ShritejShrikant_Chavan_HW6a

October 30, 2023

1 Token Classification with DistilBert- NER

1.1 Outline

1. **Setting up the Environment**: Installing necessary libraries and setting up paths.
2. **Creating Huggingface Dataset for Custom Dataset**: Understanding the structure and content of the dataset.
3. **Data Preprocessing**: Techniques to prepare the data for training, including handling different data splits and tokenization
4. **Training the Model**: Feeding data and adjusting weights.
5. **Inference**: Evaluate model on test set and making predictions.

2 Setting up the Environment

[1]:
```
# CHANGE FOLDERS AS PER YOUR SETUP
from pathlib import Path
if 'google.colab' in str(get_ipython()):
    from google.colab import drive
    drive.mount("/content/drive")
    !pip install datasets transformers evaluate wandb accelerate seqeval -U -qq
    base_folder = Path("/content/drive/MyDrive/NLP")
else:
    base_folder = Path("/home/harpreet/Insync/google_drive_shaannoor/data")


from transformers import AutoConfig, AutoModelForTokenClassification,␣
↪AutoTokenizer, Trainer, TrainingArguments
from transformers import AutoTokenizer, DataCollatorForTokenClassification,␣
↪pipeline
from datasets import load_dataset, DatasetDict, Dataset, ClassLabel, Sequence
import evaluate


import wandb
```

```python
import numpy as np
# from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

import textwrap
```

```
Mounted at /content/drive
                           493.7/493.7

kB 8.4 MB/s eta 0:00:00
                           7.7/7.7 MB
92.9 MB/s eta 0:00:00
                           84.1/84.1 kB
11.3 MB/s eta 0:00:00
                           2.1/2.1 MB
89.7 MB/s eta 0:00:00
                           261.0/261.0

kB 31.3 MB/s eta 0:00:00
                           43.6/43.6 kB
5.6 MB/s eta 0:00:00
  Preparing metadata (setup.py) … done
                           115.3/115.3

kB 15.8 MB/s eta 0:00:00
                           134.8/134.8

kB 17.7 MB/s eta 0:00:00
                           302.0/302.0

kB 34.0 MB/s eta 0:00:00
                           3.8/3.8 MB
104.6 MB/s eta 0:00:00
                           1.3/1.3 MB
82.8 MB/s eta 0:00:00
                           190.6/190.6

kB 24.2 MB/s eta 0:00:00
                           241.0/241.0

kB 29.0 MB/s eta 0:00:00
  Preparing metadata (setup.py) … done
                           62.7/62.7 kB
7.9 MB/s eta 0:00:00
                           295.0/295.0

kB 32.3 MB/s eta 0:00:00
  Building wheel for seqeval (setup.py) … done
```

```
Building wheel for pathtools (setup.py) … done
```

[27]:
```python
# CHANGE FOLDERS TO WHERE YOU WANT TO SAVE DATA AND MODELS
data_folder = base_folder/'datasets/brown_corpus'
model_folder = base_folder/'models/nlp_spring_2023/ner'
model_folder.mkdir(exist_ok=True)
data_folder.mkdir(exist_ok=True)
```

[28]:
```python
def print_wrap(text, d):
    # If the text is a list, convert it to a string
    if isinstance(text, list):
        # Convert None values to a default string (e.g., "None" or an empty␣
↪string)
        text = ' '.join(str(item) if item is not None else "None" for item in␣
↪text)

    # Wrap the text to limit the width to 'd'
    wrapped_text = textwrap.fill(text, width=d)

    # Print the wrapped text
    print(wrapped_text)
```

3 Exploring and Understanding Dataset

3.1 conll2003 Dataset

3.2 Load Data set

[29]:
```python
conll_dataset = load_dataset('conll2003')
```

[30]:
```python
conll_dataset
```

[30]:
```
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
    })
})
```

3.3 Understanding your data

[31]:
```
print(conll_dataset)
```

```
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
    })
})
```

3.4 Understanding the datatype of columns

[32]:
```
conll_dataset['train'].features
```

[32]:
```
{'id': Value(dtype='string', id=None),
 'tokens': Sequence(feature=Value(dtype='string', id=None), length=-1, id=None),
 'pos_tags': Sequence(feature=ClassLabel(names=['"', "''", '#', '$', '(', ')',
',', '.', ':', '``', 'CC', 'CD', 'DT', 'EX', 'FW', 'IN', 'JJ', 'JJR', 'JJS',
'LS', 'MD', 'NN', 'NNP', 'NNPS', 'NNS', 'NN|SYM', 'PDT', 'POS', 'PRP', 'PRP$',
'RB', 'RBR', 'RBS', 'RP', 'SYM', 'TO', 'UH', 'VB', 'VBD', 'VBG', 'VBN', 'VBP',
'VBZ', 'WDT', 'WP', 'WP$', 'WRB'], id=None), length=-1, id=None),
 'chunk_tags': Sequence(feature=ClassLabel(names=['O', 'B-ADJP', 'I-ADJP',
'B-ADVP', 'I-ADVP', 'B-CONJP', 'I-CONJP', 'B-INTJ', 'I-INTJ', 'B-LST', 'I-LST',
'B-NP', 'I-NP', 'B-PP', 'I-PP', 'B-PRT', 'I-PRT', 'B-SBAR', 'I-SBAR', 'B-UCP',
'I-UCP', 'B-VP', 'I-VP'], id=None), length=-1, id=None),
 'ner_tags': Sequence(feature=ClassLabel(names=['O', 'B-PER', 'I-PER', 'B-ORG',
'I-ORG', 'B-LOC', 'I-LOC', 'B-MISC', 'I-MISC'], id=None), length=-1, id=None)}
```

[33]:
```
conll_dataset['test'].features
```

[33]:
```
{'id': Value(dtype='string', id=None),
 'tokens': Sequence(feature=Value(dtype='string', id=None), length=-1, id=None),
 'pos_tags': Sequence(feature=ClassLabel(names=['"', "''", '#', '$', '(', ')',
',', '.', ':', '``', 'CC', 'CD', 'DT', 'EX', 'FW', 'IN', 'JJ', 'JJR', 'JJS',
'LS', 'MD', 'NN', 'NNP', 'NNPS', 'NNS', 'NN|SYM', 'PDT', 'POS', 'PRP', 'PRP$',
'RB', 'RBR', 'RBS', 'RP', 'SYM', 'TO', 'UH', 'VB', 'VBD', 'VBG', 'VBN', 'VBP',
'VBZ', 'WDT', 'WP', 'WP$', 'WRB'], id=None), length=-1, id=None),
 'chunk_tags': Sequence(feature=ClassLabel(names=['O', 'B-ADJP', 'I-ADJP',
```

```
'B-ADVP', 'I-ADVP', 'B-CONJP', 'I-CONJP', 'B-INTJ', 'I-INTJ', 'B-LST', 'I-LST',
'B-NP', 'I-NP', 'B-PP', 'I-PP', 'B-PRT', 'I-PRT', 'B-SBAR', 'I-SBAR', 'B-UCP',
'I-UCP', 'B-VP', 'I-VP'], id=None), length=-1, id=None),
 'ner_tags': Sequence(feature=ClassLabel(names=['O', 'B-PER', 'I-PER', 'B-ORG',
'I-ORG', 'B-LOC', 'I-LOC', 'B-MISC', 'I-MISC'], id=None), length=-1, id=None)}
```

3.5 Acess indivdual element

[34]:
```
# get the first example of the dataset
conll_dataset['train'][0]
```

[34]:
```
{'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]}
```

[35]:
```
print_wrap(conll_dataset['train']['tokens'][0], 80)
```

```
EU rejects German call to boycott British lamb .
```

[36]:
```
print_wrap(conll_dataset['train']['ner_tags'][0], 80)
```

```
3 0 7 0 0 0 7 0 0
```

3.6 Exploratory Data Analysis (EDA)

3.6.1 Change dataset format to Pandas

[37]:
```
# Set the format to Pandas
conll_dataset.set_format(type='pandas')
```

[38]:
```
# get all rows the dataset
df = conll_dataset['train'][:]
```

[39]:
```
df.head()
```

[39]:
```
  id                                            tokens  \
0  0  [EU, rejects, German, call, to, boycott, Briti…
1  1                                [Peter, Blackburn]
```

```
2  2                            [BRUSSELS, 1996-08-22]
3  3  [The, European, Commission, said, on, Thursday…
4  4  [Germany, 's, representative, to, the, Europea…

                                            pos_tags  \
0              [22, 42, 16, 21, 35, 37, 16, 21, 7]
1                                         [22, 22]
2                                         [22, 11]
3  [12, 22, 22, 38, 15, 22, 28, 38, 15, 16, 21, 3…
4  [22, 27, 21, 35, 12, 22, 22, 27, 16, 21, 22, 2…

                                          chunk_tags  \
0              [11, 21, 11, 12, 21, 22, 11, 12, 0]
1                                         [11, 12]
2                                         [11, 12]
3  [11, 12, 12, 21, 13, 11, 11, 21, 13, 11, 12, 1…
4  [11, 11, 12, 13, 11, 12, 12, 11, 12, 12, 12, 1…

                                            ner_tags
0                      [3, 0, 7, 0, 0, 0, 7, 0, 0]
1                                           [1, 2]
2                                           [5, 0]
3  [0, 3, 4, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, …
4  [5, 0, 0, 0, 0, 3, 4, 0, 0, 0, 1, 2, 0, 0, 0, …
```

