ner

July 23, 2024

0.1 Import Necessary Libraries

[3]:
```
import datasets
import numpy as np
from transformers import BertTokenizerFast
from transformers import DataCollatorForTokenClassification
from transformers import AutoModelForTokenClassification
```

0.2 Load Data from HuggingFace

[4]:
```
from datasets import load_dataset

raw_datasets = load_dataset("conll2003", trust_remote_code=True)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89:
UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
(https://huggingface.co/settings/tokens), set it as secret in your Google Colab
and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
public models or datasets.
  warnings.warn(
```

```
Downloading builder script:   0%|          | 0.00/9.57k [00:00<?, ?B/s]

Downloading readme:   0%|          | 0.00/12.3k [00:00<?, ?B/s]

Downloading data:   0%|          | 0.00/983k [00:00<?, ?B/s]

Generating train split:   0%|          | 0/14041 [00:00<?, ? examples/s]

Generating validation split:   0%|          | 0/3250 [00:00<?, ? examples/s]

Generating test split:   0%|          | 0/3453 [00:00<?, ? examples/s]
```

[5]:
```
raw_datasets
```

[5]:
```
DatasetDict({
    train: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'],
        num_rows: 14041
    })
    validation: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'],
        num_rows: 3250
    })
    test: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'],
        num_rows: 3453
    })
})
```

[6]:
```
raw_datasets.shape
```

[6]:
```
{'train': (14041, 5), 'validation': (3250, 5), 'test': (3453, 5)}
```

[7]:
```
raw_datasets["train"][0]
```

[7]:
```
{'id': '0',
 'tokens': ['EU',
  'rejects',
  'German',
  'call',
  'to',
  'boycott',
  'British',
  'lamb',
  '.'],
 'pos_tags': [22, 42, 16, 21, 35, 37, 16, 21, 7],
 'chunk_tags': [11, 21, 11, 12, 21, 22, 11, 12, 0],
 'ner_tags': [3, 0, 7, 0, 0, 0, 7, 0, 0]}
```

[8]:
```
raw_datasets["train"][0]["tokens"]
```

[8]:
```
['EU', 'rejects', 'German', 'call', 'to', 'boycott', 'British', 'lamb', '.']
```

[10]:
```
raw_datasets["train"].features["ner_tags"].feature.names
```

[10]:
```
['O', 'B-PER', 'I-PER', 'B-ORG', 'I-ORG', 'B-LOC', 'I-LOC', 'B-MISC', 'I-MISC']
```

[ ]:
```
raw_datasets['train'].description
```

[ ]:
```
'The shared task of CoNLL-2003 concerns language-independent named entity
recognition. We will concentrate on\nfour types of named entities: persons,
```

locations, organizations and names of miscellaneous entities that do\nnot belong to the previous three groups.\n\nThe CoNLL-2003 shared task data files contain four columns separated by a single space. Each word has been put on\na separate line and there is an empty line after each sentence. The first item on each line is a word, the second\na part-of-speech (POS) tag, the third a syntactic chunk tag and the fourth the named entity tag. The chunk tags\nand the named entity tags have the format I-TYPE which means that the word is inside a phrase of type TYPE. Only\nif two phrases of the same type immediately follow each other, the first word of the second phrase will have tag\nB-TYPE to show that it starts a new phrase. A word with tag O is not part of a phrase. Note the dataset uses IOB2\ntagging scheme, whereas the original dataset uses IOB1.\n\nFor more details see https://www.clips.uantwerpen.be/conll2003/ner/ and https://www.aclweb.org/anthology/W03-0419\n'

0.3 Tokenizer

[11]:
```
#  used to convert text into a format that a BERT model can understand, process␣
↪and tokenize text for input into a BERT model.
tokenizer = BertTokenizerFast.from_pretrained("bert-base-uncased")
```

```
tokenizer_config.json:   0%|          | 0.00/48.0 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]

config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]
```

[12]:
```
ex = raw_datasets['train'][0]
# get sequences of ex text
tokenized_input = tokenizer(ex['tokens'], is_split_into_words=True)
# return each number to it's word
tokens = tokenizer.convert_ids_to_tokens(tokenized_input['input_ids'])
# words inddices
word_ids = tokenized_input.word_ids()
```

[ ]:
```
len(ex['ner_tags']), len(tokenized_input['input_ids'])
```

[ ]: (9, 11)

[13]:
```
'''
input ids return by tokenizer longer than labels,
becouse special chars or tokenizer may split any word into multiple tokens
so will build tokenize_and_align_labels function to handle this problem
'''
```

[13]: '\ninput ids return by tokenizer longer than labels,\nbecouse special chars or tokenizer may split any word into multiple tokens\nso will build

```
tokenize_and_align_labels function to handle this problem\n'
```

