Named Entity Recognition and Classification (NERC) - BERT

```
!pip install seqeval
```

```
Collecting seqeval
  Downloading seqeval-1.2.2.tar.gz (43 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.6/43.6 kB 2.9 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.11/dist-packages (from seqeval) (1.26.4)
Requirement already satisfied: scikit-learn>=0.21.3 in /usr/local/lib/python3.11/dist-packages (from seqeval) (1.6.1)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=0.21.3->seqev
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=0.21.3->seqe
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=0.21.
Building wheels for collected packages: seqeval
  Building wheel for seqeval (setup.py) ... done
  Created wheel for seqeval: filename=seqeval-1.2.2-py3-none-any.whl size=16162 sha256=780571429eced5e16811384f22ed132af
  Stored in directory: /root/.cache/pip/wheels/bc/92/f0/243288f899c2eacdfa8c5f9aede4c71a9bad0ee26a01dc5ead
Successfully built seqeval
Installing collected packages: seqeval
Successfully installed seqeval-1.2.2
```

```
!pip install nbstripout
```

```
Collecting nbstripout
  Downloading nbstripout-0.8.1-py2.py3-none-any.whl.metadata (19 kB)
Requirement already satisfied: nbformat in /usr/local/lib/python3.11/dist-packages (from nbstripout) (5.10.4)
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.11/dist-packages (from nbformat->nbstripou
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.11/dist-packages (from nbformat->nbstripout) (4
Requirement already satisfied: jupyter-core!=5.0.*,>=4.12 in /usr/local/lib/python3.11/dist-packages (from nbformat->nbs
Requirement already satisfied: traitlets>=5.1 in /usr/local/lib/python3.11/dist-packages (from nbformat->nbstripout) (5.
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=2.6->nbformat-
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.11/dist-packages (from jso
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=2.6->nbf
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=2.6->nbformat
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.11/dist-packages (from jupyter-core!=5.0.*,>=
Requirement already satisfied: typing-extensions>=4.4.0 in /usr/local/lib/python3.11/dist-packages (from referencing>=0.
Downloading nbstripout-0.8.1-py2.py3-none-any.whl (16 kB)
Installing collected packages: nbstripout
Successfully installed nbstripout-0.8.1
```

```
!nbstripout ./content/nerc_bert.ipynb
```

```
Could not strip './content/nerc_bert.ipynb': file not found
```

```
!pwd
```

```
/content
```

```
import csv
import spacy
import re
import nltk
import pandas as pd

from typing import Set, List, Dict, Tuple

from datasets import load_dataset, load_metric

NLP = spacy.load("en_core_web_sm")


def word_shape(word: str):
    shape = re.sub("[A-Z]", "X", word)
    shape = re.sub("[a-z]", "x", shape)
    shape = re.sub("[0-9]", "d", shape)
    shape = re.sub(r"\W", "w", shape)
    return shape


def gather_test_bio_ner_tags(file_name: str) -> Set[str]:
    bio_ner_tags = set()
    with open(file_name, "r") as f:
        data = csv.DictReader(f=f, delimiter="\t")
        for row in data:
            bio_ner_tags.add(row["bio_ner_tag"])

    return bio_ner_tags
```

```
def nerc_data_to_file(raw_data: Dataset, file_name: str):
    try:
        with open(file_name, "w", newline="", encoding="utf-8") as f:
            writer: csv.writer = csv.writer(f, delimiter="\t")
            writer.writerow(["sentence_id", "token_id", "token", "bio_ner_tag"])

            for idx, sent in enumerate(raw_data):
                sentence = " ".join(sent["tokens"])
                doc = NLP(sentence)

                bio_tags = ["O"] * len(doc)

                for ent in doc.ents:
                    bio_tags[ent.start] = f"B-{ent.label_}"
                    for i in range(ent.start + 1, ent.end):
                        bio_tags[i] = f"I-{ent.label_}"

                for token_id, token in enumerate(doc):
                    writer.writerow([idx, token_id, token.text, bio_tags[token_id]])

            f.close()

        print("Converted successfully!")