[40]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14041 entries, 0 to 14040
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   id          14041 non-null  object
 1   tokens      14041 non-null  object
 2   pos_tags    14041 non-null  object
 3   chunk_tags  14041 non-null  object
 4   ner_tags    14041 non-null  object
dtypes: object(5)
memory usage: 548.6+ KB
```

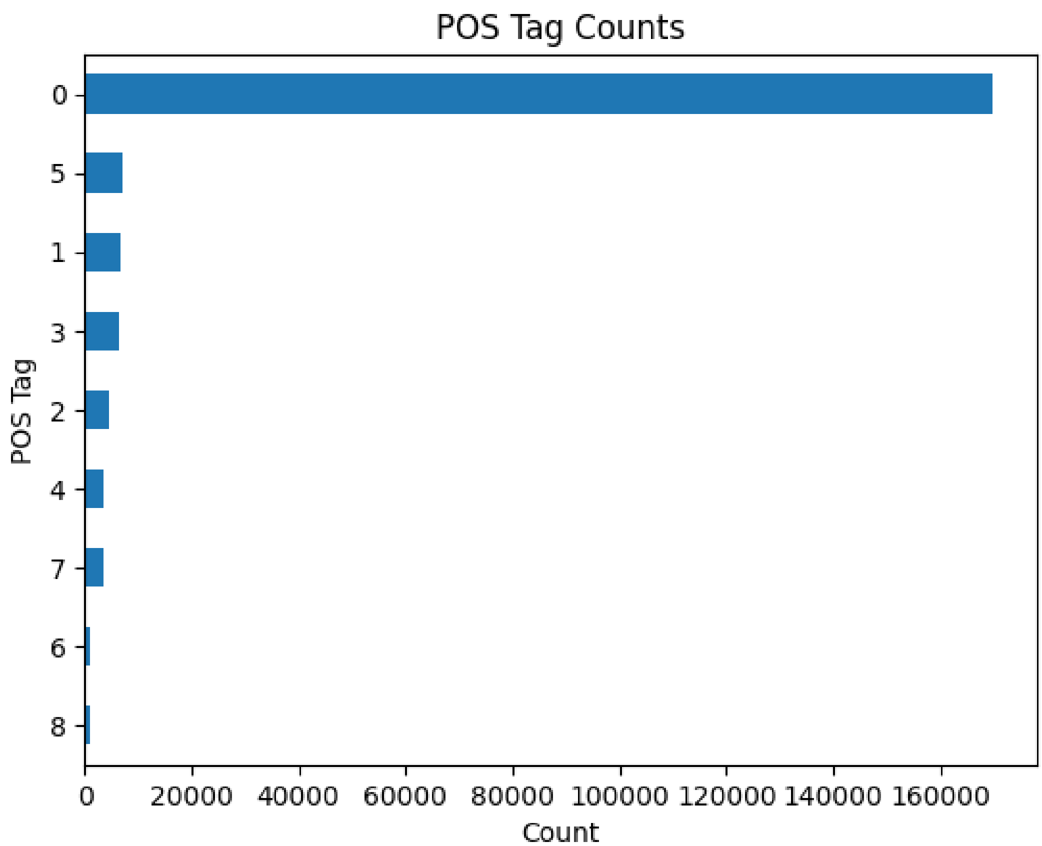
3.6.2 Visualize distribution of class labels

It is important to undetrstand the distribution of the class labels to check if there is any imbalance among the categories.

[41]:
```
# check distribution of class labels in the dataset
# Flatten the lists in df['tags']
all_tags = [tag for sublist in df['ner_tags'] for tag in sublist]
```

```python
# Convert to a pandas Series and count occurrences of each tag
tag_counts = pd.Series(all_tags).value_counts(ascending=True)

# Plot the counts
tag_counts.plot.barh()
plt.xlabel('Count')
plt.ylabel('POS Tag')
plt.title('POS Tag Counts')
plt.show()
```



Conclusions:

3.6.3 Check length of the reviews

[42]:
```python
# Calculate words per review
df['words_per_sentence'] = df['tokens'].apply(len)
```

[43]:
```python
df.head()
```

[43]:

```
   id                                             tokens  \
0  0  [EU, rejects, German, call, to, boycott, Briti…
1  1                                [Peter, Blackburn]
2  2                            [BRUSSELS, 1996-08-22]
3  3  [The, European, Commission, said, on, Thursday…
4  4  [Germany, 's, representative, to, the, Europea…

                                           pos_tags  \
0                [22, 42, 16, 21, 35, 37, 16, 21, 7]
1                                           [22, 22]
2                                           [22, 11]
3  [12, 22, 22, 38, 15, 22, 28, 38, 15, 16, 21, 3…
4  [22, 27, 21, 35, 12, 22, 22, 27, 16, 21, 22, 2…

                                         chunk_tags  \
0                [11, 21, 11, 12, 21, 22, 11, 12, 0]
1                                           [11, 12]
2                                           [11, 12]
3  [11, 12, 12, 21, 13, 11, 11, 21, 13, 11, 12, 1…
4  [11, 11, 12, 13, 11, 12, 12, 11, 12, 12, 12, 1…

                                           ner_tags  words_per_sentence
0                        [3, 0, 7, 0, 0, 0, 7, 0, 0]                   9
1                                             [1, 2]                   2
2                                             [5, 0]                   2
3  [0, 3, 4, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, …                  30
4  [5, 0, 0, 0, 0, 3, 4, 0, 0, 0, 1, 2, 0, 0, 0, …                  31
```

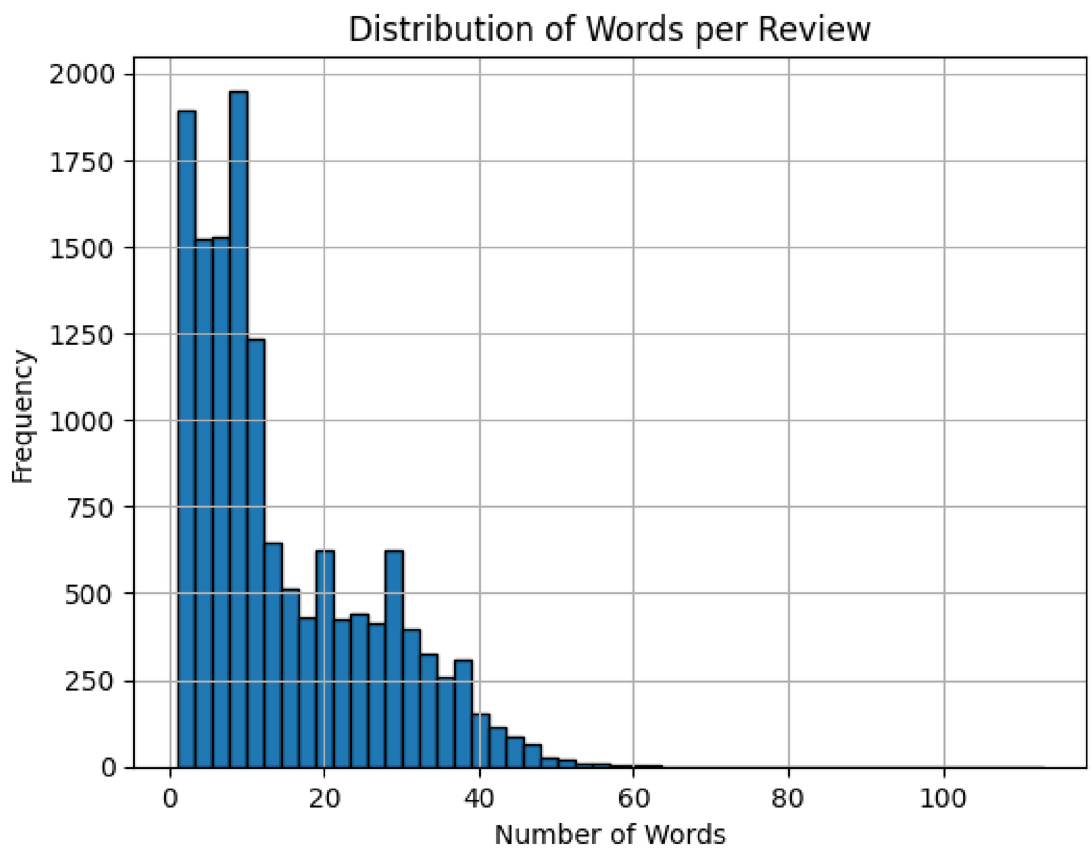
Plot the distribution of review length

[44]:

```python
# Plot a histogram of the 'words_per_review' column
df['words_per_sentence'].hist(bins=50, edgecolor='black')

# Adding labels and a title for clarity
plt.xlabel('Number of Words')
plt.ylabel('Frequency')
plt.title('Distribution of Words per Review')

# Display the plot
plt.show()
```



[45]:
```
# The model we are going to use has token (subwords) limit of 512.
# Let us check how many reviews has more than 500 words

count = (df['words_per_sentence'] > 500).sum()
print(f"Number of reviews with more than 400 words: {count}")
```

```
Number of reviews with more than 400 words: 0
```

[46]:
```
# count the rows that do not have any text
count = (df['words_per_sentence'] ==0).sum()
print(f"Number of reviews with no text words: {count}")
```

```
Number of reviews with no text words: 0
```

[47]:
```
# check the rows that have one word
count = (df['words_per_sentence'] <2).sum()
print(f"Number of reviews with less than 1 word: {count}")
```

