR@PUNC',
    8: 'CNIT',
    9: 'CVBL',
    10: 'DCNM',
    11: 'DDAC',
    12: 'DDAN',
    13: 'DDAQ',
    14: 'DDBQ',
    15: 'DIAC',
    16: 'DIAQ',
    17: 'DIBQ',
    18: 'DONM',
    19: 'EAFF',
    20: 'EITT',
    21: 'FIXN',
    22: 'FIXV',
    23: 'JCMP',
    24: 'JCRG',
    25: 'JSBR',
    26: 'NCMN',
    27: 'NCNM',
    28: 'NEG',
    29: 'NLBL',
    30: 'NONM',
    31: 'NPRP',
    32: 'NTTL',
    33: 'PDMN',
    34: 'PNTR',
    35: 'PPRS',
    36: 'PREL',
    37: 'PUNC',
    38: 'RPRE',
    39: 'VACT',
    40: 'VATT',
    41: 'VSTA',
    42: 'XVAE',
    43: 'XVAM',
    44: 'XVBB',
    45: 'XVBM',
    46: 'XVMM',
    # 47: 'O'
}
label2id = {}
for k, v in id2label.items() :
    label2id[v] = k
```

```
label2id
```

[315]:
```
{'ADVI': 0,
 'ADVN': 1,
 'ADVP': 2,
 'ADVS': 3,
 'CFQC': 4,
 'CLTV': 5,
 'CMTR': 6,
 'CMTR@PUNC': 7,
 'CNIT': 8,
 'CVBL': 9,
 'DCNM': 10,
 'DDAC': 11,
 'DDAN': 12,
 'DDAQ': 13,
 'DDBQ': 14,
 'DIAC': 15,
 'DIAQ': 16,
 'DIBQ': 17,
 'DONM': 18,
 'EAFF': 19,
 'EITT': 20,
 'FIXN': 21,
 'FIXV': 22,
 'JCMP': 23,
 'JCRG': 24,
 'JSBR': 25,
 'NCMN': 26,
 'NCNM': 27,
 'NEG': 28,
 'NLBL': 29,
 'NONM': 30,
 'NPRP': 31,
 'NTTL': 32,
 'PDMN': 33,
 'PNTR': 34,
 'PPRS': 35,
 'PREL': 36,
 'PUNC': 37,
 'RPRE': 38,
 'VACT': 39,
 'VATT': 40,
 'VSTA': 41,
 'XVAE': 42,
 'XVAM': 43,
```

```
 'XVBB': 44,
 'XVBM': 45,
 'XVMM': 46}
```

[316]:
```
labels = [i for i in id2label.values()]
labels
```

[316]:
```
['ADVI',
 'ADVN',
 'ADVP',
 'ADVS',
 'CFQC',
 'CLTV',
 'CMTR',
 'CMTR@PUNC',
 'CNIT',
 'CVBL',
 'DCNM',
 'DDAC',
 'DDAN',
 'DDAQ',
 'DDBQ',
 'DIAC',
 'DIAQ',
 'DIBQ',
 'DONM',
 'EAFF',
 'EITT',
 'FIXN',
 'FIXV',
 'JCMP',
 'JCRG',
 'JSBR',
 'NCMN',
 'NCNM',
 'NEG',
 'NLBL',
 'NONM',
 'NPRP',
 'NTTL',
 'PDMN',
 'PNTR',
 'PPRS',
 'PREL',
 'PUNC',
 'RPRE',
 'VACT',
```

```
'VATT',
'VSTA',
'XVAE',
'XVAM',
'XVBB',
'XVBM',
'XVMM']
```

5.1 Load pretrained model

Select a pretrained model for fine-tuning to develop a POS Tagger model using the Orchid corpus dataset.

- model : `wangchanberta-base-att-spm-uncased`
- Don't forget to update the num_labels.

You're ready to start training your model now! Load pretrained model with AutoModelForTokenClassification along with the number of expected labels, and the label mappings:

`In the first part, we require you to select the wangchanberta-base-att-spm-uncased.`

[317]:
```python
model_names = [
    'wangchanberta-base-att-spm-uncased',
    'wangchanberta-base-wiki-newmm',
    'wangchanberta-base-wiki-ssg',
    'wangchanberta-base-wiki-sefr',
    'wangchanberta-base-wiki-spm',
]

#@title Choose Pretrained Model
model_name = "wangchanberta-base-att-spm-uncased"

#create model
model = AutoModelForTokenClassification.from_pretrained(
    f"airesearch/{model_name}",
    revision='main',
    num_labels=47, id2label=id2label, label2id=label2id
)
```