    except Exception as e:
        return {"error": str(e)}


def gather_tokens_and_tags(df: pd.DataFrame) -> Tuple[List[str], List[str]]:
    X, y = [], []

    sent_tokens = []
    sent_tags = []

    for token, tag in zip(df["token"], df["bio_ner_tag"]):
        sent_tokens.append(token)
        sent_tags.append(tag)

        if token in [".", "!", "?"]:
            X.append(sent_tokens)
            y.append(sent_tags)
            sent_tokens = []
            sent_tags = []

    if sent_tokens:
        X.append(sent_tokens)
        y.append(sent_tags)

    return X, y



def sentiment_data_to_file(raw_data: Dataset, file_name: str):
    try:
        with open(file_name, "w", newline="", encoding="utf-8") as f:
            writer: csv.writer = csv.writer(f, delimiter="\t")
            writer.writerow(["sentence_id", "sentence", "sentiment"])

            for idx, elem in enumerate(raw_data):
                sentence = elem["sentence"]
                label = "positive" if elem["label"] == 1 else "negative"

                writer.writerow([idx, sentence, label])

        f.close()
        print("Converted successfully!")
    except Exception as e:
        return {"message": str(e)}


def topic_data_to_file(raw_data: Dataset, file_path: str):
    with open(file_path, "w", newline="", encoding="utf-8") as f:
        writer = csv.writer(f, delimiter = "\t")
        writer.writerow(["id", "question", "category"])
        for entry in raw_data:
            id_ = entry["id"]
            q = entry["question"]
            category = entry["category"]

            if category == "movies":
                category = "movie"
            elif category == "books":
```

```
                category = "book"

            writer.writerow([id_, q, category])

        f.close()


def extract_features(sentence, pos_tags, i):
    word = sentence[i]
    pos = pos_tags[i]

    if not isinstance(word, str):
        word = str(word)

    features = {
        "bias": 1.0,
        "word.lower()": word.lower(),
        "word[-3:]": word[-3:],
        "word[-2:]": word[-2:],
        "word.isupper()": word.isupper(),
        "word.istitle()": word.istitle(),
        "word.isdigit()": word.isdigit(),
        "pos": pos,
        "word.shape": word_shape(word=word)

    }
    if i > 0:
        word1 = sentence[i-1]
        pos1 = pos_tags[i-1]

        if not isinstance(word1, str):
            word1 = str(word1)

        features.update({
            "-1:word.lower()": word1.lower(),
            "-1:word.istitle()": word1.istitle(),
            "-1:word.isupper()": word1.isupper(),
            "-1:pos": pos1,
            "-1:word.shape": word_shape(word=word1)
        })
    else:
        features["BOS"] = True

    if i < len(sentence) - 1:
        word1 = sentence[i+1]
        pos1 = pos_tags[i+1]

        if not isinstance(word1, str):
            word1 = str(word1)

        features.update({
            "+1:word.lower()": word1.lower(),
            "+1:word.istitle()": word1.istitle(),
            "+1:word.isupper()": word1.isupper(),
            "+1:pos": pos1,
            "+1:pos": word_shape(word=word1)
        })
    else:
        features["EOS"] = True

    return features

def sentence_to_features(sentence):
    cleaned_sentence = [str(token) if not isinstance(token, str) else token for token in sentence]
    pos_tags = [pos for _, pos in nltk.pos_tag(cleaned_sentence)]
    return [extract_features(cleaned_sentence, pos_tags, i) for i in range(len(sentence))]




import pandas as pd
import numpy as np
import torch
import seqeval

from typing import List, Dict, Union