```
Number of reviews with less than 1 word: 179
```

[48]:
```
df[df['words_per_sentence'] <2]
```

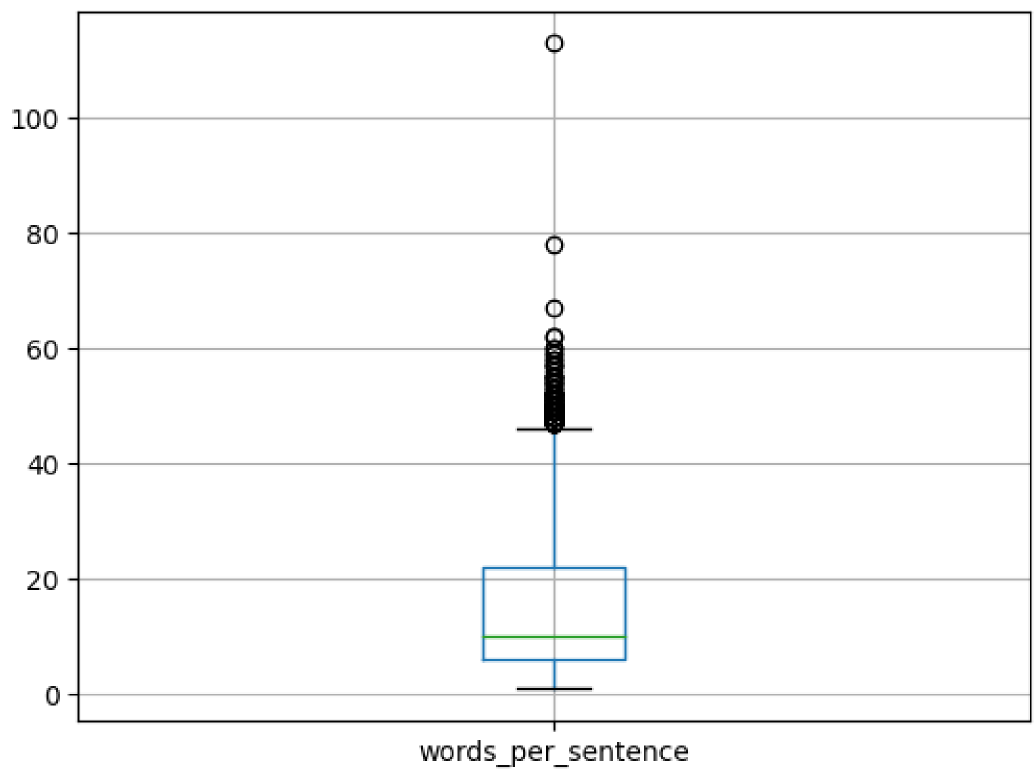
[48]:

	id	tokens	pos_tags	chunk_tags	ner_tags	words_per_sentence
11	11	[.]	[7]	[0]	[0]	1
200	200	[THAWRA]	[38]	[11]	[3]	1
203	203	[IRAQ]	[21]	[11]	[5]	1
209	209	[AN-NAHAR]	[22]	[11]	[3]	1
212	212	[AS-SAFIR]	[22]	[11]	[3]	1
…	…	…	…	…	…	…
13751	13751	[Jul-18.Jul]	[21]	[11]	[0]	1
13855	13855	[GDP]	[22]	[11]	[0]	1
13879	13879	[ABC]	[22]	[11]	[3]	1
13883	13883	[EXPANSION]	[21]	[11]	[3]	1
13907	13907	[*]	[34]	[0]	[0]	1

[179 rows x 6 columns]

[49]:
```
# distribution of number of words for each class label
df.boxplot('words_per_sentence')
```

[49]: <Axes: >



3.6.4 Reset dataset format

[50]:
```
# reset the format back to huggingface dataset
conll_dataset.reset_format()
```

[51]:
```
conll_dataset
```

[51]:
```
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
    })
})
```

[52]:
```
conll_dataset['train']['tokens'][13907]
```

[52]:
```
['*']
```

4 Data Pre-processing

4.0.1 Create small subset for experimentation

[53]:
```
train_split_small = conll_dataset['train'].shuffle(seed=42).select(range(5000))
val_split_small = conll_dataset['validation'].shuffle(seed=42).
 ↪select(range(1000))
test_split_small = conll_dataset['test'].shuffle(seed=42).select(range(1000))
```

[54]:
```
# combine train, val splits into one dataset
train_val_subset = DatasetDict({'train': train_split_small, 'val':␣
 ↪val_split_small})

# create test dataset from test split
test_subset= DatasetDict({'test': test_split_small})
```

[55]:
```
train_val_subset
```

[55]:
```
DatasetDict({
    train: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'],
```

```
        num_rows: 5000
    })
    val: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'],
        num_rows: 1000
    })
})
```

[56]:
```python
test_subset
```

[56]:
```
DatasetDict({
    test: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'],
        num_rows: 1000
    })
})
```

4.0.2 Map Targets to integers

[57]:
```python
class_names = conll_dataset['train'].features['ner_tags'].feature.names
```

[58]:
```python
class_names
```

[58]:
```
['O', 'B-PER', 'I-PER', 'B-ORG', 'I-ORG', 'B-LOC', 'I-LOC', 'B-MISC', 'I-MISC']
```

[59]:
```python
id2label = {}
for id_, label_ in enumerate(class_names):
    id2label[str(id_)] = label_
id2label
```

[59]:
```
{'0': 'O',
 '1': 'B-PER',
 '2': 'I-PER',
 '3': 'B-ORG',
 '4': 'I-ORG',
 '5': 'B-LOC',
 '6': 'I-LOC',
 '7': 'B-MISC',
 '8': 'I-MISC'}
```

[60]:
```python
label2id = {}
for id_, label_ in enumerate(class_names):
    label2id[label_] = id_
label2id
```

[60]:
```
{'O': 0,
 'B-PER': 1,
```

```
 'I-PER': 2,
 'B-ORG': 3,
 'I-ORG': 4,
 'B-LOC': 5,
 'I-LOC': 6,
 'B-MISC': 7,
 'I-MISC': 8}
```

4.1 Tokenization

[61]:
```
# Define a checkpoint for the DistilBERT model with an uncased vocabulary.
# Instantiate the tokenizer for this model using the specified checkpoint.
checkpoint = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
```

```
Downloading (…)okenizer_config.json:   0%|          | 0.00/28.0 [00:00<?, ?B/s]

Downloading (…)lve/main/config.json:   0%|          | 0.00/483 [00:00<?, ?B/s]

Downloading (…)solve/main/vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

Downloading (…)/main/tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]
```

4.1.1 Understanding pre-trained Tokenizer

- We also need to keep track of word ids so that we can align labels with tokens

[62]:
```
idx = 0
encoded_text = tokenizer(train_val_subset['train']['tokens'][idx],␣
↪is_split_into_words=True) ####### NEW ############
```

[63]:
```
encoded_text
```

[63]:
```
{'input_ids': [101, 1000, 4445, 1996, 2120, 21633, 1006, 13157, 1007, 4496,
1996, 13009, 15048, 2000, 22590, 2019, 2137, 6926, 1010, 1000, 2002, 6626, 1010,
1999, 2019, 20658, 4431, 2000, 2010, 4469, 20562, 2000, 2762, 2013, 5842, 1012,
1000, 102], '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]}
```

[64]:
```
tokens = tokenizer.convert_ids_to_tokens(encoded_text.input_ids)
print_wrap(tokens, 80)
```

```
[CLS] " neither the national socialists ( nazis ) nor the communists dared to
kidnap an american citizen , " he shouted , in an oblique reference to his extra
##dition to germany from denmark . " [SEP]
```

[65]:
```
print_wrap(tokenizer.convert_tokens_to_string(tokens),80)
```

```
[CLS] " neither the national socialists ( nazis ) nor the communists dared to
kidnap an american citizen, " he shouted, in an oblique reference to his
extradition to germany from denmark. " [SEP]
```

[66]:
```
print_wrap(train_val_subset['train']['tokens'][idx], 80)
```

```
" Neither the National Socialists ( Nazis ) nor the communists dared to kidnap
an American citizen , " he shouted , in an oblique reference to his extradition
to Germany from Denmark . "
```

[67]:
```
print_wrap(encoded_text.word_ids(), 80)
```

```
None 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
28 28 29 30 31 32 33 34 None
```

4.1.2 Create function for Tokenizer

[68]:
```
def align_targets(labels, word_ids):

    aligned_labels = []

    previous_word_id = None

    b2i = {1:2, 3:4, 5:6, 7:8}

    for w in word_ids:

        if w is None:

            label = -100
        elif w != previous_word_id:
            label = labels[w]

        else:
            label = labels[w]

            if label in b2i:
                label = b2i[label]


        aligned_labels.append(label)

        previous_word_id = w

    return aligned_labels
```

```
# CODE HERE
```

[69]:
```
# check the function
word_ids = encoded_text.word_ids()
labels = train_val_subset['train']['ner_tags'][idx]
aligned_labels = align_targets(labels, word_ids)
print('word_ids', word_ids[0:15])
print('labels:',labels[0:15])
print('aligned_labels',aligned_labels[0:15])
print(len(word_ids))
print(len(labels))
print(len(aligned_labels))
```

```
word_ids [None, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
labels: [0, 0, 0, 7, 8, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0]
aligned_labels [-100, 0, 0, 0, 7, 8, 0, 7, 0, 0, 0, 0, 0, 0, 0]
38
35
38
```

[70]:
```
# check the function
word_ids = encoded_text.word_ids()
labels = train_val_subset['train']['ner_tags'][idx]
aligned_labels = align_targets(labels, word_ids)
print('word_ids', word_ids[0:15])
print('labels:',labels[0:15])
print('aligned_labels',aligned_labels[0:15])
print(len(word_ids))
print(len(labels))
print(len(aligned_labels))
```

```
word_ids [None, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
labels: [0, 0, 0, 7, 8, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0]
aligned_labels [-100, 0, 0, 0, 7, 8, 0, 7, 0, 0, 0, 0, 0, 0, 0]
38
35
38
```

[71]:
```
def tokenize_fn(batch):
    """
    Tokenizes a batch of sequences and aligns the target labels with the␣
↪tokenized outputs.

    Args:
    - batch (dict): A dictionary containing:
        * 'tokens': A list of lists where each inner list contains tokens of a␣
↪sequence.
```

```
        * 'tags': A list of lists where each inner list contains POS tags␣
↪corresponding to the 'tokens'.