[14]:
```
def tokenize_and_align_labels(examples, label_all_tokens=True):
    tokenized_inputs = tokenizer(examples["tokens"], truncation=True,␣
↪is_split_into_words=True)
    labels = []
    for i, label in enumerate(examples["ner_tags"]):
        word_ids = tokenized_inputs.word_ids(batch_index=i)
        # word_ids() => Return a list mapping the tokens
        # to their actual word in the initial sentence.
        # It Returns a list indicating the word corresponding to each token.
        previous_word_idx = None
        label_ids = []
        # Special tokens like `<s>` and `<\s>` are originally mapped to None
        # We need to set the label to -100 so they are automatically ignored in␣
↪the loss function.
        for word_idx in word_ids:
            if word_idx is None:
                # set -100 as the label for these special tokens
                label_ids.append(-100)
            # For the other tokens in a word, we set the label to either the␣
↪current label or -100, depending on
            # the label_all_tokens flag.
            elif word_idx != previous_word_idx:
                # if current word_idx is != prev then its the most regular case
                # and add the corresponding token
                label_ids.append(label[word_idx])
            else:
                # to take care of sub-words which have the same word_idx
                # set -100 as well for them, but only if label_all_tokens ==␣
↪False
                label_ids.append(label[word_idx] if label_all_tokens else -100)
                # mask the subword representations after the first subword

            previous_word_idx = word_idx
        labels.append(label_ids)
    tokenized_inputs["labels"] = labels
    return tokenized_inputs
```

[15]:
```
q = tokenize_and_align_labels(raw_datasets['train'][4:5])
print(raw_datasets['train'][4]['tokens'])
print(q)
```

```
['Germany', "'s", 'representative', 'to', 'the', 'European', 'Union', "'s",
'veterinary', 'committee', 'Werner', 'Zwingmann', 'said', 'on', 'Wednesday',
'consumers', 'should', 'buy', 'sheepmeat', 'from', 'countries', 'other', 'than',
'Britain', 'until', 'the', 'scientific', 'advice', 'was', 'clearer', '.']
```

{'input_ids': [[101, 2762, 1005, 1055, 4387, 2000, 1996, 2647, 2586, 1005, 1055, 15651, 2837, 14121, 1062, 9328, 5804, 2056, 2006, 9317, 10390, 2323, 4965, 8351, 4168, 4017, 2013, 3032, 2060, 2084, 3725, 2127, 1996, 4045, 6040, 2001, 24509, 1012, 102]], 'token_type_ids': [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]], 'labels': [[-100, 5, 0, 0, 0, 0, 0, 3, 4, 0, 0, 0, 0, 1, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, -100]]}

[ ]:
```python
for token, label in zip(tokenizer.
    convert_ids_to_tokens(q["input_ids"][0]),q["labels"][0]):
    print(f"{token:_<40} {label}")
```

```
[CLS]___________________________________ -100
germany_________________________________ 5
'_______________________________________ 0
s_______________________________________ 0
representative__________________________ 0
to______________________________________ 0
the_____________________________________ 0
european________________________________ 3
union___________________________________ 4
'_______________________________________ 0
s_______________________________________ 0
veterinary______________________________ 0
committee_______________________________ 0
werner__________________________________ 1
z_______________________________________ 2
##wing__________________________________ 2
##mann__________________________________ 2
said____________________________________ 0
on______________________________________ 0
wednesday_______________________________ 0
consumers_______________________________ 0
should__________________________________ 0
buy_____________________________________ 0
sheep___________________________________ 0
##me____________________________________ 0
##at____________________________________ 0
from____________________________________ 0
countries_______________________________ 0
other___________________________________ 0
than____________________________________ 0
britain_________________________________ 5
until___________________________________ 0
the_____________________________________ 0
scientific______________________________ 0
```

```
advice___________________________________ 0
was______________________________________ 0
clearer__________________________________ 0
.________________________________________ 0
[SEP]____________________________________ -100
```

[16]:
```python
# map tokenizer to all data set
tokenized_datasets = raw_datasets.map(tokenize_and_align_labels, batched=True)
```

```
Map:   0%|          | 0/14041 [00:00<?, ? examples/s]

Map:   0%|          | 0/3250 [00:00<?, ? examples/s]

Map:   0%|          | 0/3453 [00:00<?, ? examples/s]
```

0.4 Model

[17]:
```python
model = AutoModelForTokenClassification.from_pretrained("bert-base-uncased",␣
  ↪num_labels=9)
```

```
model.safetensors:   0%|          | 0.00/440M [00:00<?, ?B/s]

Some weights of BertForTokenClassification were not initialized from the model
checkpoint at bert-base-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.
```

[18]:
```python
from transformers import TrainingArguments, Trainer
args = TrainingArguments(
    "test-ner",
    evaluation_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,
    )
```

```
/usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1494:
FutureWarning: `evaluation_strategy` is deprecated and will be removed in
version 4.46 of   Transformers. Use `eval_strategy` instead
  warnings.warn(
```

[19]:
```python
# automatically padding some model inputs to the length of the longest example
data_collator = DataCollatorForTokenClassification(tokenizer)
```