from transformers import AutoTokenizer, AutoModelForTokenClassification, TrainingArguments, Trainer
from torch.utils.data import Dataset
from datasets import Dataset as hf_Dataset


train_data_ner_file: str = r"./NER-train.tsv"
```

```
df = pd.read_csv(train_data_ner_file, sep="\t")


X, y = gather_tokens_and_tags(df=df)

train_data: List[Dict[str,str]] = []
for tokens, ner_tags in zip(X, y):
    tokens = [str(token) for token in tokens]
    ner_tags = [str(ner_tag) for ner_tag in ner_tags]

    train_data.append({
        "tokens": np.asarray(tokens),
        "ner_tags": np.asarray(ner_tags)
    })

dataset = hf_Dataset.from_list(train_data)


TOKENIZER = AutoTokenizer.from_pretrained("bert-base-cased")


label_list = sorted(set(label for seq in y for label in seq))
label_to_id = {label: i for i, label in enumerate(label_list)}
id_to_label = {i: label for label, i in label_to_id.items()}


def preprocess_function(examples):
    tokenized_inputs = TOKENIZER(
        examples["tokens"],
        is_split_into_words=True,
        return_offsets_mapping=True,
        padding="max_length",
        truncation=True
    )

    all_labels = []
    for i, labels in enumerate(examples["ner_tags"]):
        word_ids = tokenized_inputs.word_ids(batch_index=i)
        previous_word_idx = None
        label_ids = []
        for word_idx in word_ids:
            if word_idx is None:
                label_ids.append(-100)
            elif word_idx != previous_word_idx:
                label_ids.append(label_to_id[labels[word_idx]])
            else:
                label_ids.append(-100)
            previous_word_idx = word_idx
        all_labels.append(label_ids)

    tokenized_inputs["labels"] = all_labels
    return tokenized_inputs


tokenized_dataset = dataset.map(preprocess_function, batched=True)
```

```
Map: 100%  8128/8128 [00:07<00:00, 1069.06 examples/s]
```

```
# tokenized_dataset.set_format(
#     type="torch",
#     columns=["input_ids", "attention_mask", "token_type_ids", "labels"]
# )


model = AutoModelForTokenClassification.from_pretrained("bert-base-uncased", num_labels=len(label_list))
```

```
Some weights of BertForTokenClassification were not initialized from the model checkpoint at bert-base-uncased and are n
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
output_dir = r"./bert_model"


training_args = TrainingArguments(
    output_dir=output_dir,
    # evaluation_strategy="no",
    eval_steps=250,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    learning_rate=2e-5,
    weight_decay=0.4,
    logging_steps=100,
```

```
    save_steps=250,
    fp16=True
)


trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset,
    processing_class=TOKENIZER
)


print(tokenized_dataset[0])
```

```
{'tokens': ['EU', 'rejects', 'German', 'call', 'to', 'boycott', 'British', 'lamb', '.'], 'ner_tags': ['B-ORG', 'O', 'B-N
```

```
def start_finetuning(trainer: Trainer):
    print("Starting fine-tuning...")
    trainer.train()
    print("Fine-tuning complete!")

start_finetuning(trainer=trainer)
```

```
Starting fine-tuning...
```

[3048/3048 18:47, Epoch 3/3]

Step	Training Loss
100	1.440300
200	1.119100
300	0.992100
400	0.878900
500	0.839800
600	0.785200
700	0.771600
800	0.726200
900	0.686800
1000	0.676300
1100	0.581000
1200	0.584100
1300	0.535200
1400	0.529400
1500	0.566200
1600	0.500000
1700	0.496800
1800	0.502000
1900	0.497900
2000	0.468300
2100	0.451000
2200	0.444700
2300	0.436100
2400	0.430200
2500	0.430500
2600	0.397500
2700	0.426600
2800	0.401700
2900	0.391100
3000	0.408900

```
Fine-tuning complete!