    Returns:
    - dict: A dictionary containing tokenized inputs and their corresponding␣
↪aligned labels.
    """

    # Tokenize the 'tokens' from the batch. This returns various fields like␣
↪'input_ids', 'attention_mask', etc.
    # 'is_split_into_words=True' indicates the input is already tokenized into␣
↪words.
    # 'truncation=True' ensures sequences longer than the model's max length␣
↪are truncated.
    tokenized_inputs = tokenizer(batch['tokens'], truncation=True,␣
↪is_split_into_words=True)

    # Extract the original labels/tags from the batch.
    labels_batch = batch['ner_tags']

    # This list will store the labels aligned with the tokenized input.
    aligned_labels_batch = []

    # Iterate over each example in the batch.
    for i, labels in enumerate(labels_batch):
        # Obtain the word IDs for the tokenized example. This helps in aligning␣
↪the original labels with the tokens.
        word_ids = tokenized_inputs.word_ids(i)

        # Align the original labels with the tokenized example and append to␣
↪the aligned_labels_batch list.
        aligned_labels_batch.append(align_targets(labels, word_ids))

    # The HuggingFace trainer expects the labels for token classification tasks␣
↪to be under the key 'labels'.
    # Store the aligned labels in the 'labels' key of the tokenized_inputs␣
↪dictionary.
    tokenized_inputs['labels'] = aligned_labels_batch

    return tokenized_inputs
```

4.1.3 Use map function to apply tokenization to all splits

[72]:
```
train_val_subset
```

[72]:
```
DatasetDict({
    train: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'],
        num_rows: 5000
    })
    val: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'],
        num_rows: 1000
    })
})
```

[73]:
```
# Map the tokenize_fn function over the entire train_val_subset dataset in
↪batches.
# This will tokenize the text data in each batch and return a new dataset with
↪tokenized data.
tokenized_dataset = train_val_subset.map(tokenize_fn, batched=True)
```

```
Map:   0%|          | 0/5000 [00:00<?, ? examples/s]

Map:   0%|          | 0/1000 [00:00<?, ? examples/s]
```

[74]:
```
tokenized_dataset
```

[74]:
```
DatasetDict({
    train: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags',
'input_ids', 'attention_mask', 'labels'],
        num_rows: 5000
    })
    val: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags',
'input_ids', 'attention_mask', 'labels'],
        num_rows: 1000
    })
})
```

[75]:
```
tokenized_dataset['train'].features
```

[75]:
```
{'id': Value(dtype='string', id=None),
 'tokens': Sequence(feature=Value(dtype='string', id=None), length=-1, id=None),
 'pos_tags': Sequence(feature=ClassLabel(names=['"', "''", '#', '$', '(', ')',
',', '.', ':', '``', 'CC', 'CD', 'DT', 'EX', 'FW', 'IN', 'JJ', 'JJR', 'JJS',
'LS', 'MD', 'NN', 'NNP', 'NNPS', 'NNS', 'NN|SYM', 'PDT', 'POS', 'PRP', 'PRP$',
'RB', 'RBR', 'RBS', 'RP', 'SYM', 'TO', 'UH', 'VB', 'VBD', 'VBG', 'VBN', 'VBP',
'VBZ', 'WDT', 'WP', 'WP$', 'WRB'], id=None), length=-1, id=None),
 'chunk_tags': Sequence(feature=ClassLabel(names=['O', 'B-ADJP', 'I-ADJP',
'B-ADVP', 'I-ADVP', 'B-CONJP', 'I-CONJP', 'B-INTJ', 'I-INTJ', 'B-LST', 'I-LST',
'B-NP', 'I-NP', 'B-PP', 'I-PP', 'B-PRT', 'I-PRT', 'B-SBAR', 'I-SBAR', 'B-UCP',
```

```
'I-UCP', 'B-VP', 'I-VP'], id=None), length=-1, id=None),
 'ner_tags': Sequence(feature=ClassLabel(names=['O', 'B-PER', 'I-PER', 'B-ORG',
'I-ORG', 'B-LOC', 'I-LOC', 'B-MISC', 'I-MISC'], id=None), length=-1, id=None),
 'input_ids': Sequence(feature=Value(dtype='int32', id=None), length=-1,
id=None),
 'attention_mask': Sequence(feature=Value(dtype='int8', id=None), length=-1,
id=None),
 'labels': Sequence(feature=Value(dtype='int64', id=None), length=-1, id=None)}
```

We can see that tokenization step has added three new columns ('input_ids', 'token_type_ids', 'attention_mask') to the dataset

[76]:
```
tokenized_dataset = tokenized_dataset.remove_columns(['tokens', 'ner_tags',
 ↪'pos_tags','chunk_tags', 'id'])
```

[77]:
```
tokenized_dataset.set_format(type='torch')
```

[78]:
```
tokenized_dataset
```

[78]:
```
DatasetDict({
    train: Dataset({
        features: ['input_ids', 'attention_mask', 'labels'],
        num_rows: 5000
    })
    val: Dataset({
        features: ['input_ids', 'attention_mask', 'labels'],
        num_rows: 1000
    })
})
```

[79]:
```
tokenized_dataset['train'].features
```

[79]:
```
{'input_ids': Sequence(feature=Value(dtype='int32', id=None), length=-1,
id=None),
 'attention_mask': Sequence(feature=Value(dtype='int8', id=None), length=-1,
id=None),
 'labels': Sequence(feature=Value(dtype='int64', id=None), length=-1, id=None)}
```

[80]:
```
print(len(tokenized_dataset["train"]["input_ids"][2]))
print(len(tokenized_dataset["train"]["input_ids"][1]))
```

```
12
8
```

The varying lengths in the dataset indicate that padding has not been applied yet. Instead of padding the entire dataset, we prefer processing small batches during training. Padding is done selectively for each batch based on the maximum length in the batch. We will discuss this in more detail in a later section of this notebook.

5 Model Training

5.1 Model Config File

5.1.1 Download config file of pre-trained Model

[81]:
```
# Load the configuration associated with the specified checkpoint (e.g.,␣
↪DistilBERT model configuration).
# This configuration contains details about the model architecture and settings.
# use Autoconfig class
config = AutoConfig.from_pretrained(checkpoint)
```

[82]:
```
config
```

[82]:
```
DistilBertConfig {
  "_name_or_path": "distilbert-base-uncased",
  "activation": "gelu",
  "architectures": [
    "DistilBertForMaskedLM"
  ],
  "attention_dropout": 0.1,
  "dim": 768,
  "dropout": 0.1,
  "hidden_dim": 3072,
  "initializer_range": 0.02,
  "max_position_embeddings": 512,
  "model_type": "distilbert",
  "n_heads": 12,
  "n_layers": 6,
  "pad_token_id": 0,
  "qa_dropout": 0.1,
  "seq_classif_dropout": 0.2,
  "sinusoidal_pos_embds": false,
  "tie_weights_": true,
  "transformers_version": "4.34.1",
  "vocab_size": 30522
}
```

5.1.2 Modify Configuration File

- We need to modify configuration fie to add ids to label and label to ids mapping
- Adding id2label and label2id to the configuration file provides a consistent, interpretable, and user-friendly way to handle model outputs.

[83]:
```
config.id2label = id2label
config.label2id = label2id
```

[84]:
```
config
```

[84]:

```
DistilBertConfig {
  "_name_or_path": "distilbert-base-uncased",
  "activation": "gelu",
  "architectures": [
    "DistilBertForMaskedLM"
  ],
  "attention_dropout": 0.1,
  "dim": 768,
  "dropout": 0.1,
  "hidden_dim": 3072,
  "id2label": {
    "0": "O",
    "1": "B-PER",
    "2": "I-PER",
    "3": "B-ORG",
    "4": "I-ORG",
    "5": "B-LOC",
    "6": "I-LOC",
    "7": "B-MISC",
    "8": "I-MISC"
  },
  "initializer_range": 0.02,
  "label2id": {
    "B-LOC": 5,
    "B-MISC": 7,
    "B-ORG": 3,
    "B-PER": 1,
    "I-LOC": 6,
    "I-MISC": 8,
    "I-ORG": 4,
    "I-PER": 2,
    "O": 0
  },
  "max_position_embeddings": 512,
  "model_type": "distilbert",
  "n_heads": 12,
  "n_layers": 6,
  "pad_token_id": 0,
  "qa_dropout": 0.1,
  "seq_classif_dropout": 0.2,
  "sinusoidal_pos_embds": false,
  "tie_weights_": true,
  "transformers_version": "4.34.1",
  "vocab_size": 30522
}
```

5.2 Download pre-trained model

[85]:
```
# Instantiate a model for sequence classification using the specified␣
↪checkpoint.
# The provided configuration (config) ensures the model aligns with the␣
↪structure and settings of the original checkpoint.
# Use AutoModelForSequenceClassification
# Pass the checkpoint and config
model = AutoModelForTokenClassification.from_pretrained(checkpoint,␣
↪config=config)
```

```
Downloading model.safetensors:   0%|          | 0.00/268M [00:00<?, ?B/s]
```

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

[86]:
```
model.config
```

[86]:
```
DistilBertConfig {
  "_name_or_path": "distilbert-base-uncased",
  "activation": "gelu",
  "architectures": [
    "DistilBertForMaskedLM"
  ],
  "attention_dropout": 0.1,
  "dim": 768,
  "dropout": 0.1,
  "hidden_dim": 3072,
  "id2label": {
    "0": "O",
    "1": "B-PER",
    "2": "I-PER",
    "3": "B-ORG",
    "4": "I-ORG",
    "5": "B-LOC",
    "6": "I-LOC",
    "7": "B-MISC",
    "8": "I-MISC"
  },
  "initializer_range": 0.02,
  "label2id": {
    "B-LOC": 5,
    "B-MISC": 7,
    "B-ORG": 3,
    "B-PER": 1,
```

```
    "I-LOC": 6,
    "I-MISC": 8,
    "I-ORG": 4,
    "I-PER": 2,
    "O": 0
  },
  "max_position_embeddings": 512,
  "model_type": "distilbert",
  "n_heads": 12,
  "n_layers": 6,
  "pad_token_id": 0,
  "qa_dropout": 0.1,
  "seq_classif_dropout": 0.2,
  "sinusoidal_pos_embds": false,
  "tie_weights_": true,
  "transformers_version": "4.34.1",
  "vocab_size": 30522
}
```

5.3 Model Input/Collate Function

[87]:
```
data_collator = DataCollatorForTokenClassification(tokenizer = tokenizer,
                                                   padding=True,
                                                   label_pad_token_id=-100,
                                                   return_tensors='pt') # CODE␣
 ↪HERE
```

[88]:
```
features = [tokenized_dataset["train"][i] for i in range(2)]
```

[89]:
```
features
```

[89]:
```
[{'input_ids': tensor([  101,  1000,  4445,  1996,  2120, 21633,  1006, 13157,
1007,  4496,
          1996, 13009, 15048,  2000, 22590,  2019,  2137,  6926,  1010,  1000,
          2002,  6626,  1010,  1999,  2019, 20658,  4431,  2000,  2010,  4469,
         20562,  2000,  2762,  2013,  5842,  1012,  1000,   102]),
  'attention_mask': tensor([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': tensor([-100,    0,    0,    0,    7,    8,    0,    7,    0,    0,
0,    0,
             0,    0,    0,    0,    7,    0,    0,    0,    0,    0,    0,
0,
             0,    0,    0,    0,    0,    0,    0,    0,    5,    0,    5,
0,
             0, -100])},
```

