Drew Lickman

CSCI 4820-1

Project #5

Due: 11/19/24

A.I. Disclaimer: Work for this assignment was completed with the aid of artificial intelligence tools and comprehensive documentation of the names of, input provided to, and output obtained from, these tools is included as part of my assignment submission.

BERT Named Entity Recognition Fine Tuning Project Starter Code

Dr. Sal Barbosa, Department of Computer Science, Middle Tennessee State University

In [1]:
```
# Required on TAMU FASTER to be able to pip install packages and download the dataset from Hugging Face
import os
os.environ['http_proxy'] = 'http://10.72.8.25:8080'
os.environ['https_proxy'] = 'http://10.72.8.25:8080'
```

In [2]:
```
# pip installs - comment out after running the notebook for the first time
#!pip install datasets
#!pip install evaluate
#!pip install seqeval
#!pip install accelerate==0.26.1
```

In [3]:
```
import torch
from transformers import AutoModelForTokenClassification, AutoTokenizer, Trainer, TrainingArguments
from datasets import load_dataset, DatasetDict, Sequence, ClassLabel
import numpy as np
import evaluate
from collections import Counter
```

```
2024-11-17 17:07:11.706768: I tensorflow/core/util/port.cc:111] oneDNN custom operations are on. You may see sli
ghtly different numerical results due to floating-point round-off errors from different computation orders. To t
urn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-11-17 17:07:13.801273: E tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:9342] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered
2024-11-17 17:07:13.801308: E tensorflow/compiler/xla/stream_executor/cuda/cuda_fft.cc:609] Unable to register c
uFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered
2024-11-17 17:07:13.814736: E tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:1518] Unable to register
cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered
2024-11-17 17:07:14.643543: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optim
ized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with
the appropriate compiler flags.
2024-11-17 17:07:18.270226: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not fi
nd TensorRT
```

In [4]:
```
# Load the CONLL-2003 NER dataset
dataset = load_dataset("conll2003")

# Remove columns not used in this code
dataset = dataset.remove_columns(['id', 'pos_tags', 'chunk_tags'])
dataset
```

Out[4]:
```
DatasetDict({
    train: Dataset({
        features: ['tokens', 'ner_tags'],
        num_rows: 14041
    })
    validation: Dataset({
        features: ['tokens', 'ner_tags'],
        num_rows: 3250
    })
    test: Dataset({
        features: ['tokens', 'ner_tags'],
        num_rows: 3453
    })
})
```

In [5]:
```
# Get and display the NER tag list for the dataset
label_list = dataset["train"].features["ner_tags"].feature.names
```

```
# Rename PERSON labels to MALE labels
label_list[1] = 'B-MPER'
label_list[2] = 'I-MPER'

# Append FEMALE labels at end of label list
label_list.append('B-FPER')
label_list.append('I-FPER')

print("Label list:", label_list)
```

```
Label list: ['O', 'B-MPER', 'I-MPER', 'B-ORG', 'I-ORG', 'B-LOC', 'I-LOC', 'B-MISC', 'I-MISC', 'B-FPER', 'I-FPER'
]
```

In [6]:

```
# Load the BERT cased model
model_checkpoint = "bert-base-cased"
tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)
model = AutoModelForTokenClassification.from_pretrained(model_checkpoint, num_labels=len(label_list))
```

```
/home/u.al234966/.local/lib/python3.11/site-packages/huggingface_hub/file_download.py:797: FutureWarning: `resum
e_download` is deprecated and will be removed in version 1.0.0. Downloads always resume when possible. If you wa
nt to force a new download, use `force_download=True`.
  warnings.warn(
Some weights of BertForTokenClassification were not initialized from the model checkpoint at bert-base-cased 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.
```

I used Claude 3.5 Sonnet (New) to create a basic classifier to help in labeling the name prefix/suffix patterns, instead of labeling entire individual names. Not all of Claude's predictions were true, so I spent a few hours manually labeling the prefixes and suffixes as male or female (in the next code block). Claude was able to give me a rough probability estimate of the name's gender, but I specifically chose the actual prefixes and suffixes to be used. I also had Claude create a table of the 5 highest potential suffixes and prefixes, which was slightly helpful, but not as helpful as I was hoping.

