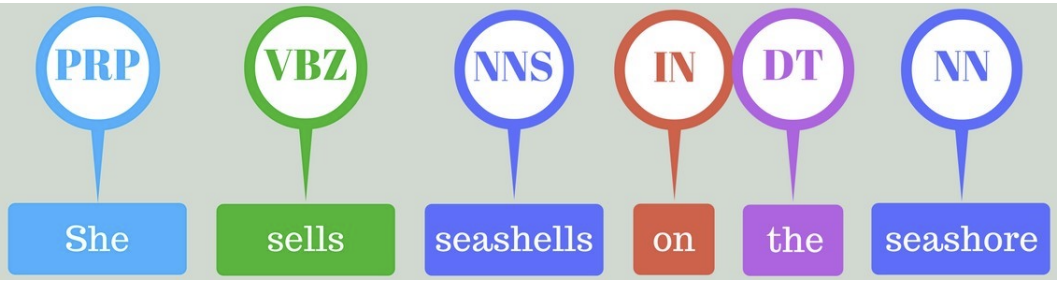
AI VIET NAM – COURSE 2024

Part-of-Speech Tagging
Medical Named Entity Recognition

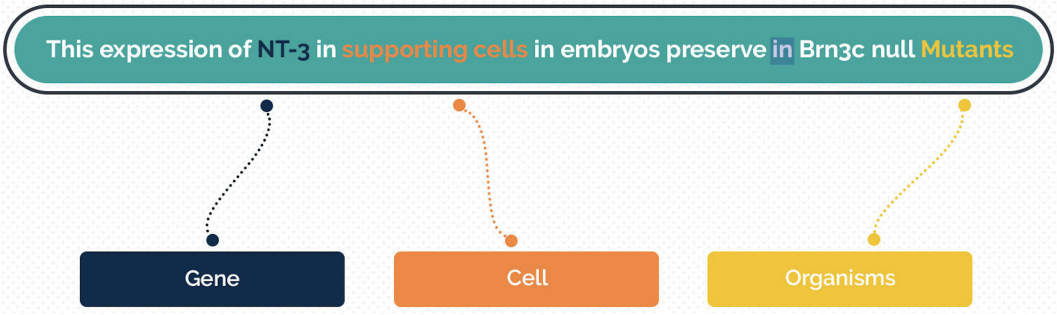
Quoc-Thai Nguyen và Quang-Vinh Dinh

Ngày 1 tháng 2 năm 2025

Phần I. Giới thiệu



Hình 1: Bài toán gán nhãn từ loại (Part-of-Speech Tagging).



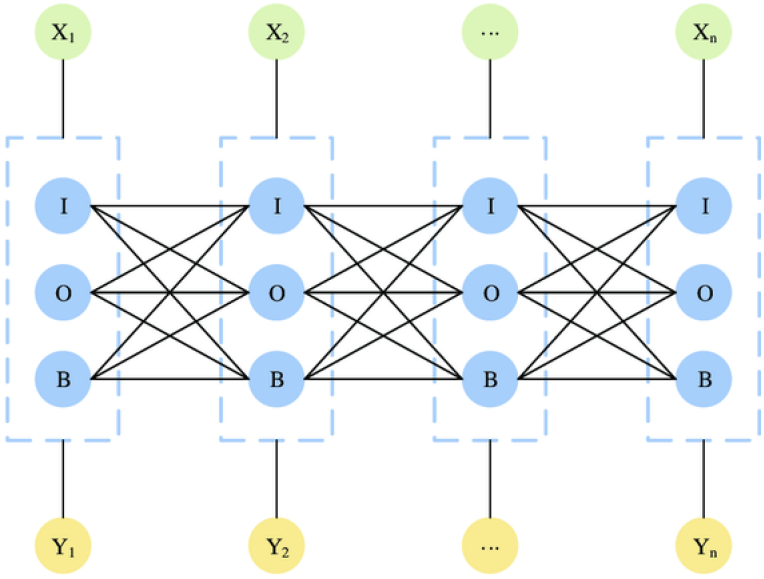
Hình 2: Bài toán nhận dạng thực thể cho dữ liệu y học (Medical Named Entity Recognition).

Part-of-Speech (POS) Tagging (Gán nhãn từ loại): là bài toán gán nhãn các từ trong câu tương ứng với từ loại thực hiện chức năng ngữ pháp của nó. Các từ loại được xây dựng và mở rộng phụ thuộc vào đặc trưng ngôn ngữ. Trong đó, một số từ loại điển hình thường xuất hiện trong hầu hết các ngôn ngữ trên thế giới hiện nay như: danh từ (Noun), tính từ (Adjective), động từ (Verb),... Ngoài các từ loại phổ biến này ra, dựa vào đặc trưng ngôn ngữ có thể xác định được tập các từ loại nhỏ hơn. Ví dụ với động từ trong tiếng anh (VERB) có thể phân chia thành: VB (Verb, base form - động từ nguyên thể), VBD (Verb, past tense - động từ quá khứ),... hoặc danh từ trong tiếng anh (NOUN) có thể được phân chia thành: NN (Noun, singular or mass - Danh từ số ít), NNS (Noun, plural - Danh từ số nhiều),...