```

```
trainer.state.log_history
```

```
 'step': 2000},
{'loss': 0.451,
 'grad_norm': 3.12327241897583,
 'learning_rate': 6.246719160104987e-06,
 'epoch': 2.0669291338582676,
 'step': 2100},
{'loss': 0.4447,
 'grad_norm': 4.637417316436768,
 'learning_rate': 5.590551181102362e-06,
 'epoch': 2.1653543307086616,
 'step': 2200},
{'loss': 0.4361,
 'grad_norm': 4.117068767547607,
 'learning_rate': 4.934383202099738e-06,
 'epoch': 2.263779527559055,
 'step': 2300},
{'loss': 0.4302,
 'grad_norm': 3.926618814468384,
 'learning_rate': 4.278215223097113e-06,
 'epoch': 2.362204724409449,
 'step': 2400},
{'loss': 0.4305,
 'grad_norm': 5.361767292022705,
 'learning_rate': 3.6220472440944887e-06,
 'epoch': 2.4606299212598426,
 'step': 2500},
{'loss': 0.3975,
 'grad_norm': 7.588459014892578,
 'learning_rate': 2.965879265091864e-06,
 'epoch': 2.559055118110236,
 'step': 2600},
{'loss': 0.4266,
 'grad_norm': 5.533886909484863,
 'learning_rate': 2.309711286089239e-06,
 'epoch': 2.65748031496063,
 'step': 2700},
{'loss': 0.4017,
 'grad_norm': 4.550146102905273,
 'learning_rate': 1.6535433070866144e-06,
 'epoch': 2.7559055118110236,
 'step': 2800},
{'loss': 0.3911,
 'grad_norm': 4.413469314575195,
 'learning_rate': 9.973753280839895e-07,
 'epoch': 2.8543307086614176,
 'step': 2900},
{'loss': 0.4089,
 'grad_norm': 5.253135681152344,
 'learning_rate': 3.4120734908136486e-07,
 'epoch': 2.952755905511811,
 'step': 3000},
{'train_runtime': 1129.4559,
 'train_samples_per_second': 21.589,
 'train_steps_per_second': 2.699,
 'total_flos': 6373361047633920.0,
 'train_loss': 0.6102312593635298,
 'epoch': 3.0,
 'step': 3048}]
```

```
test_data_path = "./NER-test.tsv"
```

```
test_df: pd.DataFrame = pd.read_csv(test_data_path, sep="\t")
```

```
test_df.head()
```

	sentence_id	token_id	token	bio_ner_tag
0	0	0	If	O
1	0	1	you're	O
2	0	2	visiting	O
3	0	3	Paris	B-LOCATION
4	0	4	,	O

```
X_test, y_test = gather_tokens_and_tags(df=test_df)

test_data: List[Dict[str,str]] = []
for tokens, ner_tags in zip(X_test, y_test):
    tokens = [str(token) for token in tokens]
    ner_tags = [str(ner_tag) for ner_tag in ner_tags]

    test_data.append({
        "tokens": np.asarray(tokens),
```

```
            "ner_tags": np.asarray(ner_tags)
        })

    test_dataset = hf_Dataset.from_list(test_data)


    def test_preprocess_function(examples):
        tokenized_inputs = TOKENIZER(
            examples["tokens"],
            truncation=True,
            is_split_into_words=True,
            padding='max_length',
            max_length=128
        )

        labels = []
        for i, label_seq in enumerate(examples["ner_tags"]):
            word_ids = tokenized_inputs.word_ids(batch_index=i)
            label_ids = []
            previous_word_idx = None
            for word_idx in word_ids:
                if word_idx is None:
                    label_ids.append(-100)
                elif word_idx != previous_word_idx:
                    label = label_seq[word_idx]
                    label_id = label_to_id.get(label, -100)  # fallback if label is unknown
                    label_ids.append(label_id)
                else:
                    label_ids.append(-100)
                previous_word_idx = word_idx
            labels.append(label_ids)

        tokenized_inputs["labels"] = labels
        return tokenized_inputs



    tokenized_test_dataset = test_dataset.map(
        lambda examples: test_preprocess_function(examples),
        batched=True
    )
```

```
Map: 100%  15/15 [00:00<00:00, 319.65 examples/s]
```

```
    tokenized_test_dataset
```

```
Dataset({
     features: ['tokens', 'ner_tags', 'input_ids', 'token_type_ids', 'attention_mask', 'labels'],
     num_rows: 15
 })
```

```