In [7]:

```
# import React, { useState, useEffect } from 'react';
# import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';
# import _ from 'lodash';

# const NameGenderProcessor = () => {
# const [input, setInput] = useState('');
# const [output, setOutput] = useState('');
# const [patternAnalysis, setPatternAnalysis] = useState('');

# const predictGender = (name) => {
#       name = name.toLowerCase().trim();
#       let femaleScore = 0;
#       let maleScore = 0;

#       // Strong male name patterns (endings)
#       const strongMaleEndings = [
#       { pattern: 'bert', weight: 0.85 }, // Robert, Albert
#       { pattern: 'son', weight: 0.85 },  // Wilson, Jackson
#       { pattern: 'ard', weight: 0.8 },   // Richard, Edward
#       { pattern: 'rick', weight: 0.8 },  // Patrick, Frederick
#       { pattern: 'les', weight: 0.75 },  // Charles, Miles
#       { pattern: 'roy', weight: 0.75 },  // Troy, Leroy
#       { pattern: 'ryan', weight: 0.75 }, // Bryan, Ryan
#       { pattern: 'ton', weight: 0.7 },   // Clinton, Winston
#       { pattern: 'ford', weight: 0.7 },  // Bradford, Clifford
#       { pattern: 'vin', weight: 0.65 }   // Kevin, Calvin
#       ];

#       // Strong female name patterns (endings)
#       const strongFemaleEndings = [
#       { pattern: 'ette', weight: 0.9 },  // Annette, Paulette
#       { pattern: 'elle', weight: 0.9 },  // Michelle, Belle
#       { pattern: 'ina', weight: 0.85 },  // Christina, Marina
#       { pattern: 'yah', weight: 0.85 },  // Mariah, Aliyah
#       { pattern: 'iah', weight: 0.85 },  // Moriah, Sariah
#       { pattern: 'lyn', weight: 0.85 },  // Evelyn, Carolyn
#       { pattern: 'anne', weight: 0.8 },  // Suzanne, Marianne
#       { pattern: 'ella', weight: 0.85 }, // Isabella, Stella
#       { pattern: 'icia', weight: 0.85 }, // Patricia, Alicia
#       { pattern: 'ley', weight: 0.8 },   // Ashley, Shirley
#       { pattern: 'acy', weight: 0.8 },   // Tracy, Stacy
#       { pattern: 'ren', weight: 0.75 },  // Lauren, Karen
#       { pattern: 'rie', weight: 0.75 },  // Marie, Carrie
```

```
#       { pattern: 'ora', weight: 0.75 },  // Flora, Nora
#       { pattern: 'ey', weight: 0.7 }     // Sydney, Casey
#       ];

#       // Male name beginnings
#       const maleBeginnings = [
#       { pattern: 'jo', weight: 0.65 },   // John, Joseph
#       { pattern: 'br', weight: 0.6 },    // Bruce, Brandon
#       { pattern: 'gr', weight: 0.6 },    // Greg, Grant
#       { pattern: 'fr', weight: 0.6 },    // Frank, Frederick
#       { pattern: 'st', weight: 0.5 }     // Steve, Stanley
#       ];

#       // Female name beginnings
#       const femaleBeginnings = [
#       { pattern: 'mel', weight: 0.6 },   // Melissa, Melody
#       { pattern: 'syl', weight: 0.6 },   // Sylvia
#       { pattern: 'bel', weight: 0.6 },   // Belinda, Isabella
#       { pattern: 'flo', weight: 0.6 },   // Florence
#       { pattern: 'mar', weight: 0.6 },   // Mary, Maria
#       { pattern: 'ali', weight: 0.6 }    // Alice, Alicia
#       ];

#       // Check strong patterns first
#       strongMaleEndings.forEach(({ pattern, weight }) => {
#       if (name.endsWith(pattern)) maleScore += weight;
#       });

#       strongFemaleEndings.forEach(({ pattern, weight }) => {
#       if (name.endsWith(pattern)) femaleScore += weight;
#       });

#       // Check beginnings
#       maleBeginnings.forEach(({ pattern, weight }) => {
#       if (name.startsWith(pattern)) maleScore += weight;
#       });

#       femaleBeginnings.forEach(({ pattern, weight }) => {
#       if (name.startsWith(pattern)) femaleScore += weight;
#       });

#       // Consonant patterns (more common in male names)
#       const hasStrongConsonantCluster = /[bcdfghjklmnpqrstvwxz]{3,}/.test(name);
#       if (hasStrongConsonantCluster) maleScore += 0.3;

#       // Soft sound patterns (more common in female names)
#       const hasSoftEnding = /[aeiou][ah]$|[aeiou]e$/.test(name);
#       if (hasSoftEnding) femaleScore += 0.5;

#       // Final letter patterns
#       const finalLetter = name.slice(-1);
#       if ('aie'.includes(finalLetter)) femaleScore += 0.4;
#       if ('ntkds'.includes(finalLetter)) maleScore += 0.3;

#       // Vowel patterns
#       const vowelCount = (name.match(/[aeiou]/g) || []).length;
#       const nameLength = name.length;
#       const vowelRatio = vowelCount / nameLength;

#       if (vowelRatio > 0.45) femaleScore += 0.4;
#       if (vowelRatio < 0.25) maleScore += 0.3;

#       // Repeated letter patterns (more common in female names)
#       const hasRepeatedLetters = /(.)\1/.test(name);
#       if (hasRepeatedLetters) femaleScore += 0.3;

#       // Additional female patterns
#       const hasMultipleVowelClusters = (name.match(/[aeiou]{2,}/g) || []).length;
#       if (hasMultipleVowelClusters > 0) femaleScore += 0.3;

#       // Ensure minimum scores
#       maleScore = Math.max(maleScore, 0.2);
#       femaleScore = Math.max(femaleScore, 0.2);

#       // Calculate confidence percentage
#       const total = femaleScore + maleScore;
#       const maxScore = Math.max(femaleScore, maleScore);
#       const confidence = Math.min(Math.round((maxScore / total) * 100), 95); // Cap at 95%

#       return {
#       gender: femaleScore > maleScore ? 'Female' : 'Male',
#       confidence
#       };
```

```
# };

# const analyzePatterns = (names) => {
#       if (!names.trim()) {
#       setPatternAnalysis('No data to analyze yet. Enter some names above.');
#       return;
#       }

#       const namesList = names.split('\n').filter(name => name.trim());
#       const patterns = {
#       maleStartings: { '1': {}, '2': {}, '3': {}, '4': {}, '5': {} },
#       maleEndings: { '1': {}, '2': {}, '3': {}, '4': {}, '5': {} },
#       femaleStartings: { '1': {}, '2': {}, '3': {}, '4': {}, '5': {} },
#       femaleEndings: { '1': {}, '2': {}, '3': {}, '4': {}, '5': {} }
#       };

#       // Process each name
#       namesList.forEach(nameInput => {
#       const name = nameInput.trim().toLowerCase();
#       if (!name) return;

#       const result = predictGender(name);
#       const confidence = result.confidence / 100;

#       // Only consider patterns from predictions with confidence > 60%
#       if (confidence < 0.6) return;

#       // Get patterns of lengths 1-5 for both start and end
#       for (let i = 1; i <= 5; i++) {
#               if (name.length >= i) {