Để hiểu rõ thêm về tập nhãn từ loại, các chuyên gia thường xây dựng và thống nhất cho các ngôn ngữ khác nhau, tập nhãn từ loại cơ bản thường là tập nhãn Penn Tree Bank. Ví dụ về POS Tagging được minh hoạ trong hình 1. Với câu đầu vào: "She sells seashells on the seashore", thông qua các mô hình gán nhãn sẽ gán nhãn từ loại cho các từ thành: "She" có nhãn từ loại là "PRP-Personal pronoun", "sells" có nhãn từ loại là "VBZ-Verb, 3rd person singular present",... đến "seashore" có nhãn từ loại là "NN-Noun, singular or mass".

Named Entity Recognition (NER - Nhận dạng thực thể): là bài toán xác định từ hoặc chuỗi từ trong văn bản là tên của một thực thể xác định, các thực thể điển hình như: tên người, tên địa danh, giới tính,... NER là bài toán có nhiều ứng dụng trong trích xuất các thông tin quan trọng thuộc nhiều lĩnh vực khác nhau. Đặc biệt là việc khai thác và trích xuất thông tin trong các bản ghi hồ sơ bệnh lĩnh vực y học (Medical NER). Ví dụ về Medical NER được minh hoạ trong hình 2. Với đoạn văn bản đầu vào: "This expression of NT-3 in supporting cells in embryos preserve in Brn3c null Mutants" thông qua mô hình nhận dạng thực thể sẽ xác định và trích xuất các thông tin quan trọng như: "NT-3" sẽ là thực thể "Gene", "supporting cells" sẽ là thực thể "Cell" và "Mutants" sẽ có thực thể là "Mutants".

Bài toán POS Tagging và NER thuộc nhóm bài toán phân loại mức từ (Token-level Text Classification). Nghĩa là với mỗi đơn vị văn bản đầu vào (các từ) sẽ được phân loại vào một lớp cụ thể (từ loại: danh từ, động từ,... trong bài POS Tagging hoặc thực thể trong bài NER)

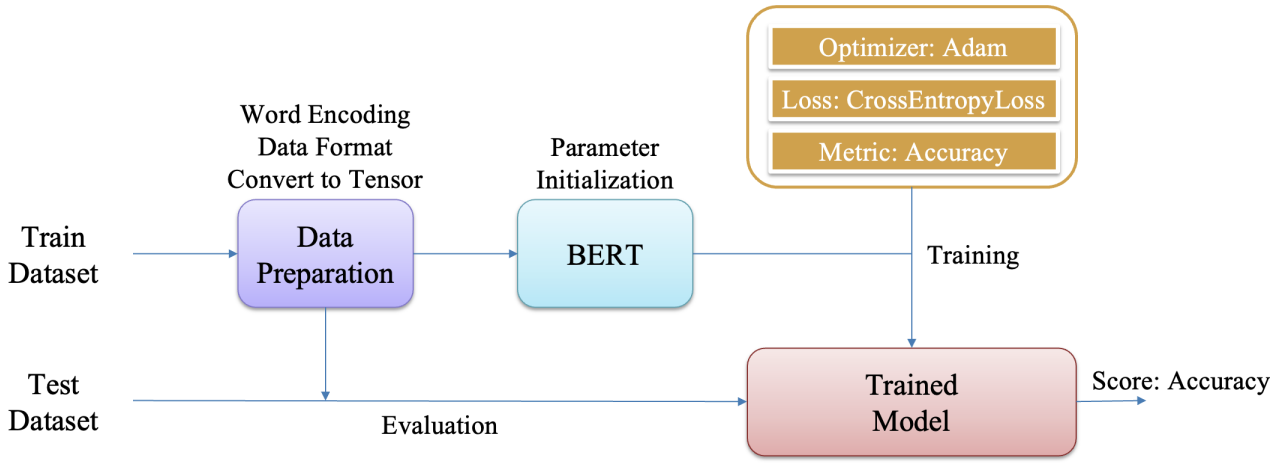


Hình 3: Token-Level Text Classification.

Phần II. Part-of-Speech Tagging

Ở trong phần này, chúng ta xây dựng mô hình POS Tagger trên bộ dữ liệu Penn Tree Bank của ngôn ngữ tiếng anh sử dụng pre-trained model như BERT,...