    metric = load_metric("seqeval")

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

        true_predictions = [
            [id_to_label[pred] for pred, label in zip(pred_seq, label_seq) if label != -100]
            for pred_seq, label_seq in zip(predictions, labels)
        ]
        true_labels = [
            [id_to_label[label] for pred, label in zip(pred_seq, label_seq) if label != -100]
            for pred_seq, label_seq in zip(predictions, labels)
        ]

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



    id_to_label = {v: k for k, v in label_to_id.items()}


    eval_trainer = Trainer(
        model=model,
```

```
    tokenizer=TOKENIZER,
    compute_metrics=compute_metrics
)
```

```
<ipython-input-155-f218211a00a4>:1: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `T
  eval_trainer = Trainer(
```

```
results = eval_trainer.evaluate(tokenized_test_dataset)
print(results)
```

```
[2/2 00:00]
{'eval_loss': 1.0500532388687134, 'eval_model_preparation_time': 0.0155, 'eval_precision': 0.21052631578947367, 'eval_re
/usr/local/lib/python3.11/dist-packages/seqeval/metrics/v1.py:57: UndefinedMetricWarning: Precision and F-score are ill-
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.11/dist-packages/seqeval/metrics/v1.py:57: UndefinedMetricWarning: Recall and F-score are ill-def
  _warn_prf(average, modifier, msg_start, len(result))
```

```
predictions, labels, _ = eval_trainer.predict(tokenized_test_dataset)
pred_labels = np.argmax(predictions, axis=2)

for i in range(5):  # show 5 examples
    tokens = test_dataset[i]["tokens"]
    preds = [id_to_label[pred] for pred, label in zip(pred_labels[i], labels[i]) if label != -100]
    golds = [id_to_label[label] for pred, label in zip(pred_labels[i], labels[i]) if label != -100]

    print(f"TOKENS : {tokens}")
    print(f"PRED   : {preds}")
    print(f"GOLD   : {golds}")
    print("-" * 50)
```

```
TOKENS : ['If', "you're", 'visiting', 'Paris', ',', 'make', 'sure', 'to', 'see', 'the', 'Louvre', ',', 'as', 'they', 'ex
PRED   : ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']
GOLD   : ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-ORG', 'O', 'O', 'O', 'O', 'O', 'B-WORK_OF_ART', 'I-WORK_OF_ART
--------------------------------------------------
TOKENS : ['Amazon', ',', 'Google', 'and', 'Meta', 'control', 'a', 'huge', 'share', 'of', 'the', 'technology', 'market',
PRED   : ['O', 'O', 'O', 'O', 'B-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']
GOLD   : ['B-ORG', 'O', 'B-ORG', 'O', 'B-ORG', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']
--------------------------------------------------
TOKENS : ['Did', 'you', 'hear', 'Pharoah', 'Sanders', 'recorded', 'an', 'album', 'with', 'Floating', 'Points', '?']
PRED   : ['O', 'O', 'O', 'B-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']
GOLD   : ['O', 'O', 'O', 'B-PERSON', 'I-PERSON', 'O', 'O', 'O', 'O', 'B-PERSON', 'I-PERSON', 'O']
--------------------------------------------------
TOKENS : ['Madvillainy', 'is', 'still', 'my', 'favourite', 'MF', 'DOOM', 'record', '.']
PRED   : ['B-PERSON', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']
GOLD   : ['B-WORK_OF_ART', 'O', 'O', 'O', 'O', 'B-PERSON', 'I-PERSON', 'O', 'O']
--------------------------------------------------
TOKENS : ['My', 'friend', 'Kevin', 'just', 'finished', 'watching', 'Succession', ',', 'and', "won't", 'stop', 'talking',
PRED   : ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-ORG', 'I-PERSON', 'O', 'O', 'O']
GOLD   : ['O', 'O', 'B-PERSON', 'O', 'O', 'O', 'B-WORK_OF_ART', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-PERSON', 'I-PERSON', 'O
--------------------------------------------------
```

```
Begin met programmeren of genereer code met AI.
```