#               const start = name.slice(0, i);
#               const end = name.slice(-i);

#               if (result.gender === 'Male') {
#                       patterns.maleStartings[i][start] = (patterns.maleStartings[i][start] || 0) + confidence
#                       patterns.maleEndings[i][end] = (patterns.maleEndings[i][end] || 0) + confidence;
#               } else {
#                       patterns.femaleStartings[i][start] = (patterns.femaleStartings[i][start] || 0) + confid
#                       patterns.femaleEndings[i][end] = (patterns.femaleEndings[i][end] || 0) + confidence;
#               }
#               }
#       }
#       });

#       // Helper function to get top patterns
#       const getTopPatterns = (patternObj, count = 5) => {
#       return Object.entries(patternObj)
#               .sort((a, b) => b[1] - a[1])
#               .slice(0, count)
#               .map(([pattern, score]) => `${pattern} (${(score/namesList.length * 100).toFixed(1)}%)`)
#               .join('\n    ');
#       };

#       // Format analysis for each length and type
#       // Helper function to format top 5 patterns for a cell
#       const formatCell = (patterns) => {
#       if (!patterns || patterns.length === 0) return '-';
#       return Object.entries(patterns)
#               .sort((a, b) => b[1] - a[1])
#               .slice(0, 5)
#               .map(([pattern, score]) => `${pattern} (${(score/namesList.length * 100).toFixed(1)}%)`)
#               .join('\n\t');
#       };

#       // Create table header
#       let table = ['Pattern Analysis (based on ' + namesList.length + ' names)\n\n'];
#       table.push('Length\tMale Start\tMale End\tFemale Start\tFemale End\n');
#       table.push('-'.repeat(105) + '\n');

#       // Generate each row of the table for lengths 1-5
#       for (let length = 1; length <= 5; length++) {
#       // Get top 5 patterns for each category
#       const maleStarts = formatCell(patterns.maleStartings[length.toString()]);
#       const maleEnds = formatCell(patterns.maleEndings[length.toString()]);
#       const femaleStarts = formatCell(patterns.femaleStartings[length.toString()]);
#       const femaleEnds = formatCell(patterns.femaleEndings[length.toString()]);

#       // Split each category into lines (they'll have 5 lines each)
#       const maleStartLines = maleStarts.split('\n');
#       const maleEndLines = maleEnds.split('\n');
#       const femaleStartLines = femaleStarts.split('\n');
#       const femaleEndLines = femaleEnds.split('\n');
```

```
#       // Add the length indicator and first line
#       table.push(`${length}\t${maleStartLines[0]}\t${maleEndLines[0]}\t${femaleStartLines[0]}\t${femaleEndLin

#       // Add remaining lines with proper spacing
#       for (let i = 1; i < 5; i++) {
#               table.push(`\t${maleStartLines[i] || ''}\t${maleEndLines[i] || ''}\t${femaleStartLines[i] || ''
#       }

#       // Add separator between lengths
#       table.push('-'.repeat(105) + '\n');
#       }

#       table.push('\nNote: Percentages indicate frequency weighted by confidence scores.');

#       setPatternAnalysis(table.join(''));

#       setPatternAnalysis(table.join(''));

#       setPatternAnalysis(table.join(''));

#       setPatternAnalysis(analysisText);
# };

# const processNames = (text) => {
#       const lines = text.split('\n');
#       const results = {
#       male: [],
#       female: []
#       };

#       // Process and categorize each name
#       lines.forEach(line => {
#       const trimmedLine = line.trim();
#       if (!trimmedLine) return;

#       const result = predictGender(trimmedLine);
#       const entry = {
#               name: trimmedLine,
#               confidence: result.confidence,
#               formatted: `${trimmedLine}: ${result.gender} (${result.confidence}% confidence)`
#       };

#       if (result.gender === 'Male') {
#               results.male.push(entry);
#       } else {
#               results.female.push(entry);
#       }
#       });

#       // Sort each category by confidence (descending)
#       results.male.sort((a, b) => b.confidence - a.confidence);
#       results.female.sort((a, b) => b.confidence - a.confidence);

#       // Format the output with headers and sorted results
#       const outputText = [
#       `FEMALE NAMES (${results.female.length} total):`,
#       ...results.female.map(entry => entry.formatted),
#       '',
#       `MALE NAMES (${results.male.length} total):`,
#       ...results.male.map(entry => entry.formatted)
#       ].join('\n');

#       setOutput(outputText);

#       // Analyze patterns whenever names are processed
#       analyzePatterns(text);
# };

# const handleInputChange = (e) => {
#       const newInput = e.target.value;
#       setInput(newInput);
#       processNames(newInput);
# };

# return (
#       <Card className="w-full max-w-4xl">
#       <CardHeader>
#               <CardTitle>Enhanced Name Gender Processor</CardTitle>
#       </CardHeader>
#       <CardContent className="space-y-4">
#               <div>
#               <label className="block text-sm font-medium mb-2">
#                       Input Names (one per line)
```

```
#               </label>
#               <textarea
#                       value={input}
#                       onChange={handleInputChange}
#                       className="w-full h-48 p-2 border rounded-md"
#                       placeholder="Enter names here..."
#               />
#               </div>
#               <div>
#               <label className="block text-sm font-medium mb-2">
#                       Processed Results (with confidence scores)
#               </label>
#               <textarea
#                       value={output}
#                       readOnly
#                       className="w-full h-48 p-2 border rounded-md bg-gray-50"
#               />
#               </div>
#               <div>
#               <label className="block text-sm font-medium mb-2">
#                       Pattern Analysis
#               </label>
#               <textarea
#                       value={patternAnalysis}
#                       readOnly
#                       className="w-full h-48 p-2 border rounded-md bg-gray-50 font-mono text-sm"
#               />
#               </div>
#       </CardContent>
#       </Card>
# );
# };

# export default NameGenderProcessor;
```