Quá trình huấn luyện và đánh giá mô hình POS Tagging dựa vào BERT được mô tả như sau"



Hình 4: POS Tagging Pipeline.

1. Load Dataset

```
# install library
pip install evaluate

# import library
from typing import List
import numpy as np
import torch
import evaluate
from sklearn.model_selection import train_test_split
import nltk
nltk.download('treebank')


# load tree bank dataset
tagged_sentences = nltk.corpus.treebank.tagged_sents()
print("Number of samples:", len(tagged_sentences))

# save sentences and tags
sentences, sentence_tags =[], []
for tagged_sentence in tagged_sentences:
    sentence, tags = zip(*tagged_sentence)
    sentences.append([word.lower() for word in sentence])
    sentence_tags.append([tag for tag in tags])
```

2. Preprocessing

- Split dataset into train, validation and test set

```
train_sentences, test_sentences, train_tags, test_tags = train_test_split(
    sentences,
    sentence_tags,
```

```
    test_size=0.3
)

valid_sentences, test_sentences, valid_tags, test_tags = train_test_split(
    test_sentences,
    test_tags,
    test_size=0.5
)
```

- Build dataset

```
# tokenization
from transformers import AutoTokenizer
from torch.utils.data import Dataset

model_name = "QCRI/bert-base-multilingual-cased-pos-english"

tokenizer = AutoTokenizer.from_pretrained(
    model_name,
    use_fast=True
)
MAX_LEN = 256
class PosTagging_Dataset(Dataset):
    def __init__(self,
                 sentences: List[List[str]],
                 tags: List[List[str]],
                 tokenizer,
                 label2id,
                 max_len=MAX_LEN
                ):
        super().__init__()
        self.sentences = sentences
        self.tags = tags
        self.max_len = max_len
        self.tokenizer = tokenizer
        self.label2id = label2id

    def __len__(self):
        return len(self.sentences)

    def __getitem__(self, idx):
        input_token = self.sentences[idx]
        label_token = self.tags[idx]

        input_token = self.tokenizer.convert_tokens_to_ids(input_token)
        attention_mask = [1] * len(input_token)
        labels = [self.label2id[token] for token in label_token]

        return {
            "input_ids": self.pad_and_truncate(input_token, pad_id=self.tokenizer.
    pad_token_id),
            "labels": self.pad_and_truncate(labels, pad_id=label2id["0"]),
            "attention_mask": self.pad_and_truncate(attention_mask, pad_id=0)
        }

    def pad_and_truncate(self, inputs: List[int], pad_id: int):
        if len(inputs) < self.max_len:
            padded_inputs = inputs + [pad_id] * (self.max_len - len(inputs))
        else:
            padded_inputs = inputs[:self.max_len]
        return torch.as_tensor(padded_inputs)
```

- Dataset loader

```
train_dataset = PosTagging_Dataset(train_sentences, train_tags, tokenizer, label2id)
val_dataset = PosTagging_Dataset(valid_sentences, valid_tags, tokenizer, label2id)
test_dataset = PosTagging_Dataset(test_sentences, test_tags, tokenizer, label2id)
```

3. Modeling

```
from transformers import AutoTokenizer, AutoModelForTokenClassification

model_name = "QCRI/bert-base-multilingual-cased-pos-english"

model = AutoModelForTokenClassification.from_pretrained(model_name)
```

4. Metric

```
accuracy = evaluate.load("accuracy")

ignore_label = len(label2id)

def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    mask = labels != ignore_label
    predictions = np.argmax(predictions, axis=-1)
    return accuracy.compute(predictions=predictions[mask], references=labels[mask])
```

5. Trainer

```
from transformers import TrainingArguments, Trainer

training_args = TrainingArguments(
    output_dir="out_dir",
    learning_rate=1e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=10,
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    tokenizer = tokenizer,
    compute_metrics=compute_metrics,
)

trainer.train()
```

6. Training

Kết quả training và accuracy trên tập test là 99.09

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	0.051634	0.985287
2	No log	0.041280	0.987875
3	0.150500	0.037712	0.988907
4	0.150500	0.035710	0.989532
5	0.150500	0.034171	0.989985
6	0.030800	0.033480	0.990071
7	0.030800	0.033291	0.990371
8	0.030800	0.032909	0.990437
9	0.024600	0.032835	0.990484
10	0.024600	0.032753	0.990457

Hình 5: Quá trình huấn luyện mô hình POS Tagging trên bộ dữ liệu Penn Tree Bank.