```
 {'input_ids': tensor([  101, 25317,  2727,  1011,  5511,  1011,  2570,   102]),
  'attention_mask': tensor([1, 1, 1, 1, 1, 1, 1, 1]),
  'labels': tensor([-100,    5,    0,    0,    0,    0,    0, -100])}]
```

[90]:
```
model_input = data_collator(features)
model_input.keys()
```

```
You're using a DistilBertTokenizerFast tokenizer. Please note that with a fast
tokenizer, using the `__call__` method is faster than using a method to encode
the text followed by a call to the `pad` method to get a padded encoding.
```

[90]:
```
dict_keys(['input_ids', 'attention_mask', 'labels'])
```

[91]:
```
print(model_input.input_ids[0][0:10])
print(model_input.input_ids[0][-20:])
print(model_input.input_ids[1][0:10])
print(model_input.input_ids[1][-20:])
```

```
tensor([  101,  1000,  4445,  1996,  2120, 21633,  1006, 13157,  1007,  4496])
tensor([ 1010,  1000,  2002,  6626,  1010,  1999,  2019, 20658,  4431,  2000,
         2010,  4469, 20562,  2000,  2762,  2013,  5842,  1012,  1000,   102])
tensor([  101, 25317,  2727,  1011,  5511,  1011,  2570,   102,     0,     0])
tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

[92]:
```
print(model_input.attention_mask[0][-20:])
print(model_input.attention_mask[1][-20:])
```

```
tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

[93]:
```
print(tokenizer.convert_ids_to_tokens(model_input.input_ids[0][0:10]))
```

```
['[CLS]', '"', 'neither', 'the', 'national', 'socialists', '(', 'nazis', ')',
'nor']
```

[94]:
```
print(tokenizer.convert_ids_to_tokens(model_input.input_ids[0][-10:]))
```

```
['his', 'extra', '##dition', 'to', 'germany', 'from', 'denmark', '.', '"',
'[SEP]']
```

[95]:
```
print(tokenizer.convert_ids_to_tokens(model_input.input_ids[1][0:10]))
```

```
['[CLS]', 'tunis', '1996', '-', '08', '-', '22', '[SEP]', '[PAD]', '[PAD]']
```

[96]:
```
print(tokenizer.convert_ids_to_tokens(model_input.input_ids[1][-10:]))
```

```
['[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]',
'[PAD]', '[PAD]']
```

5.4 Understanding Model Output

[97]:
```
# model output
model=model.to(device=0)
model_input= model_input.to(device=0)
```

[98]:
```
model_input['labels'].shape
```

[98]:
```
torch.Size([2, 38])
```

[99]:
```
model_input['input_ids'].shape
```

[99]:
```
torch.Size([2, 38])
```

[100]:
```
model.train()
model_output = model(**model_input)
```

[101]:
```
# keys in model output
model_output.keys()
```

[101]:
```
odict_keys(['loss', 'logits'])
```

[102]:
```
# let us look at logits
model_output.logits
```

[102]:
```
tensor([[[-2.4308e-01, -1.1708e-01, -2.1971e-01,  1.6307e-01,  1.4601e-01,
           2.0145e-01, -2.4227e-01,  9.8193e-02, -1.7242e-01],
         [ 1.5708e-01, -4.1788e-01, -3.2049e-01, -2.4070e-01,  3.1934e-01,
          -2.4135e-01, -8.6391e-02, -3.3494e-01,  3.6294e-01],
         [ 2.7314e-01,  2.8903e-01,  7.6480e-02, -2.5455e-01,  7.5137e-02,
           1.9715e-01,  9.4742e-02,  1.8268e-01,  9.6559e-02],
         [-3.0259e-01,  4.6464e-02, -1.6077e-01, -1.9405e-01,  5.1598e-01,
           3.0119e-01, -5.3167e-01,  2.8823e-01,  7.4973e-02],
         [-1.9633e-01,  2.8106e-01,  3.9790e-02, -1.0849e-01, -1.2074e-01,
           1.9476e-01,  2.4687e-01,  2.0661e-01,  2.9793e-01],
         [-2.9915e-01,  1.2213e-01, -1.2629e-01,  9.0612e-02,  6.0007e-02,
           2.7508e-01,  1.3689e-01,  1.6719e-01, -1.8419e-02],
         [-1.4729e-01,  3.6326e-01,  1.3238e-01, -1.5370e-01,  3.1487e-01,
           2.0215e-01, -4.4245e-02,  7.4063e-02,  4.3045e-02],
         [ 2.3387e-02,  2.5816e-01,  9.7993e-02, -1.0122e-01,  2.8138e-01,
           4.2406e-01, -2.7684e-01,  1.1888e-01,  6.0005e-02],
         [-3.6054e-02,  1.2859e-01,  1.2520e-01,  2.6222e-02,  2.9395e-01,
           3.8932e-01,  9.5785e-02,  4.7880e-01,  1.7103e-01],
         [ 2.1900e-01,  1.6538e-01, -8.2698e-02, -4.4440e-01, -3.0839e-02,
           2.5328e-01, -1.3919e-01,  1.7866e-01, -1.0481e-01],
         [-2.6228e-01, -4.5818e-02, -1.9338e-01, -1.6800e-01,  5.0525e-01,
           5.7025e-01, -3.2305e-01,  2.2169e-01, -7.3414e-02],
```

```
[-4.1632e-02,  2.4055e-01,  1.6474e-01, -2.8791e-01,  1.2808e-01,
  4.6229e-01,  3.6829e-02,  1.2428e-01,  1.0949e-02],
[ 2.1165e-01,  2.6506e-01,  1.2149e-01, -2.3629e-01, -7.0973e-02,
  3.1031e-01, -1.8414e-01,  3.7809e-01,  3.0266e-01],
[-2.2368e-01, -4.2080e-02,  3.6031e-02, -1.2157e-01,  1.4863e-01,
  8.4245e-02, -4.5533e-01,  4.4793e-01,  8.1735e-02],
[ 1.3723e-01,  8.7642e-02,  1.1060e-01, -1.6464e-01,  2.1153e-01,
  2.4869e-01, -2.2810e-01,  3.3577e-01, -1.1523e-01],
[ 1.6426e-02, -1.2406e-01,  7.3887e-02, -5.2098e-03,  2.6672e-01,
  9.7872e-02, -3.9534e-01,  2.1387e-01,  1.6361e-01],
[ 1.1369e-01,  1.2118e-01,  2.1789e-01, -1.9039e-01,  2.3140e-01,
 -5.3362e-02, -7.4113e-02,  2.2404e-01, -2.1638e-01],
[-5.6112e-02,  1.0001e-01,  1.8240e-01,  5.2237e-02,  3.8803e-01,
  7.5641e-02, -8.4060e-02,  1.6293e-01,  4.9080e-02],
[-4.4642e-02,  3.9294e-02,  6.1811e-02, -9.8542e-02, -4.8278e-02,
  2.4929e-01, -2.1913e-01,  3.4126e-02,  1.1004e-01],
[ 1.3079e-01, -4.0225e-02,  1.2678e-01, -1.4156e-01,  1.4078e-01,
 -7.9843e-02, -1.4238e-01, -5.8185e-02,  1.8576e-03],
[ 1.5961e-02,  1.4755e-01,  1.2745e-01,  7.5197e-02,  4.1568e-01,
  2.9661e-01, -9.2544e-02, -3.6067e-02, -8.3766e-02],
[ 2.7828e-01, -1.5819e-01, -6.2081e-03, -7.5443e-02,  2.0539e-01,
 -1.5721e-01, -3.4885e-01,  5.2231e-02, -1.0361e-01],
[-1.3047e-01,  1.8934e-01,  4.0262e-02, -3.8933e-01, -2.4939e-01,
  3.9941e-01, -4.1721e-01,  2.0205e-02,  8.0781e-02],
[ 1.7202e-01,  6.5326e-02,  1.1557e-01, -9.8234e-02,  2.5159e-01,
  1.7834e-01, -2.3793e-01,  1.7681e-01, -1.9534e-01],
[ 3.9090e-01,  5.3971e-02,  1.6221e-01, -1.4980e-01,  2.1268e-01,
  1.7880e-01, -2.2931e-01, -1.7563e-02, -6.4062e-02],
[ 2.8860e-01,  3.1061e-01,  3.7685e-02,  4.6254e-02,  6.9564e-02,
  3.1272e-01, -3.9299e-01,  8.0721e-02,  2.1920e-02],
[ 1.4067e-01, -3.0568e-02,  2.8054e-02, -3.4065e-01,  1.0300e-01,
  1.2178e-01,  2.7892e-02, -4.5932e-02,  1.4710e-01],
[ 1.6985e-01,  2.5830e-01,  1.4991e-02, -9.5868e-02, -4.2107e-02,
  2.6054e-01,  3.0279e-02,  2.1132e-01, -2.2884e-02],
[-1.1382e-02,  9.1637e-02,  2.2954e-01, -3.1012e-02,  2.5852e-01,
  2.9295e-01, -1.5226e-01,  3.5388e-01,  9.8706e-02],
[ 7.1794e-02,  2.5115e-01,  1.1930e-01, -1.0353e-01,  3.6960e-01,
  2.4954e-01, -1.9440e-01,  1.7964e-01,  2.1950e-01],
[ 3.6519e-01,  1.8696e-01,  7.2636e-02, -2.7964e-01,  2.0214e-01,
  3.9132e-02,  7.2714e-03, -5.6606e-03,  2.1404e-01],
[ 1.8553e-01,  4.0193e-01,  1.2591e-01, -1.5896e-01,  8.8783e-02,
  1.2854e-01, -6.4918e-02,  9.4355e-02, -7.0484e-03],
[-3.8935e-02,  1.5647e-01,  2.3925e-01, -1.1883e-01,  2.2330e-01,
  3.8848e-01, -1.1111e-01,  2.7577e-01, -1.7336e-02],
[ 3.0163e-01,  4.3054e-02, -2.1095e-02, -8.9814e-02,  2.1151e-01,
  1.4489e-01,  9.8293e-02,  2.5365e-01,  9.9316e-03],
[-1.6944e-02,  2.7420e-01,  3.1073e-01, -1.1678e-01, -7.8724e-03,
```