In [8]:

```
def isShortNonsense(token):
    # Removes any tokens that include non alphabetic characters, or single chars
    if not token.isalpha() or len(token) == 1:
        return True
    return False

# Some names came from https://nameberry.com/blog/the-most-popular-baby-name-endings 11/15/24
femaleWhole = ("Taha", "Olga", "Andi", "Inga", "Ro", "Deby", "Abu", "Mia", "Rui", "Tracy", "El", "Kim", "Lauren
femaleBeginnings = ("ali", "hart", "sene", "xu", "cuo", "esp", "blen", "nas", "arc", "jass", "anin", "oue", "pa
femaleEndings = ("ati", "xis", "lde", "yte", "gla", "rwe", "hla", "ta", "tto", "tle", "ye", "ivo", "evre", "odi
def is_female_name(token):
    # Uses algorithmic approach to assign gender tag and returns boolean
    result = False
    #if any(token.lower().endswith(ending) for ending in femaleEndings):
    if token.lower().endswith(femaleEndings):
        result = True
    #elif any(token.lower().startswith(start) for start in femaleBeginnings):
    elif token.lower().startswith(femaleBeginnings):
        result = True
    elif token in femaleWhole:
        result = True
    #if result: print("F:",token)
    return result

maleWhole = ("Jimi", "Levy", "Sammy", "Anders", "Ty", "Jens", "Andre", "Cam", "Mo", "Alec", "Gale", "Andy", "Fr
maleBeginnings = ("man", "mr", "bat", "jyr", "japhe", "betho", "map", "gil", "lou", "rup", "arr", "beck", "jin"
maleEndings = ("drew", "ner", "had", "das", "pt", "epp", "rki", "map", "shu", "rav", "nat", "ko", "lab", "ndi",
def is_male_name(token):
    # Uses algorithmic approach to assign gender tag and returns boolean
    result = False
    #if any(token.lower().endswith(ending) for ending in maleEndings):
    if token.lower().endswith(maleEndings):
        result = True
    #elif any(token.lower().startswith(start) for start in maleBeginnings):
    elif token.lower().startswith(maleBeginnings):
        result = True
    elif token in maleWhole:
        result = True
    #if result: print("M:",token)
    return result

# Implement with more algorithms to specify weights for name parts:
# def chooseGender(femResult, maleResult):
#       # If I were to use weights (instead of booleans), I can compare
#       result = femResult - maleResult
#       if result >= 0:
#               return "Female"
#       else:
```

```
#               return "Male"
```