```
Some weights of the model checkpoint at airesearch/wangchanberta-base-att-spm-
uncased were not used when initializing CamembertForTokenClassification:
['lm_head.dense.weight', 'lm_head.layer_norm.bias', 'lm_head.layer_norm.weight',
'lm_head.dense.bias', 'lm_head.bias']
- This IS expected if you are initializing CamembertForTokenClassification from
the checkpoint of a model trained on another task or with another architecture
(e.g. initializing a BertForSequenceClassification model from a
BertForPreTraining model).
- This IS NOT expected if you are initializing CamembertForTokenClassification
from the checkpoint of a model that you expect to be exactly identical
(initializing a BertForSequenceClassification model from a
```

```
BertForSequenceClassification model).
Some weights of CamembertForTokenClassification were not initialized from the
model checkpoint at airesearch/wangchanberta-base-att-spm-uncased and are newly
initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

5.1.1 #TODO 2

- Configure your training hyperparameters using `**TrainingArguments**`. The only required parameter is is `output_dir`, which determines the directory where your model will be saved. To upload the model to the Hugging Face Hub, set push_to_hub=True (note: you must be logged into Hugging Face for this). During training, the Trainer will compute seqeval metrics at the end of each epoch and store the training checkpoint.
- Provide the `**Trainer**` with the training arguments, as well as the model, dataset, tokenizer, data collator, and compute_metrics function.
- Use `**train()**` to fine-tune the model.

Read huggingface's tutorial for more details.

[318]:
```
# from transformers import TrainingArguments

# training_args = TrainingArguments(
#     output_dir="./models",
#     evaluation_strategy="epoch",
#     save_strategy="epoch",
#     learning_rate=2e-5,
#     per_device_train_batch_size=16,
#     per_device_eval_batch_size=16,
#     num_train_epochs=3,
#     weight_decay=0.01,
#     logging_dir="./logs",
#     logging_steps=10,
#     report_to="none",
#     save_total_limit=2,
#     fp16=True,
#     push_to_hub=True,  # <-- This must be True
#     hub_model_id="pupipatsk/pos-wangchanberta-base-att-spm-uncased",  #␣
 ↪Specify model name
#     hub_strategy="every_save",  # Upload model after each save
# )

# from transformers import Trainer

# trainer = Trainer(
#     model=model,
#     args=training_args,
#     train_dataset=tokenized_orchid["train"],
```

```
#     eval_dataset=tokenized_orchid["test"],
#     tokenizer=tokenizer,
#     data_collator=data_collator,
#     compute_metrics=compute_metrics
# )

# trainer.train()
```

6 Inference

With your model fine-tuned, you can now perform inference.

[319]:
```
text = "             1"
```

In the first part, we require you to select the wangchanberta-base-att-spm-uncased.

[320]:
```
from transformers import AutoTokenizer

# Load pretrained tokenizer from Hugging Face
#@title Choose Pretrained Model
model_name = "airesearch/wangchanberta-base-att-spm-uncased"

tokenizer = Tokenizer(model_name).from_pretrained(model_name)
inputs = tokenizer(text, return_tensors="pt")
```

```
The tokenizer class you load from this checkpoint is not the same type as the
class this function is called from. It may result in unexpected tokenization.
The tokenizer class you load from this checkpoint is 'CamembertTokenizer'.
The class this function is called from is 'WangchanbertaTokenizer'.
The tokenizer class you load from this checkpoint is not the same type as the
class this function is called from. It may result in unexpected tokenization.
The tokenizer class you load from this checkpoint is 'CamembertTokenizer'.
The class this function is called from is 'WangchanbertaTokenizer'.
```

[321]:
```
inputs
```

[321]:
```
{'input_ids': tensor([[   5,   10,  882, 8222,    8,   10, 1014,    8,   10,
59,    6]]), 'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])}
```

[322]:
```
from transformers import AutoModelForTokenClassification

## Load your fine-tuned model from Hugging Face
model = AutoModelForTokenClassification.from_pretrained("pupipatsk/
 ↪pos-wangchanberta-base-att-spm-uncased") ## your model path from Hugging Face
with torch.no_grad():
    logits = model(**inputs).logits
```

```
/usr/local/lib/python3.10/dist-packages/transformers/modeling_utils.py:463:
FutureWarning: You are using `torch.load` with `weights_only=False` (the current
```

```
default value), which uses the default pickle module implicitly. It is possible
to construct malicious pickle data which will execute arbitrary code during
unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for
more details). In a future release, the default value for `weights_only` will be
flipped to `True`. This limits the functions that could be executed during
unpickling. Arbitrary objects will no longer be allowed to be loaded via this
mode unless they are explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control of the
loaded file. Please open an issue on GitHub for any issues related to this
experimental feature.
  return torch.load(checkpoint_file, map_location="cpu")
```