```
  2.3978e-01,  1.8836e-02,  3.2012e-01, -6.0005e-02],
[ 6.8423e-01,  1.8781e-01, -7.9735e-02,  8.9320e-02,  3.3255e-01,
  6.2550e-01,  2.3859e-01,  3.7777e-01,  1.9251e-01],
[-5.4162e-02, -2.7453e-01, -1.7088e-01, -3.6186e-02,  1.4387e-01,
 -1.5467e-01, -2.3389e-01,  5.6318e-02, -1.2408e-01],
[ 3.6487e-02,  1.3104e-01, -3.9452e-01, -1.2927e-01,  7.7670e-02,
 -3.0397e-01, -1.2955e-01,  3.7063e-01,  2.2609e-01]],

[[-1.3207e-01,  4.6425e-02, -2.8454e-01,  1.7875e-01,  1.6414e-02,
 -1.2625e-03, -4.7316e-02, -3.2452e-01, -1.7022e-01],
[-1.0556e-01, -8.5534e-02, -3.8193e-02, -2.0101e-01, -7.0444e-02,
  1.4243e-01,  5.4477e-02, -1.4848e-02,  4.9440e-01],
[-1.8582e-01, -1.8448e-01, -2.1163e-01, -1.9196e-02, -4.7335e-02,
 -1.0849e-02, -2.9216e-02,  9.1131e-02,  3.9124e-01],
[-1.0885e-01,  4.3732e-02,  8.0510e-02, -3.2730e-01,  2.0294e-01,
  4.5153e-01, -3.9164e-01,  2.7084e-01,  8.9653e-02],
[ 2.3762e-01, -2.8000e-02, -1.5195e-01,  1.0063e-01,  1.6624e-01,
  2.2671e-01,  6.5483e-02,  3.3695e-01,  1.4455e-01],
[-2.2072e-01,  5.1164e-02,  2.7859e-02, -3.4611e-01,  2.5394e-01,
  3.1690e-01, -3.6392e-01,  1.8091e-01,  5.6314e-02],
[ 2.9447e-01,  3.8165e-01, -2.6883e-01, -7.7644e-03,  1.8334e-01,
  2.2266e-01,  2.8640e-01,  1.7516e-01, -5.4838e-02],
[ 4.0880e-01,  3.7150e-01,  1.6931e-01,  5.5526e-02,  2.7462e-01,
  3.8976e-01,  3.0415e-01,  2.2605e-01,  4.0077e-01],
[-2.3007e-01,  3.8591e-02,  1.7244e-01, -2.5252e-01,  7.1330e-02,
  1.7972e-01,  1.2813e-01,  2.8785e-01,  3.5319e-02],
[-1.8366e-02,  1.2448e-01,  8.5812e-03,  7.2981e-02,  2.1079e-01,
  1.9973e-01,  2.0588e-02,  2.4797e-01, -4.0517e-02],
[-2.0845e-01, -9.2533e-02, -2.5692e-02, -1.4288e-01,  1.2498e-01,
 -1.0062e-01,  8.1950e-02,  3.3524e-01, -8.3826e-02],
[-3.0556e-01,  1.5009e-01, -1.2376e-02, -2.0274e-01,  9.5362e-02,
  5.3737e-02,  9.1721e-02,  1.4021e-01, -1.6583e-01],
[-2.4333e-01,  1.0944e-02,  9.0385e-02, -1.9808e-01,  7.6844e-02,
  1.3253e-01, -6.9164e-02,  2.0500e-02, -6.6059e-02],
[-2.5619e-01, -6.9965e-02, -2.7585e-02, -2.3087e-02,  1.4775e-01,
 -9.9302e-02,  8.7520e-02,  1.7375e-01, -1.4479e-03],
[-1.4015e-01,  6.2605e-02,  1.1693e-01, -2.8308e-01,  1.3393e-01,
  1.4016e-01,  6.7038e-02,  2.5033e-01,  7.6175e-05],
[-3.1132e-01, -5.8828e-02,  1.8384e-01, -1.4374e-01,  1.9261e-01,
  9.6008e-02,  6.6253e-02,  2.4847e-02, -1.5122e-01],
[-3.0243e-01,  2.2903e-02,  1.2496e-01,  4.8065e-02,  1.9275e-01,
  1.3451e-01,  8.3378e-02,  1.9400e-01, -1.5115e-01],
[-2.1367e-01,  5.1456e-02,  1.3364e-01,  1.0151e-01,  2.2910e-01,
  1.1301e-01,  4.3543e-03,  1.7627e-01,  8.4374e-03],
[-1.4606e-01,  3.6216e-02,  2.7568e-02, -7.4876e-02,  1.8247e-01,
  1.0634e-01,  4.5949e-03,  1.5272e-01, -2.1194e-02],
[-2.1683e-02,  1.5791e-01,  2.7745e-01, -1.1506e-01,  1.1444e-01,
```

```
          2.8365e-01,  8.9834e-04,  2.4249e-01, -5.3190e-03],
         [-2.4350e-01,  2.7827e-01,  7.2172e-02, -8.4349e-02,  2.0047e-01,
           1.3623e-01,  6.7657e-02,  3.2218e-01, -1.5450e-01],
         [-3.1913e-01,  2.9251e-02,  2.2870e-03, -5.5257e-02,  1.0716e-01,
          -1.1451e-01, -6.6570e-02,  5.9661e-02, -1.2439e-01],
         [-1.9809e-01,  1.9013e-01,  2.1451e-01, -3.6503e-02,  8.7015e-02,
          -6.1289e-03,  4.4296e-02,  4.1847e-02,  8.5270e-02],
         [-3.3179e-01,  8.3413e-03,  9.8252e-03,  2.8275e-02,  1.7189e-01,
           1.3507e-01,  1.8842e-01,  1.7323e-01,  6.5040e-03],
         [-2.7349e-01,  2.2010e-01,  3.4446e-02, -2.3082e-01,  1.5880e-03,
           1.9898e-01,  1.1796e-01,  9.9631e-02, -9.0609e-04],
         [-2.0019e-01,  5.4679e-02,  1.6741e-01, -1.9313e-01, -3.8146e-02,
          -1.2317e-02,  5.2626e-02, -2.0950e-02,  5.4567e-02],
         [-1.0716e-01,  1.1003e-01,  6.7365e-02, -2.6921e-01,  8.7715e-02,
           1.1837e-01, -2.2902e-02, -6.7065e-04, -1.1049e-02],
         [-1.8665e-01,  3.6356e-02,  1.9036e-02, -1.4833e-01, -5.6587e-02,
           3.5387e-02,  1.4453e-01,  8.7667e-02, -1.4935e-01],
         [-8.0747e-02, -7.9219e-02,  8.7398e-02, -1.9665e-01,  1.2146e-01,
          -1.4930e-01, -1.0379e-02,  2.2458e-01,  1.5086e-01],
         [-3.8703e-02,  6.5841e-02,  9.9638e-02, -1.0641e-01,  6.9073e-02,
           5.5848e-02,  5.0154e-02,  8.5417e-02,  1.3534e-02],
         [ 5.4132e-02,  3.0474e-01,  2.4478e-01, -1.3950e-01,  2.1393e-01,
           1.6001e-01, -1.7678e-01,  2.6255e-01,  2.3760e-02],
         [-1.5732e-02,  1.2661e-01,  2.2484e-01, -1.3949e-01,  3.8199e-02,
           1.1933e-01, -4.0743e-02,  1.3398e-01,  2.6007e-02],
         [ 5.8137e-03,  1.0904e-01,  1.4117e-01,  7.0675e-03,  7.9955e-02,
           2.5340e-01,  1.0360e-01,  1.6314e-01, -1.5926e-01],
         [-1.2364e-01,  6.3513e-02,  1.6761e-01, -1.2783e-01,  6.8255e-02,
           3.3450e-01, -2.6385e-02,  2.8739e-01, -4.5595e-02],
         [ 1.8756e-01,  6.0634e-02,  7.8142e-02,  1.8228e-02,  3.5752e-02,
           3.0865e-01,  3.4277e-02,  3.6696e-01,  1.1644e-01],
         [-2.3969e-01, -2.7868e-02,  5.7071e-02, -1.3531e-01,  8.8892e-02,
           1.3272e-01,  1.8393e-01,  1.3242e-01, -4.3792e-02],
         [-2.3416e-01,  1.3178e-02,  6.1747e-02, -1.3045e-02,  1.1218e-01,
           2.0279e-02, -1.1773e-01,  1.5341e-01,  3.4761e-03],
         [-2.1174e-01, -2.0982e-01, -4.3058e-02, -1.3064e-01,  4.7092e-02,
          -1.6100e-01,  8.4483e-03,  1.3512e-01,  5.8324e-04]]],
       device='cuda:0', grad_fn=<ViewBackward0>)
```

[103]:
```
model_output.logits.shape
```

[103]:
```
torch.Size([2, 38, 9])
```

[104]:
```
model_output.loss
```

[104]:
```
tensor(2.1762, device='cuda:0', grad_fn=<NllLossBackward0>)
```

5.5 Evaluation metric(s)

5.5.1 Function to compute metric

[105]:
```
# Understanding and checking the function
true_labels = np.array([[-100, 0, 0, 1, 2, 1,-100], [-100, 0, 2, 1, 0, 1,-100]])
logits = np.array([

    [[0.8, 0.1, 0.1],
    [0.8, 0.1, 0.1],
    [0.8, 0.1, 0.1],
    [0.1, 0.8, 0.1],
    [0.1, 0.8, 0.1],
    [0.1, 0.8, 0.1],
    [0.8, 0.1, 0.1]],