In [9]:
```
# Tokenization and tag distribution function
def tokenize_and_distribute_tags(examples):
        tokenized_inputs = tokenizer(
                examples["tokens"],
                truncation=True,
                is_split_into_words=True,
                padding='max_length',
                max_length=128
        )

        labels = []
        for i, label in enumerate(examples["ner_tags"]):
                word_ids = tokenized_inputs.word_ids(batch_index=i)

                # For loop written by Claude 3.5 Sonnet
                # It iterates through all the example tokens and classifies it as female or male
                modified_labels = []
                for j, tag in enumerate(label):
                        if (tag == 1 or tag == 2) and isShortNonsense(examples["tokens"][i][j]):        # Imple
                                modified_labels.append(tag)
                        elif tag == 1:  # If it's a B-PERSON tag
                                # Check the actual token using examples["tokens"][i][j]
                                if is_female_name(examples["tokens"][i][j]):    # Implemented by myself, Drew L
                                        modified_labels.append(9)       # Convert B-PERSON to B-FPER index
                                elif is_male_name(examples["tokens"][i][j]):    # Implemented by myself, Drew L
                                        modified_labels.append(1)       # Convert B-PERSON to B-MPER index
                                else:
                                        print(examples["tokens"][i][j]) # Print person tags that aren't classif
                                        modified_labels.append(tag)
                        elif tag == 2:  # If it's an I-PERSON tag       # Note: this might miss names split int
                                # Keep the same type (MPER or FPER) as the previous B- tag
                                if modified_labels[-1] == 9:    # If previous was B-FPER
                                        modified_labels.append(10)      # I-FPER index
                                else:
                                        modified_labels.append(2)       # I-MPER index
                        else:   # Not a person tag
                                modified_labels.append(tag)

                label_ids = [-100 if word_id is None else modified_labels[word_id] for word_id in word_ids]
                #label_ids = [-100 if word_id is None else label[word_id] for word_id in word_ids]
                #print(f"Tag List: {label_list}\n\nTokens: {examples['tokens'][0]}\n\nTokenized: {tokenized_inp
                #\n\nTags: {label}\n\nTokenized word ids: {word_ids}\n\nDistributed tags: {label_ids}")
                #input()
                labels.append(label_ids)

        tokenized_inputs["labels"] = labels
        return tokenized_inputs

# Apply the tokenization function to the dataset
tokenized_datasets = dataset.map(tokenize_and_distribute_tags, batched=True)
```