7. Inference

Sau quá trình huấn luyện và mô hình có độ đo đánh giá tốt nhất được sử dụng để gán nhãn cho các câu đầu vào

```
# tokenization
test_sentence = "We are exploring the topic of deep learning"
input = torch.as_tensor([tokenizer.convert_tokens_to_ids(test_sentence.split())])
input = input.to("cuda")

# prediction
outputs = model(input)
_, preds = torch.max(outputs.logits, -1)
preds = preds[0].cpu().numpy()

# decode
pred_tags = ""
for pred in preds:
    pred_tags += id2label[pred] + " "
pred_tags # => PRP VBP RB DT NN IN JJ NN
```

Phần III. Medical NER

Ở trong phần này chúng ta sẽ xây dựng mô hình nhận dạng các thực thể trong y học dựa trên bộ dữ liệu MACCROBAT2018.

Quay trở lại với ví dụ ở hình 2. Với đoạn văn bản đầu vào: "This expression of NT-3 in supporting cells in embryos preserve in Brn3c null Mutants" thông qua mô hình gán nhãn thực thể sẽ xác định và trích xuất các thông tin quan trọng như: "NT-3" sẽ là thực thể "Gene", "supporting cells" sẽ là thực thể "Cell" và "Mutants" sẽ có thực thể là "Mutants". Vì vậy, kết quả của mô hình sẽ có những từ không thuộc vào thực thể nào hoặc một thực thể có thể chứa nhiều từ. Nhưng mô hình giải quyết cho bài toán Token-Level Text Classification sẽ gán nhãn cho mỗi từ (hoặc gọi là token) thuộc vào một nhãn cụ thể. Cho nên, để giải quyết vấn đề này, chúng ta tần xây dựng một tập nhãn mới và quy bài toán NER về thành chuẩn bài toán Token-Level Text Classification.

Tập nhãn mới được xác định gồm:

1. Với những từ không thuộc vào thực thể nào sẽ được gán nhãn thành nhãn: "O"
2. Với những thực thể có thể có một hoặc nhiều từ đi kèm, chúng ta tạo ra tập nhãn BI (hoặc I, BIE). Ví dụ, thực thể "Cell" sẽ tạo thành tập nhãn bao gồm: "B-Cell" (Xác định vị trí từ bắt đầu của thực thể), "I-Cell" (Xác định vị trí của các từ theo sau vị trí bắt đầu thực thể) hoặc có thêm "E-Cell" (Xác định vị trí kết thúc của thực thể).

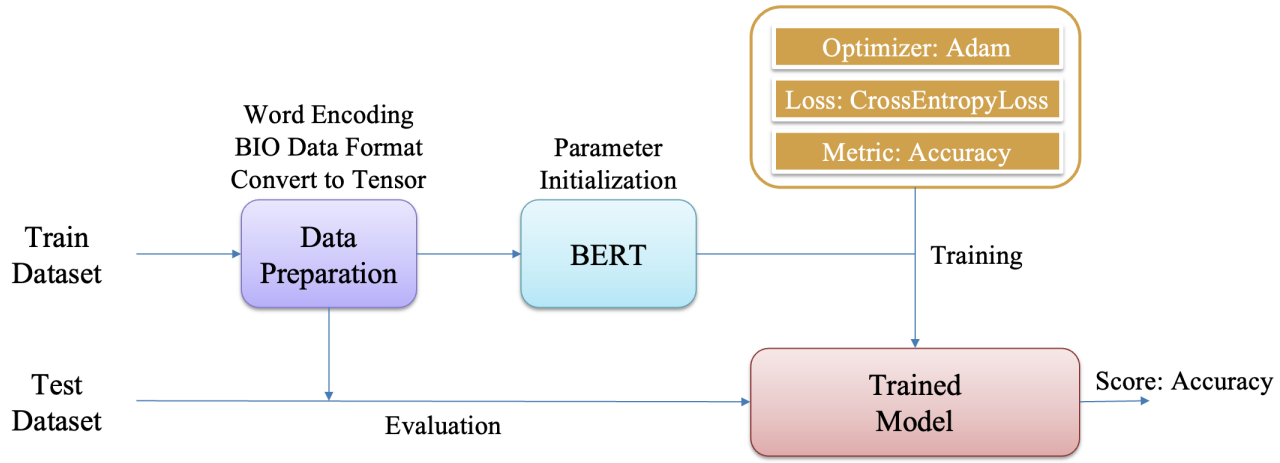
   Vì vậy chúng ta có các phương pháp xác định tập nhãn mới có thể là BIO, hoặc IO, hoặc BIOE.

Vì vậy với bộ dữ liệu với N thực thể, chúng ta sẽ tạo thành bộ nhãn mới gồm: 2*N (Nhãn B và I cho mỗi thực thể) + 1 (tương ứng nhãn O). Ví dụ với câu trên có 3 thực thể "Gene", "Cell", "Mutants" sẽ tạo thành 7 nhãn mới: "B-Gene", "I-Gene", "B-Cell", "I-Cell", "B-Mutants", "I-Mutants" và "O".