    [[0.8, 0.1, 0.1],
    [0.8, 0.1, 0.1],
    [0.8, 0.1, 0.1],
    [0.1, 0.8, 0.1],
    [0.1, 0.8, 0.1],
    [0.1, 0.8, 0.1],
    [0.1, 0.8, 0.1]]

])
```

[106]:
```
predicted_indices = np.argmax(logits, axis=-1)
predicted_indices
```

[106]:
```
array([[0, 0, 0, 1, 1, 1, 0],
       [0, 0, 0, 1, 1, 1, 1]])
```

[107]:
```
string_true_labels = [[class_names[label_id] for label_id in sequence if␣
 ↪label_id != -100] for sequence in true_labels]

# Convert predicted indices to their string representation, but only for tokens␣
 ↪where the true label isn't -100
string_predictions = [
    [class_names[pred_id] for pred_id, true_label_id in zip(pred_sequence,␣
 ↪true_sequence) if true_label_id != -100]
    for pred_sequence, true_sequence in zip(predicted_indices, true_labels)
]
```

[108]:
```
print(string_predictions)
print(string_true_labels)
```

```
[['O', 'O', 'B-PER', 'B-PER', 'B-PER'], ['O', 'O', 'B-PER', 'B-PER', 'B-PER']]
[['O', 'O', 'B-PER', 'I-PER', 'B-PER'], ['O', 'I-PER', 'B-PER', 'O', 'B-PER']]
```

[109]:
```
metric = evaluate.load("seqeval")
```

```
Downloading builder script:   0%|          | 0.00/6.34k [00:00<?, ?B/s]
```

[110]:
```
metric??
```

[111]:
```
metric.compute(predictions=string_predictions, references=string_true_labels)
```

[111]:
```
{'PER': {'precision': 0.5,
  'recall': 0.6,
  'f1': 0.5454545454545454,
  'number': 5},
 'overall_precision': 0.5,
 'overall_recall': 0.6,
 'overall_f1': 0.5454545454545454,
 'overall_accuracy': 0.7}
```

[112]:
```
seqeval_metric = evaluate.load('seqeval')
def compute_metrics(logits_and_labels):
    """
    Compute sequence tagging metrics using the seqeval metric.

    Args:
    - logits_and_labels (tuple): A tuple containing model logits and true␣
↪labels.

    Returns:
    - dict: A dictionary containing precision, recall, f1-score, and accuracy.
    """

    # Separate logits and labels from the input tuple
    logits, true_labels = logits_and_labels

    # Obtain predicted label indices by selecting the label with the highest␣
↪logit value for each token
    predicted_indices = np.argmax(logits, axis=-1)  # Shape: (batch_size,␣
↪sequence_length)

    # Convert label indices to their string representation, ignoring special␣
↪tokens (label index = -100)
    string_true_labels = [[class_names[label_id] for label_id in sequence if␣
↪label_id != -100] for sequence in true_labels]

    # Convert predicted indices to their string representation, but only for␣
↪tokens where the true label isn't -100
    string_predictions = [
```

```
        [class_names[pred_id] for pred_id, true_label_id in zip(pred_sequence,␣
↪true_sequence) if true_label_id != -100]
        for pred_sequence, true_sequence in zip(predicted_indices, true_labels)
    ]

    # Compute the metrics using seqeval
    metrics_results = seqeval_metric.compute(predictions=string_predictions,␣
↪references=string_true_labels)

    return {
        'precision': metrics_results['overall_precision'],
        'recall': metrics_results['overall_recall'],
        'f1': metrics_results['overall_f1'],
        'accuracy': metrics_results['overall_accuracy']
    }

  # CODE HERE
```

[113]:
```
compute_metrics((logits, true_labels))
```

[113]:
```
{'precision': 0.5, 'recall': 0.6, 'f1': 0.5454545454545454, 'accuracy': 0.7}
```

5.6 Set up Logger for experiments

[114]:
```
# YOU WILL NEED TO CREATE AN ACCOUNT FOR WANDB
# It may provide a link for token , copy paste the token following instructions
# setup wandb
wandb.login()  # you will need to create wandb account first
# Set project name for logging
%env WANDB_PROJECT = nlp_course_fall_2023-ner
```

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

```
wandb: Appending key for api.wandb.ai to your netrc file:
/root/.netrc
```

```
env: WANDB_PROJECT=nlp_course_fall_2023-ner
```

5.7 Hyperparameters and Checkpointing

[115]:
```
# Define the directory where model checkpoints will be saved
model_folder = base_folder / "models"/"nlp_spring_2023/ner"
# Create the directory if it doesn't exist
model_folder.mkdir(exist_ok=True, parents=True)

# Configure training parameters
training_args = TrainingArguments(
    # Training-specific configurations
```

```
    num_train_epochs=2,  # Total number of training epochs
    # Number of samples per training batch for each device
    per_device_train_batch_size=16,
    # Number of samples per evaluation batch for each device
    per_device_eval_batch_size=16,
    weight_decay=0.01,  # Apply L2 regularization to prevent overfitting
    learning_rate=2e-5,  # Step size for the optimizer during training
    optim='adamw_torch',  # Optimizer,

    # Checkpoint saving and model evaluation settings
    output_dir=str(model_folder),  # Directory to save model checkpoints
    evaluation_strategy='steps',  # Evaluate model at specified step intervals
    eval_steps=20,  # Perform evaluation every 10 training steps
    save_strategy="steps",  # Save model checkpoint at specified step intervals
    save_steps=20,  # Save a model checkpoint every 10 training steps
    load_best_model_at_end=True,  # Reload the best model at the end of training
    save_total_limit=2,  # Retain only the best and the most recent model␣
↪checkpoints
    # Use 'accuracy' as the metric to determine the best model
    metric_for_best_model="accuracy",
    greater_is_better=True,  # A model is 'better' if its accuracy is higher


    # Experiment logging configurations (commented out in this example)
    logging_strategy='steps',
    logging_steps=20,
    report_to='wandb',  # Log metrics and results to Weights & Biases platform
    run_name= 'ner_exp1',  # Experiment name for Weights & Biases
)
```

5.8 Initialize Trainer

[116]:
```
# initialize trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset["train"],
    eval_dataset=tokenized_dataset["val"],
    compute_metrics=compute_metrics,
    tokenizer=tokenizer,
    data_collator=data_collator,
)
```

5.9 Start Training

[117]:
```
trainer.data_collator
```

[117]:
```
DataCollatorForTokenClassification(tokenizer=DistilBertTokenizerFast(name_or_pat
h='distilbert-base-uncased', vocab_size=30522, model_max_length=512,
is_fast=True, padding_side='right', truncation_side='right',
special_tokens={'unk_token': '[UNK]', 'sep_token': '[SEP]', 'pad_token':
'[PAD]', 'cls_token': '[CLS]', 'mask_token': '[MASK]'},
clean_up_tokenization_spaces=True),  added_tokens_decoder={
        0: AddedToken("[PAD]", rstrip=False, lstrip=False, single_word=False,
normalized=False, special=True),
        100: AddedToken("[UNK]", rstrip=False, lstrip=False, single_word=False,
normalized=False, special=True),
        101: AddedToken("[CLS]", rstrip=False, lstrip=False, single_word=False,
normalized=False, special=True),
        102: AddedToken("[SEP]", rstrip=False, lstrip=False, single_word=False,
normalized=False, special=True),
        103: AddedToken("[MASK]", rstrip=False, lstrip=False, single_word=False,
normalized=False, special=True),
}, padding=True, max_length=None, pad_to_multiple_of=None,
label_pad_token_id=-100, return_tensors='pt')
```

[118]:
```
trainer.train()  # start training
```

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

wandb: Currently logged in as: shritej24c
(redeem_team). Use `wandb login --relogin` to force relogin

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

/usr/local/lib/python3.10/dist-packages/seqeval/metrics/v1.py:57:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/seqeval/metrics/v1.py:57:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 due to no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
/usr/local/lib/python3.10/dist-packages/seqeval/metrics/v1.py:57:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/seqeval/metrics/v1.py:57:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/seqeval/metrics/v1.py:57:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[118]:
```
TrainOutput(global_step=626, training_loss=0.20577812966066428,
metrics={'train_runtime': 210.4831, 'train_samples_per_second': 47.51,
'train_steps_per_second': 2.974, 'total_flos': 121120544570256.0, 'train_loss':
0.20577812966066428, 'epoch': 2.0})
```

5.10 Evaluation

5.10.1 Check performance on validation set

[119]:
```
# Evaluate the trained model on the tokenized validation dataset.
# This will provide metrics like loss, accuracy, etc. based on the model's␣
↪performance on the validation set.
trainer.evaluate(tokenized_dataset["val"])
```

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

[119]:
```
{'eval_loss': 0.07934200018644333,
 'eval_precision': 0.841684434968017,
 'eval_recall': 0.8801560758082497,
 'eval_f1': 0.8604904632152588,
 'eval_accuracy': 0.9776327561707279,
 'eval_runtime': 2.115,
 'eval_samples_per_second': 472.821,
 'eval_steps_per_second': 29.788,
 'epoch': 2.0}
```

5.10.2 Check Confusion Matrix

[120]:
```
# Use the trainer to generate predictions on the tokenized validation dataset.
# The resulting object, valid_output, will contain the model's logits (raw␣
↪prediction scores) for each input in the validation set.
```

```
valid_output = trainer.predict(tokenized_dataset["val"])
```

<IPython.core.display.HTML object>

[121]:
```
# Retrieve the named fields (attributes) of the valid_output object.
# This helps understand the structure of the prediction output and the
↪available information it contains.
valid_output._fields
```

[121]: ('predictions', 'label_ids', 'metrics')

[122]:
```
# Check and print the shape of the predictions and label_ids from the
↪valid_output object.
# This provides insight into the dimensions of the predicted outputs and the
↪true labels for the validation set.
print(valid_output.predictions.shape)
print(valid_output.label_ids.shape)
```