[323]:
```
predictions = torch.argmax(logits, dim=2)
predicted_token_class = [model.config.id2label[t.item()] for t in␣
 ↪predictions[0]]
predicted_token_class
```

[323]:
```
['PUNC',
 'PUNC',
 'VACT',
 'NCMN',
 'PUNC',
 'PUNC',
 'CFQC',
 'DONM',
 'DONM',
 'DONM',
 'PUNC']
```

[324]:
```
id2label
```

[324]:
```
{0: 'ADVI',
 1: 'ADVN',
 2: 'ADVP',
 3: 'ADVS',
 4: 'CFQC',
 5: 'CLTV',
 6: 'CMTR',
 7: 'CMTR@PUNC',
 8: 'CNIT',
 9: 'CVBL',
 10: 'DCNM',
 11: 'DDAC',
 12: 'DDAN',
 13: 'DDAQ',
 14: 'DDBQ',
```

```
 15: 'DIAC',
 16: 'DIAQ',
 17: 'DIBQ',
 18: 'DONM',
 19: 'EAFF',
 20: 'EITT',
 21: 'FIXN',
 22: 'FIXV',
 23: 'JCMP',
 24: 'JCRG',
 25: 'JSBR',
 26: 'NCMN',
 27: 'NCNM',
 28: 'NEG',
 29: 'NLBL',
 30: 'NONM',
 31: 'NPRP',
 32: 'NTTL',
 33: 'PDMN',
 34: 'PNTR',
 35: 'PPRS',
 36: 'PREL',
 37: 'PUNC',
 38: 'RPRE',
 39: 'VACT',
 40: 'VATT',
 41: 'VSTA',
 42: 'XVAE',
 43: 'XVAM',
 44: 'XVBB',
 45: 'XVBM',
 46: 'XVMM'}
```

[325]:
```
# Inference
# ignore special tokens
text = '                  '
inputs = tokenizer(text, return_tensors="pt")
tokenized_input = tokenizer([text], is_split_into_words=True)
tokens = tokenizer.convert_ids_to_tokens(tokenized_input["input_ids"])
print('tokens :', tokens)
with torch.no_grad():
    logits = model(**inputs).logits
predictions = torch.argmax(logits, dim=2)
predicted_token_class = [model.config.id2label[t.item()] for t in␣
 ↪predictions[0]]
print('predict pos :', predicted_token_class)
```

```
tokens : ['<s>', ' ', '    ', '  ', '  ', '   ', '  ', '  ',
```

```
'  ', '    ', '</s>']
predict pos : ['PUNC', 'PUNC', 'JSBR', 'ADVS', 'NCMN', 'NCMN', 'RPRE', 'PPRS',
'VATT', 'ADVN', 'PUNC']
```

Evaluate model :

The output from the model is a softmax over classes. We choose the maximum class as the answer for evaluation. Again, we will ignore the -100 labels.

[326]:
```python
import pandas as pd
from IPython.display import display

def evaluation_report(y_true, y_pred, get_only_acc=False):
    # retrieve all tags in y_true
    tag_set = set()
    for sent in y_true:
        for tag in sent:
            tag_set.add(tag)
    for sent in y_pred:
        for tag in sent:
            tag_set.add(tag)
    tag_list = sorted(list(tag_set))

    # count correct points
    tag_info = dict()
    for tag in tag_list:
        tag_info[tag] = {'correct_tagged': 0, 'y_true': 0, 'y_pred': 0}

    all_correct = 0
    all_count = sum([len(sent) for sent in y_true])
    speacial_tag = 0
    for sent_true, sent_pred in zip(y_true, y_pred):
        for tag_true, tag_pred in zip(sent_true, sent_pred):
            # pass special token
            if tag_true == -100 :
                speacial_tag += 1
                pass
            if tag_true == tag_pred:
                tag_info[tag_true]['correct_tagged'] += 1
                all_correct += 1
            tag_info[tag_true]['y_true'] += 1
            tag_info[tag_pred]['y_pred'] += 1
    print('speacial_tag :',speacial_tag) # delete number of special token from all_count
    accuracy = (all_correct / (all_count-speacial_tag))

    # get only accuracy for testing
    if get_only_acc:
```

```
        return accuracy

    accuracy *= 100


    # summarize and make evaluation result
    eval_list = list()
    for tag in tag_list:
        eval_result = dict()
        eval_result['tag'] = tag
        eval_result['correct_count'] = tag_info[tag]['correct_tagged']
        precision = (tag_info[tag]['correct_tagged']/
↪tag_info[tag]['y_pred'])*100 if tag_info[tag]['y_pred'] else '-'
        recall = (tag_info[tag]['correct_tagged']/tag_info[tag]['y_true'])*100␣
↪if (tag_info[tag]['y_true'] > 0) else 0
        eval_result['precision'] = precision
        eval_result['recall'] = recall
        eval_result['f1_score'] = (2*precision*recall)/(precision+recall) if␣
↪(type(precision) is float and recall > 0) else '-'

        eval_list.append(eval_result)

    eval_list.append({'tag': 'accuracy=%.2f' % accuracy, 'correct_count': '',␣
↪'precision': '', 'recall': '', 'f1_score': ''})

    df = pd.DataFrame.from_dict(eval_list)
    df = df[['tag', 'precision', 'recall', 'f1_score', 'correct_count']]

    display(df)
```