Do đó, bài toán chuyển thành, với đoạn văn bản đầu vào: "This expression of NT-3 in supporting cells in embryos preserve in Brn3c null Mutants" sẽ được gán nhãn thành: "O O O B-Gene O B-Cell I-Cell O O O O O O B-Mutants". Vì vậy, chúng ta sẽ quy về bài toán Token-Level Text Classification và xây dựng mô hình tương tự như bài toán POS Tagging.

Tiếp theo, chúng ta sẽ xây dựng mô hình dựa trên pre-trained model như BERT,... sau khi đã tải về bộ dữ liệu MACCROBAT2018.

Quá trình huấn luyện và đánh giá mô hình POS Tagging dựa vào BERT được mô tả như sau"



Hình 6: Named Entity Recognition Pipeline.

1. Load Dataset

Sau khi tải về bộ dữ liệu, chúng ta sẽ thu được folder có cấu trúc như sau:

Hình 7: Cấu trúc thư mục bộ dữ liệu MACCROBAT2018.

Mỗi sample tương ứng gồm 2 file: các file có đuôi "...txt", chứa đoạn văn bản và các file có đuôi"...ann" chứa các nhãn thực thể tương ứng được mô tả theo chuẩn BioNLP Shared Task standoff format được mô tả cụ thể như hình sau:

```
T1          Organization 0 4                Sony
T2          MERGE-ORG 14 27                 joint venture
T3          Organization 33 41              Ericsson
E1          MERGE-ORG:T2 Org1:T1 Org2:T3
T4          Country 75 81                   Sweden
R1          Origin Arg1:T3 Arg2:T4
```

Hình 8: Ví dụ về mô tả các thực thể và mối quan hệ tương ứng trong bộ dữ liệu MACCROBAT2018.

Với cột đầu tiên: T đại diện cho Entity, E đại diện cho Event và R đại diện cho Relation. Vì vậy, trong bài toán Medical NER chúng ta sẽ sử dụng các nhãn T.

Tương ứng với các thực thể T sẽ có cột thứ 2 là tên thực thể, ví dụ Organization, cột thứ 3 sẽ là vị trí trong văn bản đầu vào lấy từ file có đuôi ".txt", cột thứ 4 sẽ là đoạn văn bản tương ứng.

