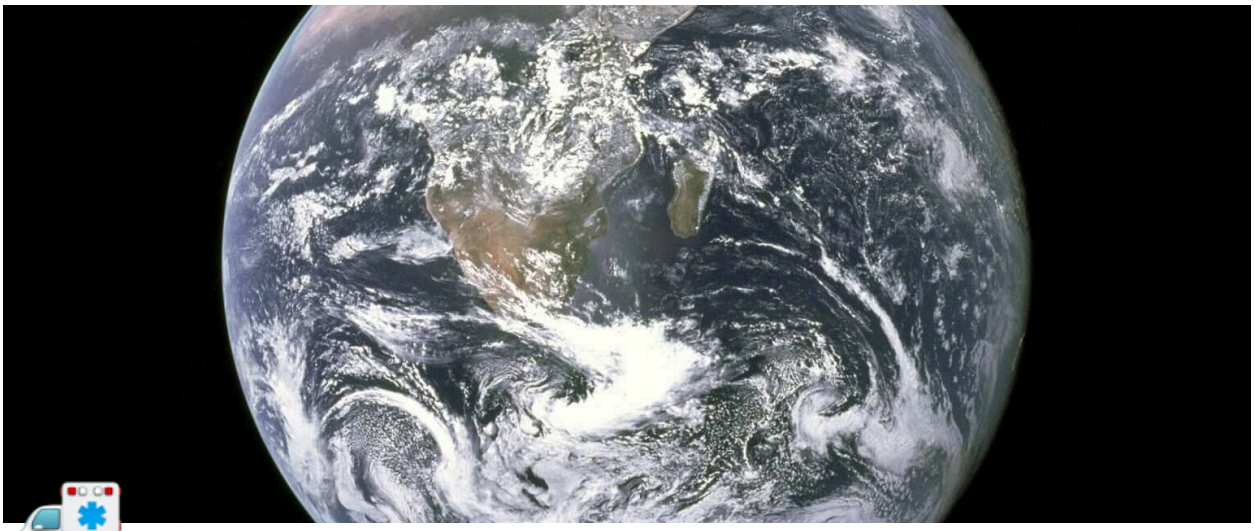

Token Classification

```
from transformers import AutoTokenizer, AutoModelForTokenClassification, Tr
ainer, TrainingArguments, DataCollatorForTokenClassification, pipeline
from datasets import load_dataset
import evaluate
import numpy as np

dataset = load_dataset("conll2003", split="train+validation+test", trust_remote
_code=True)
dataset = dataset.train_test_split(test_size=0.2)

tokenizer = AutoTokenizer.from_pretrained("distilbert/distilbert-base-uncase
d")

### Tokenizer split words into subwords which causes difference in tokens an
d labels length
### Assigning the label -100 to the special tokens [CLS] and [SEP] so they’re i
gnored by the PyTorch loss function
### (see CrossEntropyLoss).
### Only labeling the first token of a given word. Assign -100 to other subtoke
```

```
ns from the same word.

def tokenize_and_align_labels(examples):
    # Since sentence is split into words, we add is_split_into_words
    tokenized_inputs = tokenizer(examples["tokens"], is_split_into_words=True,
truncation=True)
    new_labels = []
    for idx, ner_tags in enumerate(examples["ner_tags"]):
        word_ids = tokenized_inputs.word_ids(batch_index=idx)
        new_label = []
        previous_word_id = None
        for word_id in word_ids:
            current_word_id = word_id
            # Special token
            if word_id is None:
                new_label.append(-100)
            # Start of a new word!
            elif previous_word_id != current_word_id:
                new_label.append(ner_tags[word_id])
            # Same word as previous token
            else:
                # # Optional to keep the label for subword tokens
                # # But If the label is B-XXX, we change it to I-XXX
                # label = ner_tags[word_id]
                # if label % 2 == 1:
                #     label += 1 # which is I-XXX
                # # Keep the label -100 for all subword token
                label = -100
                new_label.append(label)
            previous_word_id = word_id
        new_labels.append(new_label)
    tokenized_inputs["labels"] = new_labels
    return tokenized_inputs

labels_list = dataset["train"].features["ner_tags"].feature.names
dataset = dataset.map(tokenize_and_align_labels, batched=True, remove_colu
```

```
mns=['id', 'ner_tags', 'tokens'])
id2label = {idx: label for idx, label in enumerate(labels_list)}
seqeval = evaluate.load("seqeval")

def compute_metrics(output):
    predictions, labels = output
    predictions = np.argmax(predictions, axis=-1)
    new_predictions = [[labels_list[p] for p, l in zip(prediction, label) if l!=-100] f
or prediction, label in zip(predictions, labels)]
    new_labels = [[labels_list[l] for p, l in zip(prediction, label) if l!=-100] for pre
diction, label in zip(predictions, labels)]
    results = seqeval.compute(predictions=new_predictions, references=new_l
abels)
    return {
        "precision": results["overall_precision"],
        "recall": results["overall_recall"],
        "f1": results["overall_f1"],
        "accuracy": results["overall_accuracy"]}

data_collator = DataCollatorForTokenClassification(tokenizer=tokenizer)

model = AutoModelForTokenClassification.from_pretrained("distilbert/distilber
t-base-uncased",
                                        id2label=id2label)

training_args = TrainingArguments(output_dir="ner_model_wnut",
                per_device_train_batch_size=16,
                per_device_eval_batch_size=16,
                save_strategy="epoch",
                eval_strategy="epoch",
                load_best_model_at_end=True,
                metric_for_best_model="eval_loss",
                push_to_hub=False,
                num_train_epochs=10)
```

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

trainer.train()
trainer.save_model("ner_model_wnut")



classifier = pipeline("ner", "best_model", grouped_entities=True)
results = classifier("Vijay is going to be cm of tamilnadu")



from spacy import displacy

## Optional: For Visualization in Jupyter/Colab
# Create a spaCy-style Doc for visualization
doc = {
    "text": "Vijay is going to be cm of tamilnadu",
    "ents": [
        {"start": ent["start"], "end": ent["end"], "label": ent["entity_group"]}
        for ent in results
    ],
    "title": None
}

displacy.render(doc, style="ent", manual=True, jupyter=True)



Note:
Prefers only one label per word, and assign -100 to the other subtokens in a gi
ven word to avoid long words that split into lots of subtokens contributing hea
vily to the loss
```

Refer Link

conll2003 Labels Explanation

O means the word doesn’t correspond to any entity.
B-PER/I-PER means the word corresponds to the beginning of/is inside a pers on entity.
B-ORG/I-ORG means the word corresponds to the beginning of/is inside an or ganization entity.
B-LOC/I-LOC means the word corresponds to the beginning of/is inside a loca tion entity.
B-MISC/I-MISC means the word corresponds to the beginning of/is inside a m iscellaneous entity.

Note: MISC is used for nationalities (German, Japanese), events (Olympics, W orld Cup), artifacts (Windows 95, iPhone), sometimes adjectival forms of locat ions (French, Italian)
So MISC is like a "bucket" for named entities that aren’t strictly a person, orga nization, or location.