[327]:
```
# prepare test set
test_data = tokenized_orchid["test"]
```

[328]:
```
# labels for test set
y_test = []
for inputs in test_data:
  y_test.append(inputs['labels'])
```

[329]:
```
# y_pred = []
# device = 'cuda' if torch.cuda.is_available() else 'cpu'
# for inputs in tqdm(test_data):
#     text = inputs['sentence']
#     inputs = tokenizer(text, return_tensors="pt")
#     with torch.no_grad():
#         pred = model(**inputs).logits
#         predictions = torch.argmax(pred, dim=2)
#         # Append padded predictions to y_pred
```

```
#         y_pred.append(predictions.tolist()[0])

# # save y_pred to local
# import json
# with open('y_pred-pos-wangchanberta-base-att-spm-uncased.json', 'w') as f:
#     json.dump(y_pred, f)
```

[330]:
```
# load y_pred from local
with open('y_pred-pos-wangchanberta-base-att-spm-uncased.json', 'r') as f:
    y_pred = json.load(f)
```

[331]:
```
# check our prediction with label
# -100 is special tokens : [<s>, </s>, _]
print(y_pred[0])
print(y_test[0])
```

```
[37, 29, 39, 26, 26, 26, 37, 37, 26, 26, 26, 41, 37, 37, 26, 26, 39, 26, 37]
[-100, 29, 39, 26, 26, 26, 37, -100, 26, 26, 26, 41, 37, -100, 26, 26, 39, 26,
-100]
```

[332]:
```
evaluation_report(y_test, y_pred)
```

```
speacial_tag : 21042
```

	tag	precision	recall	f1_score	correct_count
0	-100	-	0.0	-	0
1	0	50.0	25.0	33.333333	4
2	1	76.384535	72.376238	74.326385	731
3	2	50.0	4.310345	7.936508	5
4	3	57.5	39.655172	46.938776	23
5	4	88.333333	94.642857	91.37931	53
6	5	65.317919	65.317919	65.317919	113
7	6	93.39498	98.6053	95.929444	707
8	7	-	0.0	-	0
9	8	62.711864	75.126904	68.360277	296
10	10	91.772772	90.26975	91.015056	937
11	11	94.660194	85.526316	89.861751	390
12	12	54.83871	81.730769	65.637066	85
13	13	50.0	27.272727	35.294118	3
14	14	68.253968	83.495146	75.10917	86
15	15	89.473684	93.865031	91.616766	306
16	16	-	0.0	-	0
17	17	97.142857	95.967742	96.551724	238
18	18	69.70091	98.168498	81.520913	1072
19	19	-	0.0	-	0
20	20	100.0	76.470588	86.666667	13
21	21	93.619792	95.802798	94.698716	1438
22	22	72.39819	95.238095	82.262211	160
23	23	83.636364	96.842105	89.756098	92

```
24             24  88.725986  97.391787  92.857143         1755
25             25  84.262149  86.538462  85.385137         1890
26             26  89.422783  93.336315  91.337647        29218
27             27  82.302772  62.764228  71.217712          386
28             28  89.256198  93.103448  91.139241          108
29             29  88.811189  98.755832  93.519882          635
30             31  71.401274  86.430224  78.200209         2242
31             32       87.5      100.0  93.333333          147
32             33  65.116279  60.869565  62.921348           56
33             34       64.0       64.0       64.0           16
34             35   78.26087       60.0  67.924528           54
35             36   94.34365  87.698413  90.899743          884
36             37  40.123377  98.477966  57.016309        12358
37             38   92.34181  91.395961  91.866451         3123
38             39  88.809369  91.267719  90.021764         6825
39             40  69.061708  71.666667  70.340077          817
40             41  80.286468  78.743068  79.507279         2130
41             42  85.553279  93.609865  89.400428          835
42             43  95.211268  99.411765  97.266187          676
43             45  90.927022  96.645702  93.699187          461
44             46  96.065574   91.84953  93.910256          293
45   accuracy=91.89
```