[20]:
```python
metric = datasets.load_metric("seqeval", trust_remote_code=True)
```

```
<ipython-input-20-ca4db3689d00>:1: FutureWarning: load_metric is deprecated and
will be removed in the next major version of datasets. Use 'evaluate.load'
instead, from the new library  Evaluate: https://huggingface.co/docs/evaluate
  metric = datasets.load_metric("seqeval", trust_remote_code=True)
```

```
Downloading builder script:   0%|          | 0.00/2.47k [00:00<?, ?B/s]
```

[21]:
```
example = raw_datasets['train'][0]
label_list = raw_datasets["train"].features["ner_tags"].feature.names

label_list
```

[21]:
```
['O', 'B-PER', 'I-PER', 'B-ORG', 'I-ORG', 'B-LOC', 'I-LOC', 'B-MISC', 'I-MISC']
```

[22]:
```
# give example on metric
labels = [label_list[i] for i in example["ner_tags"]]

metric.compute(predictions=[labels], references=[labels])
```

[22]:
```
{'MISC': {'precision': 1.0, 'recall': 1.0, 'f1': 1.0, 'number': 2},
 'ORG': {'precision': 1.0, 'recall': 1.0, 'f1': 1.0, 'number': 1},
 'overall_precision': 1.0,
 'overall_recall': 1.0,
 'overall_f1': 1.0,
 'overall_accuracy': 1.0}
```

[23]:
```
# control output function
def compute_metrics(eval_preds):
    """
    Function to compute the evaluation metrics for Named Entity Recognition␣
  ↪(NER) tasks.
    The function computes precision, recall, F1 score and accuracy.

    Parameters:
    eval_preds (tuple): A tuple containing the predicted logits and the true␣
  ↪labels.

    Returns:
    A dictionary containing the precision, recall, F1 score and accuracy.
    """
    pred_logits, labels = eval_preds

    pred_logits = np.argmax(pred_logits, axis=2)
    # the logits and the probabilities are in the same order,
    # so we don't need to apply the softmax

    # We remove all the values where the label is -100
    predictions = [
```

```
        [label_list[eval_preds] for (eval_preds, l) in zip(prediction, label)␣
↪if l != -100]
        for prediction, label in zip(pred_logits, labels)
    ]

    true_labels = [
      [label_list[l] for (eval_preds, l) in zip(prediction, label) if l != -100]
       for prediction, label in zip(pred_logits, labels)
   ]
    results = metric.compute(predictions=predictions, references=true_labels)
    return {
   "precision": results["overall_precision"],
   "recall": results["overall_recall"],
   "f1": results["overall_f1"],
  "accuracy": results["overall_accuracy"],
  }
```

[24]:
```
trainer = Trainer(
    model,
    args,
   train_dataset=tokenized_datasets["train"],
   eval_dataset=tokenized_datasets["validation"],
   data_collator=data_collator,
   tokenizer=tokenizer,
   compute_metrics=compute_metrics
)
```

[25]:
```
trainer.train()
model.save_pretrained("ner_model")
tokenizer.save_pretrained("tokenizer")
```

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

[25]:
```
('tokenizer/tokenizer_config.json',
 'tokenizer/special_tokens_map.json',
 'tokenizer/vocab.txt',
 'tokenizer/added_tokens.json',
 'tokenizer/tokenizer.json')
```

0.5 Load Fine Tuned Model

[26]:
```
id2label = {
    str(i): label for i,label in enumerate(label_list)
}
label2id = {
    label: str(i) for i,label in enumerate(label_list)
}
```

[27]:
```
import json
```

[28]:
```
config = json.load(open("ner_model/config.json"))
```

[29]:
```
config["id2label"] = id2label
config["label2id"] = label2id
```

[30]:
```
json.dump(config, open("ner_model/config.json","w"))
```

[31]:
```
model_fine_tuned = AutoModelForTokenClassification.from_pretrained("ner_model")
```

0.6 Pipeline

[32]:
```
from transformers import pipeline
```

[33]:
```
nlp = pipeline("ner", model=model_fine_tuned, tokenizer=tokenizer)


example = "Ayman applied for an intership at widebot"

ner_results = nlp(example)

print(ner_results)
```

```
Hardware accelerator e.g. GPU is available in the environment, but no `device`
argument is passed to the `Pipeline` object. Model will be on CPU.
```

```
[{'entity': 'B-PER', 'score': 0.99834406, 'index': 1, 'word': 'a', 'start': 0,
'end': 1}, {'entity': 'B-PER', 'score': 0.99818975, 'index': 2, 'word':
'##yman', 'start': 1, 'end': 5}, {'entity': 'B-ORG', 'score': 0.98740256,
'index': 9, 'word': 'wide', 'start': 34, 'end': 38}, {'entity': 'B-ORG',
'score': 0.72021335, 'index': 10, 'word': '##bot', 'start': 38, 'end': 41}]
```

[ ]:
```
```