2. Preprocessing

Vì các nhãn thực thể được mô tả theo chuẩn standoff, nên trước tiên chúng ta cần tiền xử lý để lấy ra các vị trí tương ứng trong văn bản đầu vào và xây dựng tập nhãn "BIO" tương ứng.

```python
import os
from typing import List, Dict, Tuple

class Preprocessing_Maccrobat:
    def __init__(self, dataset_folder, tokenizer):
        self.file_ids = [f.split(".")[0] for f in os.listdir(dataset_folder) if f.endswith('.txt')]

        self.text_files = [f+".txt" for f in self.file_ids]
        self.anno_files = [f+".ann" for f in self.file_ids]

        self.num_samples = len(self.file_ids)

        self.texts: List[str] = []
```

```
        for i in range(self.num_samples):
            file_path = os.path.join(dataset_folder, self.text_files[i])
            with open(file_path, 'r') as f:
                self.texts.append(f.read())

        self.tags: List[Dict[str, str]] = []
        for i in range(self.num_samples):
            file_path = os.path.join(dataset_folder, self.anno_files[i])
            with open(file_path, 'r') as f:
                text_bound_ann = [t.split("\t") for t in f.read().split("\n") if t.
    startswith("T")]
                text_bound_lst = []
                for text_b in text_bound_ann:
                    label = text_b[1].split(" ")
                    try:
                        _ = int(label[1])
                        _ = int(label[2])
                        tag = {
                            "text": text_b[-1],
                            "label": label[0],
                            "start": label[1],
                            "end": label[2]
                        }
                        text_bound_lst.append(tag)
                    except:
                        pass

                self.tags.append(text_bound_lst)
        self.tokenizer = tokenizer

    def process(self) -> Tuple[List[List[str]], List[List[str]]]:
        input_texts = []
        input_labels = []

        for idx in range(self.num_samples):
            full_text = self.texts[idx]
            tags = self.tags[idx]

            label_offset = []
            continuous_label_offset = []
            for tag in tags:
                offset = list(range(int(tag["start"]), int(tag["end"])+1))
                label_offset.append(offset)
                continuous_label_offset.extend(offset)

            all_offset = list(range(len(full_text)))
            zero_offset = [offset for offset in all_offset if offset not in
    continuous_label_offset]
            zero_offset = Preprocessing_Maccrobat.find_continuous_ranges(zero_offset)

            self.tokens = []
            self.labels = []
            self._merge_offset(full_text, tags, zero_offset, label_offset)
            assert len(self.tokens) == len(self.labels), f"Length of tokens and labels
     are not equal"

            input_texts.append(self.tokens)
            input_labels.append(self.labels)

        return input_texts, input_labels
```

```
71
72      def _merge_offset(self, full_text, tags, zero_offset, label_offset):
73          i = j = 0
74          while i < len(zero_offset) and j < len(label_offset):
75              if zero_offset[i][0] < label_offset[j][0]:
76                  self._add_zero(full_text, zero_offset, i)
77                  i += 1
78              else:
79                  self._add_label(full_text, label_offset, j, tags)
80                  j += 1
81
82          while i < len(zero_offset):
83              self._add_zero(full_text, zero_offset, i)
84              i += 1
85
86          while j < len(label_offset):
87              self._add_label(full_text, label_offset, j, tags)
88              j += 1
89
90      def _add_zero(self, full_text, offset, index):
91          start, *_ ,end =  offset[index] if len(offset[index]) > 1 else (offset[index
    ][0], offset[index][0]+1)
92          text = full_text[start:end]
93          text_tokens = self.tokenizer.tokenize(text)
94
95          self.tokens.extend(text_tokens)
96          self.labels.extend(
97              ["O"]*len(text_tokens)
98          )
99
100     def _add_label(self, full_text, offset, index, tags):
101         start, *_ ,end =  offset[index] if len(offset[index]) > 1 else (offset[index
    ][0], offset[index][0]+1)
102         text = full_text[start:end]
103         text_tokens = self.tokenizer.tokenize(text)
104
105         self.tokens.extend(text_tokens)
106         self.labels.extend(
107             [f"B-{tags[index]['label']}"] + [f"I-{tags[index]['label']}"]*(len(
    text_tokens)-1)
108         )
109
110     @staticmethod
111     def build_label2id(tokens: List[List[str]]):
112         label2id = {}
113         id_counter = 0
114         for token in [token for sublist in tokens for token in sublist]:
115             if token not in label2id:
116                 label2id[token] = id_counter
117                 id_counter += 1
118         return label2id
119
120     @staticmethod
121     def find_continuous_ranges(data: List[int]):
122         if not data:
123             return []
124         ranges = []
125         start = data[0]
126         prev = data[0]
127         for number in data[1:]:
```

```
            if number != prev + 1:
                ranges.append(list(range(start, prev + 1)))
                start = number
            prev = number
        ranges.append(list(range(start, prev + 1)))
        return ranges

# Preprocessing
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("d4data/biomedical-ner-all")

dataset_folder = "./MACCROBAT2018"

Maccrobat_builder = Preprocessing_Maccrobat(dataset_folder, tokenizer)
input_texts, input_labels = Maccrobat_builder.process()

label2id = Preprocessing_Maccrobat.build_label2id(input_labels)
id2label = {v: k for k, v in label2id.items()}


# Split
from sklearn.model_selection import train_test_split

inputs_train, inputs_val, labels_train, labels_val = train_test_split(
    input_texts,
    input_labels,
    test_size=0.2,
    random_state=42
)
```

3. Dataloader

```
import torch
from torch.utils.data import Dataset

MAX_LEN = 512

class NER_Dataset(Dataset):
    def __init__(self, input_texts, input_labels, tokenizer, label2id, max_len=MAX_LEN
    ):
        super().__init__()
        self.tokens = input_texts
        self.labels = input_labels
        self.tokenizer = tokenizer
        self.label2id = label2id
        self.max_len = max_len

    def __len__(self):
        return len(self.tokens)

    def __getitem__(self, idx):
        input_token = self.tokens[idx]
        label_token = [self.label2id[label] for label in self.labels[idx]]

        input_token = self.tokenizer.convert_tokens_to_ids(input_token)
        attention_mask = [1] * len(input_token)

        input_ids = self.pad_and_truncate(input_token, pad_id= self.tokenizer.
    pad_token_id)
```

```
        labels = self.pad_and_truncate(label_token, pad_id=0)
        attention_mask =  self.pad_and_truncate(attention_mask, pad_id=0)

        return {
            "input_ids": torch.as_tensor(input_ids),
            "labels": torch.as_tensor(labels),
            "attention_mask": torch.as_tensor(attention_mask)
            }

    def pad_and_truncate(self, inputs: List[int], pad_id: int):
        if len(inputs) < self.max_len:
            padded_inputs = inputs + [pad_id] * (self.max_len - len(inputs))
        else:
            padded_inputs = inputs[:self.max_len]
        return padded_inputs

    def label2id(self, labels: List[str]):
        return [self.label2id[label] for label in labels]

train_set = NER_Dataset(inputs_train, labels_train, tokenizer, label2id)
val_set = NER_Dataset(inputs_val, labels_val, tokenizer, label2id)
```