7 Other Pretrained model

In this section, we will experiment by fine-tuning other pretrained models, such as airesearch/wangchanberta-base-wiki-newmm, to see how about their performance.

Since each model uses a different word-tokenization method. for example, **airesearch/wangchanberta-base-wiki-newmm uses newmm**, while **airesearch/wangchanberta-base-att-spm-uncased uses SentencePiece**. please try fine-tuning and compare the performance of these models.

7.0.1 #TODO 3

[333]:
```
model_names = [
    'airesearch/wangchanberta-base-att-spm-uncased',
    'airesearch/wangchanberta-base-wiki-newmm',
    'airesearch/wangchanberta-base-wiki-ssg',
    'airesearch/wangchanberta-base-wiki-sefr',
    'airesearch/wangchanberta-base-wiki-spm',
]

#@title Choose Pretrained Model
```

```
model_name = "airesearch/wangchanberta-base-wiki-newmm" #@param ["airesearch/
↪wangchanberta-base-att-spm-uncased", "airesearch/
↪wangchanberta-base-wiki-newmm", "airesearch/
↪wangchanberta-base-wiki-syllable", "airesearch/
↪wangchanberta-base-wiki-sefr", "airesearch/wangchanberta-base-wiki-spm"]

#create tokenizer
tokenizer = Tokenizer(model_name).from_pretrained(
                f'{model_name}',
                revision='main',
                model_max_length=416,)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/file_download.py:795:
FutureWarning: `resume_download` is deprecated and will be removed in version
1.0.0. Downloads always resume when possible. If you want to force a new
download, use `force_download=True`.
  warnings.warn(
The tokenizer class you load from this checkpoint is not the same type as the
class this function is called from. It may result in unexpected tokenization.
The tokenizer class you load from this checkpoint is 'RobertaTokenizer'.
The class this function is called from is 'ThaiWordsNewmmTokenizer'.
The tokenizer class you load from this checkpoint is not the same type as the
class this function is called from. It may result in unexpected tokenization.
The tokenizer class you load from this checkpoint is 'RobertaTokenizer'.
The class this function is called from is 'ThaiWordsNewmmTokenizer'.
```

[334]:
```
example = orchidl["train"][1899]
print('sentence :', example["sentence"])
tokenized_input = tokenizer([example["sentence"]], is_split_into_words=True)
tokens = tokenizer.convert_ids_to_tokens(tokenized_input["input_ids"])
print('tokens :',tokens)
print('label tokens :', example["label_tokens"])
```

```
sentence :                (bilingual transfer dictionary)
tokens : ['<s>', '  ', '    ', '  ', '     ', '   ', '', '<_>',
'<unk>', '<_>', 'transfer', '<_>', 'dictionary', ')', '</s>']
label tokens : ['  ', '    ', '  ', '     ', '   ', '', ' ', '(',
'bilingual transfer dictionary', ')']
```

It's the same problem as above.

```
**Warning: Can we use same function as above ?**
```

```
**Warning: Please beware of <unk>, an unknown word token.**
```

```
**Warning: Please be careful of "   ", the 'am' vowel. WangchanBERTa's internal
preprocessing replaces all "   " to '' and ' '**
```