```
(1000, 146, 9)
(1000, 146)
```

[123]:
```
# Convert the logits (raw prediction scores) from the valid_output object into
↪class predictions.
# For each input, pick the class with the highest logit as the predicted class.
# Also, extract the true label IDs from valid_output and store them as an array
↪for further analysis.
valid_predictions = np.argmax(valid_output.predictions, axis=2)
valid_labels = np.array(valid_output.label_ids)

# 2. Filter out any tokens with label -100 (typically used for padding or
↪special tokens)
mask = valid_labels != -100
filtered_predictions = valid_predictions[mask]
filtered_labels = valid_labels[mask]
```

[124]:
```
class_names
```

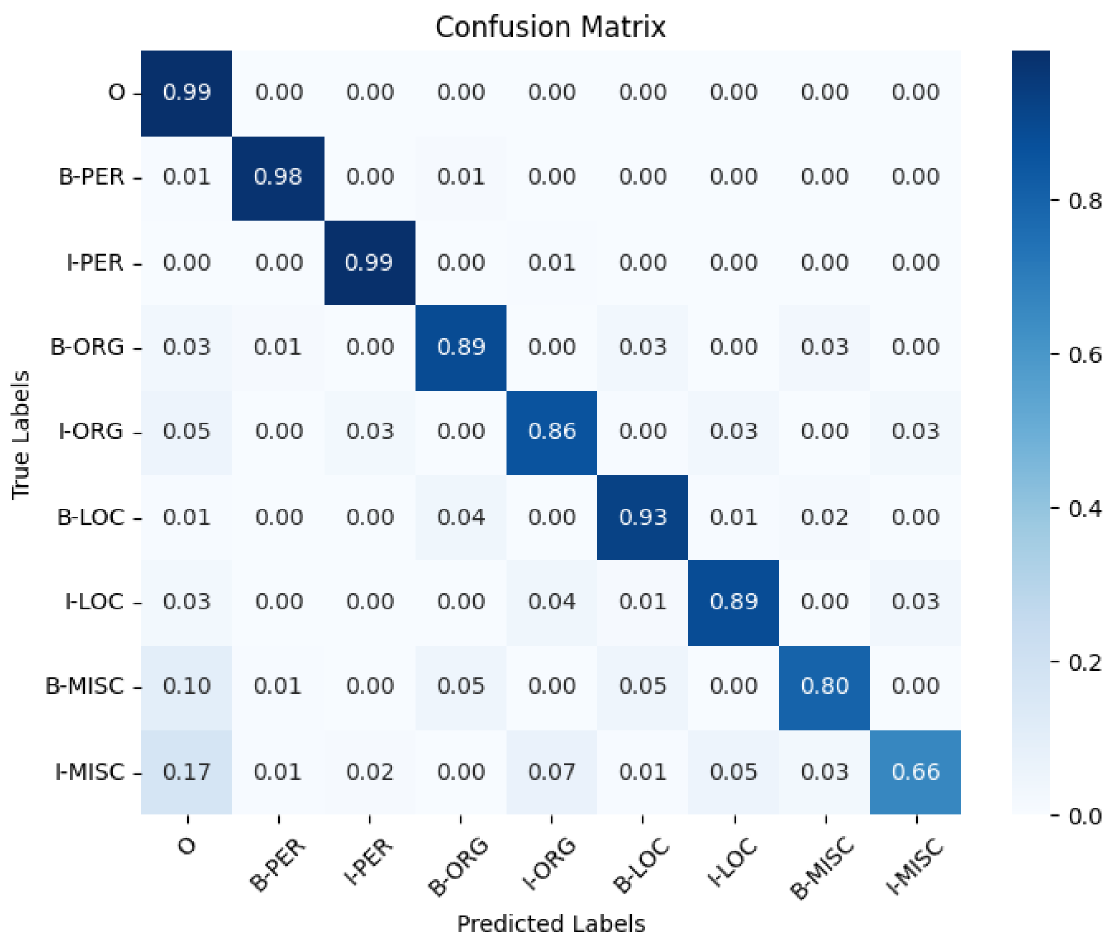
[124]: ['O', 'B-PER', 'I-PER', 'B-ORG', 'I-ORG', 'B-LOC', 'I-LOC', 'B-MISC', 'I-MISC']

[125]:
```
#  Generate the confusion matrix
cm = confusion_matrix(filtered_labels, filtered_predictions,normalize='true')

# Plotting the confusion matrix
plt.figure(figsize=(8, 6))
ax= sns.heatmap(cm, annot=True, fmt=".2f", cmap="Blues",
↪xticklabels=class_names, yticklabels=class_names)
# Ensure x-labels are vertical
```

```
ax.set_xticklabels(ax.get_xticklabels(), rotation=45)

# Ensure y-labels are horizontal
ax.set_yticklabels(ax.get_yticklabels(), rotation=0)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```



[126]:

```
# Log the confusion matrix to the Weights & Biases (Wandb) platform for␣
↪monitoring and visualization.
# This allows for tracking the model's classification performance across␣
↪different runs or iterations.

# log the Confusion Matrix to Wandb
wandb.log({
    "conf_mat": wandb.plot.confusion_matrix(
```

```
        preds=filtered_predictions,         # Model's predicted class labels.
        y_true=filtered_labels,       # Actual labels from the validation set.
        class_names=class_names    # Custom class names for display in the␣
 ↪confusion matrix.
    )
})
```

[127]:
```
wandb.finish()
```

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

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

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

5.10.3 Check the best saved model

[130]:
```
# After training, let us check the best checkpoint
# We need this for Predictions and Evaluations
best_model_checkpoint_step = trainer.state.best_model_checkpoint.split('-')[-1]
print(f"The best model was saved at step {best_model_checkpoint_step}.")
```

```
The best model was saved at step 620.
```

6 Inference

6.1 Pipeline for Predictions

[129]:
```
test_subset
```

[129]:
```
DatasetDict({
    test: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'],
        num_rows: 1000
    })
})
```

6.2 Create pipelne for inference

[131]:
```
checkpoint = str(model_folder / "checkpoint-620")
checkpoint
# Create a text classification pipeline using the Hugging Face's pipeline␣
 ↪method.
# The pipeline is initialized with:
# - The task set to "text-classification".
# - Model and tokenizer both loaded from the specified checkpoint path.
```

```python
# - Execution set to the primary device (typically the first GPU).

custom_pipeline = pipeline(
    task="token-classification",
    model=checkpoint,
    tokenizer=checkpoint,
    device='cpu',
    aggregation_strategy="simple",
    framework="pt")
```

6.3 Prediction for individual or small list of examples

[132]:
```python
idx=100
sample = " ".join(test_subset['test']['tokens'][idx])
print(sample)
```

```
Brian Shimer piloted USA III to a surprise victory in a World Cup two-man
bobsleigh race on Saturday .
```

[133]:
```python
preds = custom_pipeline(sample)
display(pd.DataFrame(preds))
```

	entity_group	score	word	start	end
0	PER	0.989874	brian shimer	0	12
1	ORG	0.615834	usa iii	21	28
2	MISC	0.866845	world cup	56	65

6.4 Prediction for large dataset

[134]:
```python
def preprocess_for_pipeline(example):
    return {"processed_text": " ".join(example['tokens'])}

test_subset['test'] = test_subset['test'].map(preprocess_for_pipeline)
```

```
Map:   0%|          | 0/1000 [00:00<?, ? examples/s]
```

[135]:
```python
test_subset
```

[135]:
```
DatasetDict({
    test: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags',
'processed_text'],
        num_rows: 1000
    })
})
```

[136]:
```python
custom_pipeline = pipeline(
    task="token-classification",
    model=checkpoint,
    tokenizer=checkpoint,
    device=0,
    aggregation_strategy="simple",
    framework="pt")

predictions = custom_pipeline(test_subset['test']['processed_text'],␣
 ↪batch_size=16)
```

[137]:
```python
predictions[0]
```

[137]:
```
[{'entity_group': 'ORG',
  'score': 0.98068917,
  'word': 'hartford',
  'start': 0,
  'end': 8},
 {'entity_group': 'ORG',
  'score': 0.97446424,
  'word': 'boston',
  'start': 11,
  'end': 17}]
```

6.5 Test Set Evaluations

[138]:
```python
test_subset
```

[138]:
```
DatasetDict({
    test: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags',
'processed_text'],
        num_rows: 1000
    })
})
```

[139]:
```python
test_subset_tokenized = test_subset.map(tokenize_fn, batched=True)
```

```
Map:   0%|          | 0/1000 [00:00<?, ? examples/s]
```

[140]:
```python
test_subset_tokenized
```

[140]:
```
DatasetDict({
    test: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags',
'processed_text', 'input_ids', 'attention_mask', 'labels'],
        num_rows: 1000
```

```
    })
})
```

[141]:
```
test_subset_tokenized = test_subset_tokenized.remove_columns(['tokens',␣
↪'ner_tags', 'pos_tags','chunk_tags', 'id', 'processed_text'])
```

[142]:
```
test_subset_tokenized
```

[142]:
```
DatasetDict({
    test: Dataset({
        features: ['input_ids', 'attention_mask', 'labels'],
        num_rows: 1000
    })
})
```

[143]:
```
from transformers import AutoModelForTokenClassification, AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained(checkpoint)
model = AutoModelForTokenClassification.from_pretrained(checkpoint)

from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="./results",
    per_device_eval_batch_size=16,  # adjust based on your GPU memory
    do_train = False,
    do_eval=True,
    report_to=[]  # disable logging
)

trainer = Trainer(
    model=model,
    args=training_args,
    eval_dataset=test_subset_tokenized['test'],  # Make sure this dataset is␣
↪tokenized!
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)

results = trainer.evaluate()
```

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

[144]:
```
results
```

[144]:

```
{'eval_loss': 0.12417706102132797,
 'eval_precision': 0.8077144502014968,
 'eval_recall': 0.8575794621026895,
 'eval_f1': 0.8319003854135784,
 'eval_accuracy': 0.9651442307692307,
 'eval_runtime': 1.8534,
 'eval_samples_per_second': 539.546,
 'eval_steps_per_second': 33.991}
```

[ ]:

```

```