4. Modeling

```
from transformers import AutoModelForTokenClassification
model = AutoModelForTokenClassification.from_pretrained(
    "d4data/biomedical-ner-all",
    label2id=label2id,
    id2label=id2label,
    ignore_mismatched_sizes=True
)
```

5. Metric

```
import evaluate
import numpy as np

accuracy = evaluate.load("accuracy")

def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    mask = labels != 0
    predictions = np.argmax(predictions, axis=-1)
    return accuracy.compute(predictions=predictions[mask], references=labels[mask])
```

6. Trainer

```
from transformers import TrainingArguments, Trainer

training_args = TrainingArguments(
    output_dir="out_dir",
    learning_rate=1e-4,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=20,
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    optim="adamw_torch"
```

```
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_set,
    eval_dataset=val_set,
    tokenizer = tokenizer,
    compute_metrics=compute_metrics,
)

trainer.train()
```

7. Training

Kết quả training và accuracy trên tập validation là 78.5%

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	1.474108	0.323930
2	No log	0.900565	0.600812
3	No log	0.721831	0.694619
4	No log	0.610314	0.738286
5	No log	0.573626	0.760023
6	No log	0.567352	0.763211
7	No log	0.563499	0.765916
8	No log	0.558194	0.774804
9	No log	0.562186	0.777026
10	No log	0.565547	0.779828

Hình 9: Quá trình huấn luyện trên bộ dữ liệu MACCROBAT2018.

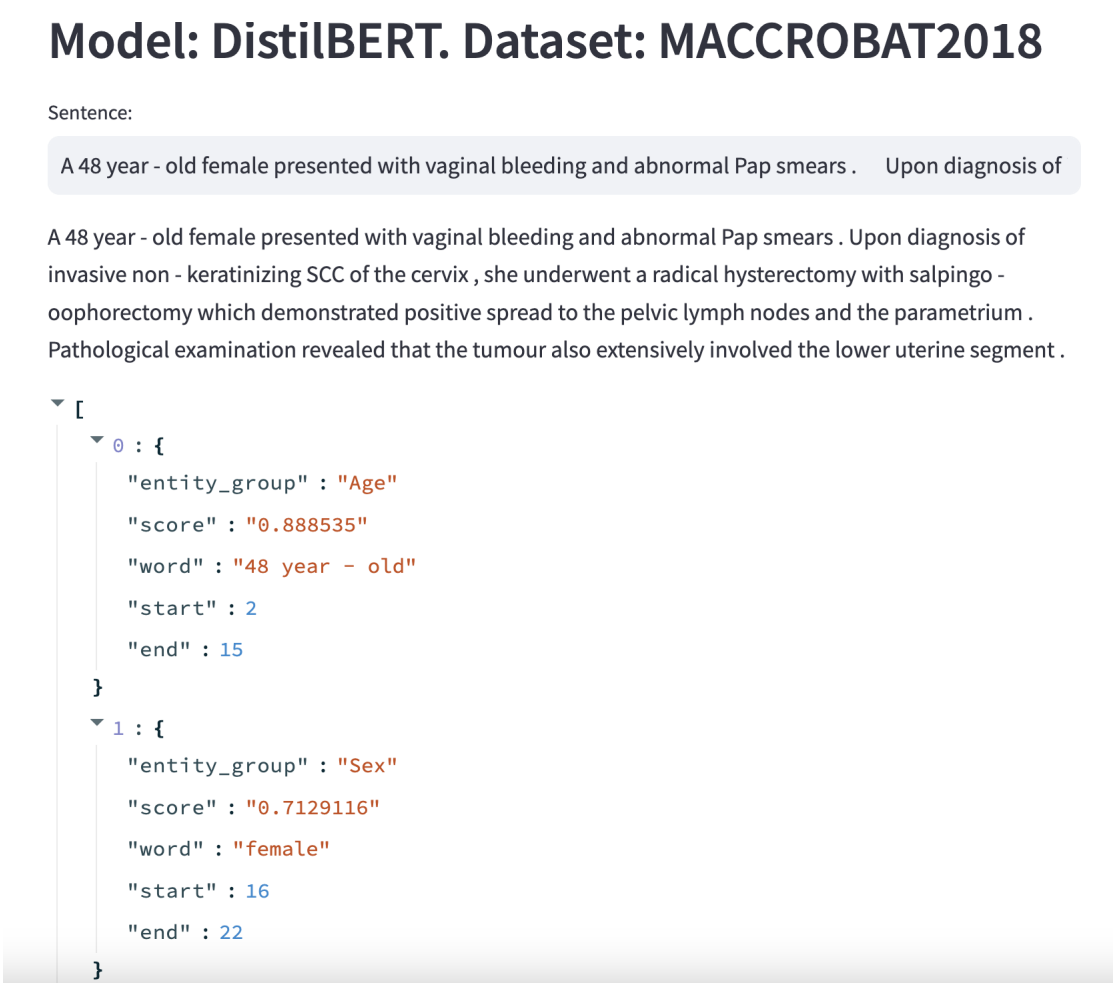
8. Inference

```
test_sentence = """A 48 year-old female presented with vaginal bleeding and abnormal
    Pap smears.  Upon diagnosis of invasive non-keratinizing SCC of the cervix,  she
    underwent a radical hysterectomy with salpingo-oophorectomy which demonstrated
    positive spread to the pelvic lymph nodes and the parametrium.  Pathological
    examination revealed that the tumour also extensively involved the lower uterine
    segment. """

# tokenization
input = torch.as_tensor([tokenizer.convert_tokens_to_ids(test_sentence.split())])

input = input.to("cuda")

# prediction
outputs = model(input)
_, preds = torch.max(outputs.logits, -1)
preds = preds[0].cpu().numpy()
```

```
12
13 # decode
14 for token, pred in zip(test_sentence.split(), preds):
15     print(f"{token}\t{id2label[pred]}")
```