[384]:
```
def majority_vote_pos(examples):
    """
```

```
    # TO DO: Since the tokens from the output of the pretrained tokenizer
    # do not match the tokens in the label tokens of the dataset,
    # the task is to create a function to determine the POS tags of the tokens generated by the pretrained tokenizer.
    # This should be done by referencing the POS tags in the label tokens. If a token partially overlaps with others,
    # the POS tag from the segment with the greater number of characters should be assigned.
    #
    # Example :
    # "     " (9 chars) is formed from "  " (3 chars) + "   " (6 chars).
    # "  " has a POS tag of 21,
    # and "   " has a POS tag of 39.
    # Therefore, the POS tag for "     " is 39,
    # as "     " is derived more from the "   " part than from the "  " part.
    #
    # '      ' (10 chars) is formed from '  ' (3 chars) + '   ' (7 chars)
    # "  " has a POS tag of 26,
    # and "    " has a POS tag of 2.
    # Therefore, the POS tag for "      " is 2,
    # as "      " is derived more from the "  " part than from the "   " part.
    """
    # TODO: FILL CODE HERE
    tokenized_inputs = tokenizer([examples["sentence"]], is_split_into_words=True)
    new_tokens = tokenizer.convert_ids_to_tokens(tokenized_inputs["input_ids"])
    label_tokens = examples["label_tokens"]
    pos_tags = examples["pos_tags"]

    new_pos_result = []
    label_idx = 0  # Current index in label_tokens
    char_idx = 0   # Current character index within the current label token

    for token in new_tokens:
        if token in ["<s>", "</s>", "_", "<unk>"]:
            new_pos_result.append(-100)
            continue

        token = token.replace(' ', ' ').replace("<_>", " ")
        if token.startswith(" "):
            token = token[1:]

        if not token:
            new_pos_result.append(-100)
            continue

        pos_counts = {}
```

```
        accumulated = ""
        token_length = len(token)
        prev_accumulated_len = -1  # Track previous length to detect no progress

        while accumulated != token and label_idx < len(label_tokens):
            current_label = label_tokens[label_idx]
            remaining_in_label = current_label[char_idx:] if char_idx < len(current_label) else ''
            max_overlap = token_length - len(accumulated)
            overlap = min(len(remaining_in_label), max_overlap)

            # Check if no progress is made
            if overlap == 0 and len(accumulated) == prev_accumulated_len:
                break  # Exit loop to avoid infinite iteration
            prev_accumulated_len = len(accumulated)

            accumulated += remaining_in_label[:overlap]
            for _ in range(overlap):
                pos_tag = pos_tags[label_idx]
                pos_counts[pos_tag] = pos_counts.get(pos_tag, 0) + 1
            char_idx += overlap

            if char_idx >= len(current_label):
                label_idx += 1
                char_idx = 0

        best_pos = max(pos_counts, key=pos_counts.get) if pos_counts else -100
        new_pos_result.append(best_pos)

    tokenized_inputs['tokens'] = new_tokens
    tokenized_inputs['labels'] = new_pos_result
    return tokenized_inputs
```

[385]:
```
tokenized_orchid = orchidl.map(majority_vote_pos)
```

```
Map:   0%|          | 0/18500 [00:00<?, ? examples/s]

Map:   0%|          | 0/4625 [00:00<?, ? examples/s]
```

[386]:
```
# hard test case
example = tokenized_orchid["train"][1899]
for i in example :
    print(i, ":", example[i])
```

```
id : 1899
label_tokens : ['  ', '   ', '  ', '    ', '  ', '', ' ', '(',
'bilingual transfer dictionary', ')']
pos_tags : [25, 39, 38, 26, 26, 5, 37, 37, 26, 37]
sentence :                 (bilingual transfer dictionary)
```

```
input_ids : [0, 80, 3973, 45, 12252, 3496, 592, 5, 3, 5, 30055, 5, 63190, 178,
2]
token_type_ids : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
attention_mask : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
tokens : ['<s>', ' ', '    ', ' ', '     ', ' ', '', '<_>',
'<unk>', '<_>', 'transfer', '<_>', 'dictionary', ')', '</s>']
labels : [-100, 25, 39, 38, 26, 26, 5, 37, -100, 37, 26, 26, 26, 26, -100]
```