In [10]:
```
# Metric fucntion
metric = evaluate.load("seqeval")

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

    true_labels = \
        [ [label_list[label] for label in label_seq if label != -100] for label_seq in labels ]
    model_predictions = \
        [ [label_list[pred] for (pred, label) in zip(pred_seq, label_seq) if label != -100] for pred_seq, label_

    results = metric.compute(predictions=model_predictions, references=true_labels)
    return {
        "precision": results["overall_precision"],
        "recall": results["overall_recall"],
        "f1": results["overall_f1"],
        "accuracy": results["overall_accuracy"],
    }
```

In [11]:
```
# Set training arguments
batch_size = 64
epochs = 1

training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
```

```
    num_train_epochs=epochs,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
    save_strategy="epoch",
    push_to_hub=False,
    report_to="none",
)

# Instantiate trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
)
```

```
Detected kernel version 4.18.0, which is below the recommended minimum of 5.5.0; this can cause the process to h
ang. It is recommended to upgrade the kernel to the minimum version or higher.
```

In [ ]:
```
# Train the model
trainer.train()

# Evaluate the model
results = trainer.evaluate()
print("Evaluation Results:", results)
```

```
You're using a BertTokenizerFast 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
.
```

 [220/220 05:50, Epoch 1/1]

Epoch	Training Loss	Validation Loss

 [20/51 00:08 < 00:13, 2.37 it/s]

In [ ]:
```
# Make predictions on the test set
predictions = trainer.predict(tokenized_datasets["test"])
pred_labels = np.argmax(predictions.predictions, axis=2)
true_labels = predictions.label_ids
```