9. Deployment

Triển khai ứng dụng sử dụng streamlit tham khảo mã nguồn tại đây, thử nghiệm ứng dụng tại đây. Giao diện và kết quả:



Hình 10: Triển khai ứng dụng.

Phần 4. Câu hỏi trắc nghiệm

Câu hỏi 1 Nhiệm vụ của bài toán gán nhãn từ loại (Part-of-Speech Tagging) là?
a) Phân tích cảm xúc
b) Dịch máy
c) Hỏi đáp
d) Xác định loại từ của các từ trong câu

Câu hỏi 2 Bộ dữ liệu nào sau đây được sử dụng để huấn luyện mô hình POS Tagging?
a) IMDB-Review
b) Penn Tree Bank
c) BioNLP
d) SQuAD

Câu hỏi 3 Số lượng sample của bộ dữ liệu sử dụng để huấn luyện mô hình POS Tagging trong phần thực nghiệm là?
a) 3914
b) 3915
c) 3916
d) 3917

Câu hỏi 4 Số lượng nhãn từ loại được sử dụng trong xây dựng mô hình POS Tagging là?
a) 45
b) 46
c) 47
d) 48

Câu hỏi 5 Mục đích của bài toán Named Entity Recognition là gì?
a) Xác định các đoạn văn bản chứa các thực thể
b) Xác định loại từ của các từ trong câu
c) Cả 2 đáp án a và b đều đúng
d) Cả 2 đáp án a và b đều sai

Câu hỏi 6 Với N thực thể và sử dụng các đánh nhãn "BIO" thì số nhãn mới thu được sẽ là bao nhiêu?
a) 2*N + 4
b) 2*N + 3
c) 2*N + 2
d) 2*N + 1

Câu hỏi 7 Với N thực thể và sử dụng các đánh nhãn "BIOE" thì số nhãn mới thu được sẽ là bao nhiêu?
a) 3*N + 4

b) 3*N + 3
c) 3*N + 2
d) 3*N + 1

Câu hỏi 8 Bộ dữ liệu sử dụng cho nhận dạng thực thể trong y học được sử dụng trong phần thực nghiệm là?
a) NCBI-Disease
b) BioNLP13
c) MACCROBAT2018
d) BC5CDR

Câu hỏi 9 Mô hình tiền huấn luyện được sử dụng cho bài toán Medical NER là?
a) biomedical-base
b) d4data/biomedical-ner-all
c) bert-base-uncased
d) bert-large

Câu hỏi 10 Số lượng nhãn "BIO" được sử dụng cho bộ dữ liệu MACCROBAT2018 là?
a) 81
b) 82
c) 83
d) 84

Phần 5. Phụ lục

1. **Hint**: Dựa vào file tải về tại đây để hoàn thiện các đoạn code.
2. **Solution**: Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải về tại đây (Lưu ý: Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Rubric**:

Phần	**Kiến Thức**	**Đánh Giá**
1	- Hiểu rõ bài toán POS Tagging - Sử dụng pretrained model cho POS Tagging	- Xây dựng mô hình POS Tagging trên bộ Penn Tree Bank
2.	- Hiểu rõ bài toán Named Entity Recognition - Sử dụng pretrained model cho bài toán NER	- Tiền xử lý dữ liệu cho bài toán NER - Xây dựng mô hình NER trên bộ dữ liệu y học MACCROBAT2018

- Hết -