[369]:
```python
model_names = [
    'wangchanberta-base-att-spm-uncased',
    'wangchanberta-base-wiki-newmm',
    'wangchanberta-base-wiki-ssg',
    'wangchanberta-base-wiki-sefr',
    'wangchanberta-base-wiki-spm',
]

#@title Choose Pretrained Model
model_name = "wangchanberta-base-wiki-newmm" #@param ["wangchanberta-base-att-spm-uncased", "wangchanberta-base-wiki-newmm", "wangchanberta-base-wiki-syllable", "wangchanberta-base-wiki-sefr", "wangchanberta-base-wiki-spm"]

#create model
model = AutoModelForTokenClassification.from_pretrained(
    f"airesearch/{model_name}",
    revision='main',
    num_labels=47, id2label=id2label, label2id=label2id
)
```

```
Some weights of the model checkpoint at airesearch/wangchanberta-base-wiki-newmm
were not used when initializing RobertaForTokenClassification:
['lm_head.dense.weight', 'lm_head.decoder.bias', 'lm_head.layer_norm.bias',
'lm_head.layer_norm.weight', 'lm_head.dense.bias', 'lm_head.decoder.weight',
'lm_head.bias']
- This IS expected if you are initializing RobertaForTokenClassification from
the checkpoint of a model trained on another task or with another architecture
(e.g. initializing a BertForSequenceClassification model from a
BertForPreTraining model).
- This IS NOT expected if you are initializing RobertaForTokenClassification
from the checkpoint of a model that you expect to be exactly identical
(initializing a BertForSequenceClassification model from a
BertForSequenceClassification model).
Some weights of RobertaForTokenClassification were not initialized from the
model checkpoint at airesearch/wangchanberta-base-wiki-newmm and are newly
initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

[370]:
```
data_collator = DataCollatorForTokenClassification(tokenizer=tokenizer)
```

7.0.2 #TODO 4

Fine-tuning other pretrained model with our orchid corpus.

[372]:
```
from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="./models-pos-wangchanberta-base-wiki-newmm",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=10,
    report_to="none",
    save_total_limit=2,
    fp16=True,
    push_to_hub=True,
    hub_model_id="pupipatsk/pos-wangchanberta-base-wiki-newmm",  # Specify␣
 ↪model name
    hub_strategy="every_save",  # Upload model after each save
)

from transformers import Trainer

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_orchid["train"],
    eval_dataset=tokenized_orchid["test"],
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics
)

trainer.train()
```

```
/usr/local/lib/python3.10/dist-
packages/huggingface_hub/utils/_deprecation.py:131: FutureWarning: 'Repository'
(from 'huggingface_hub.repository') is deprecated and will be removed from
version '1.0'. Please prefer the http-based alternatives instead. Given its
large adoption in legacy code, the complete removal is only planned on next
major release.
```

```
For more details, please read
https://huggingface.co/docs/huggingface_hub/concepts/git_vs_http.
  warnings.warn(warning_message, FutureWarning)
Cloning https://huggingface.co/pupipatsk/pos-wangchanberta-base-wiki-newmm into
local empty directory.
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:411:
FutureWarning: This implementation of AdamW is deprecated and will be removed in
a future version. Use the PyTorch implementation torch.optim.AdamW instead, or
set `no_deprecation_warning=True` to disable this warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torch/nn/parallel/_functions.py:71:
UserWarning: Was asked to gather along dimension 0, but all input tensors were
scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

```
<IPython.core.display.HTML object>
```

```
/usr/local/lib/python3.10/dist-packages/torch/nn/parallel/_functions.py:71:
UserWarning: Was asked to gather along dimension 0, but all input tensors were
scalars; will instead unsqueeze and return a vector.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torch/nn/parallel/_functions.py:71:
UserWarning: Was asked to gather along dimension 0, but all input tensors were
scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

[372]:
```
TrainOutput(global_step=1737, training_loss=0.6481462119294091,
metrics={'train_runtime': 681.0439, 'train_samples_per_second': 81.493,
'train_steps_per_second': 2.55, 'total_flos': 1371637012822776.0, 'train_loss':
0.6481462119294091, 'epoch': 3.0})
```

[381]:
```python
######## EVALUATE YOUR MODEL ########
test_data = tokenized_orchid["test"]
y_test = [inputs['labels'] for inputs in test_data]

model = AutoModelForTokenClassification.from_pretrained("pupipatsk/
 ↪pos-wangchanberta-base-wiki-newmm")
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

y_pred = []
# run inference
for inputs in tqdm(test_data):
    text = inputs['sentence']
    inputs = tokenizer(text, return_tensors="pt").to(device)
    with torch.no_grad():
        pred = model(**inputs).logits  # Model prediction
        predictions = torch.argmax(pred, dim=2)  # Convert logits to labels
```

```python
        y_pred.append(predictions.cpu().tolist()[0])

# save y_pred to local
with open('y_pred-pos-wangchanberta-base-wiki-newmm.json', 'w') as f:
    json.dump(y_pred, f)

# load y_pred from local
with open('y_pred-pos-wangchanberta-base-wiki-newmm.json', 'r') as f:
    y_pred = json.load(f)

print(y_pred[0])
print(y_test[0])

evaluation_report(y_test, y_pred)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/file_download.py:795:
FutureWarning: `resume_download` is deprecated and will be removed in version
1.0.0. Downloads always resume when possible. If you want to force a new
download, use `force_download=True`.
  warnings.warn(

config.json:   0%|          | 0.00/2.42k [00:00<?, ?B/s]

pytorch_model.bin:   0%|          | 0.00/643M [00:00<?, ?B/s]

/usr/local/lib/python3.10/dist-packages/transformers/modeling_utils.py:463:
FutureWarning: You are using `torch.load` with `weights_only=False` (the current
default value), which uses the default pickle module implicitly. It is possible
to construct malicious pickle data which will execute arbitrary code during
unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for
more details). In a future release, the default value for `weights_only` will be
flipped to `True`. This limits the functions that could be executed during
unpickling. Arbitrary objects will no longer be allowed to be loaded via this
mode unless they are explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control of the
loaded file. Please open an issue on GitHub for any issues related to this
experimental feature.
  return torch.load(checkpoint_file, map_location="cpu")
100%|     | 4625/4625 [00:52<00:00, 88.34it/s]

[26, 29, 37, 21, 39, 26, 26, 37, 26, 26, 41, 37, 26, 39, 39, 26, 37]
[-100, 29, 37, 21, 39, 26, 26, 37, 26, 26, 41, 37, 26, 39, 39, 26, -100]
speacial_tag : 11485

               tag  precision     recall   f1_score correct_count
0             -100          -        0.0          -             0
1                0  22.222222  26.666667  24.242424             4
2                1  71.910112  60.606061  65.775951           640
```

3	2	30.0	2.678571	4.918033	3
4	3	57.692308	26.315789	36.144578	15
5	4	83.636364	75.409836	79.310345	46
6	5	70.16129	54.375	61.267606	87
7	6	73.298429	62.921348	67.714631	280
8	7	0.0	0.0	-	0
9	8	63.207547	62.229102	62.714509	201
10	10	74.610778	74.969916	74.789916	623
11	11	91.59292	82.142857	86.610879	414
12	12	49.285714	65.09434	56.097561	69
13	13	-	0.0	-	0
14	14	79.230769	72.027972	75.457875	103
15	15	90.604027	85.173502	87.804878	270
16	16	-	0.0	-	0
17	17	84.722222	89.705882	87.142857	244
18	18	83.923941	74.235474	78.782961	971
19	19	-	0.0	-	0
20	20	81.818182	60.0	69.230769	9
21	21	87.617766	85.507881	86.549967	1953
22	22	74.675325	81.560284	77.966102	115
23	23	82.608696	73.786408	77.948718	76
24	24	92.113734	88.09646	90.06032	1717
25	25	85.757576	82.576073	84.136759	1981
26	26	71.50413	87.867603	78.845808	21901
27	27	60.616438	38.064516	46.76354	177
28	28	91.41791	91.41791	91.41791	245
29	29	77.934936	88.585209	82.919488	551
30	31	63.113093	76.286134	69.077196	1557
31	32	83.333333	77.777778	80.45977	70
32	33	59.52381	60.240964	59.88024	50
33	34	68.181818	68.181818	68.181818	15
34	35	89.130435	66.666667	76.27907	82
35	36	92.911392	81.314623	86.727058	1101
36	37	60.502981	85.346129	70.808731	7915
37	38	91.606399	84.683955	88.009264	4180
38	39	65.52838	86.606489	74.607261	7100
39	40	65.050167	58.805745	61.770544	778
40	41	64.250582	74.467447	68.982768	2482
41	42	86.284289	84.390244	85.326757	1038
42	43	91.119221	92.69802	91.90184	749
43	45	94.230769	95.516569	94.869313	980
44	46	88.950276	90.449438	89.693593	322
45	accuracy=83.60				

7.0.3 #TODO 5

Compare the results between both models. Are they comparable? (Think about the ground truths of both models).

Propose a way to fairly evaluate the models.

Write your answer here :

WangchanBERTa-base-att-spm-uncased (SPM): 91.89% due to its ability to handle complex terms and unseen words.
WangchanBERTa-base-wiki-newmm (Newmm): 83.60% aligns better with human-readable segmentation (‘ ’ instead of ‘ ’)
attributed to its subword tokenization handling complex terms better, while Newmm's word-level tokenization aligned more closely with the ORCHID corpus's original segmentation but struggled with unseen words.

To fairly evaluate models with differing tokenization strategies, predictions should be mapped to the dataset's original word-level labels, alongside stratified evaluation of POS categories, cross-validation, error analysis, and statistical testing.

While SPM's flexibility enhances performance, Newmm's alignment with human-interpretable word tokens offers practical advantages.

A note on preprocessing data.

`process_transformers` in `thaixtransformers.preprocess` also provides a preprocess code that deals with many issues such as casing, text cleaning, and white space replacement with <_>. You can also use this to preprocess your text. Note that space replacement is done automatically without preprocessing in thaixtransformers.