Lab 6: Thực hành chuyên sâu về Phân tích cú pháp phụ thuộc (Dependency Parsing)

Mục tiêu

Sau buổi thực hành này, bạn sẽ có khả năng:

1. Sử dụng thư viện `spaCy` để thực hiện phân tích cú pháp phụ thuộc cho một câu.
2. Trực quan hóa cây phụ thuộc để hiểu rõ cấu trúc câu.
3. Truy cập và duyệt (traverse) cây phụ thuộc theo chương trình.
4. Trích xuất thông tin có ý nghĩa từ các mối quan hệ phụ thuộc (ví dụ: tìm chủ ngữ, tân ngữ, bổ ngữ).

Yêu cầu

- Python 3.6+
- Thư viện `spaCy` và mô hình ngôn ngữ tiếng Anh.

Phần 1: Giới thiệu và Cài đặt

Phân tích cú pháp phụ thuộc là một kỹ thuật nền tảng cho phép chúng ta hiểu cấu trúc ngữ pháp của câu dưới dạng các mối quan hệ `head` (điều khiển) và `dependent` (phụ thuộc). Trong bài thực hành này, chúng ta sẽ sử dụng `spaCy`, một thư viện NLP công nghiệp, để khám phá kỹ thuật này.

Cài đặt:

Mở terminal và chạy các lệnh sau:

```
# Cài đặt spaCy
pip install -U spacy

# Tải về mô hình tiếng Anh (kích thước trung bình, có đủ thông tin cho parsing)
python -m spacy download en_core_web_md
```

Phần 2: Phân tích câu và Trực quan hóa

Trực quan hóa là cách tốt nhất để bắt đầu hiểu về cây phụ thuộc. `spaCy` cung cấp một công cụ tuyệt vời tên là `displaCy`.

2.1. Tải mô hình và phân tích câu

```
import spacy
from spacy import displacy
```

```
# Tải mô hình tiếng Anh đã cài đặt
# Sử dụng en_core_web_md vì nó chứa các vector từ và cây cú pháp đầy đủ
nlp = spacy.load("en_core_web_md")

# Câu ví dụ
text = "The quick brown fox jumps over the lazy dog."

# Phân tích câu với pipeline của spaCy
doc = nlp(text)
```

2.2. Trực quan hóa cây phụ thuộc

`displaCy` có thể khởi chạy một web server nhỏ để hiển thị kết quả.

```
# Tùy chọn để hiển thị trong trình duyệt
# options = {"compact": True, "color": "blue", "font": "Source Sans Pro"}

# Khởi chạy server tại http://127.0.0.1:5000
# Bạn có thể truy cập địa chỉ này trên trình duyệt để xem cây phụ thuộc
# Nhấn Ctrl+C trong terminal để dừng server
displacy.serve(doc, style="dep")
```

Sau khi chạy đoạn code trên, hãy mở trình duyệt và truy cập `http://127.0.0.1:5000`.

Câu hỏi:

- Từ nào là gốc (ROOT) của câu?
- `jumps` có những từ phụ thuộc (dependent) nào? Các quan hệ đó là gì?
- `fox` là `head` của những từ nào?

Phần 3: Truy cập các thành phần trong cây phụ thuộc

Trực quan hóa rất hữu ích, nhưng sức mạnh thực sự đến từ việc truy cập cây phụ thuộc theo chương trình. Mỗi `Token` trong đối tượng `Doc` của spaCy chứa đầy đủ thông tin về vị trí của nó trong cây.

Hãy phân tích các thuộc tính quan trọng của một token:

```
# Lấy một câu khác để phân tích
text = "Apple is looking at buying U.K. startup for $1 billion"
doc = nlp(text)

# In ra thông tin của từng token
print(f"{'TEXT':<12} | {'DEP':<10} | {'HEAD TEXT':<12} | {'HEAD POS':<8} | {'CHILDREN'}")
print("-" * 70)
```

```python
for token in doc:
    # Trích xuất các thuộc tính
    children = [child.text for child in token.children]

    print(f"{token.text:<12} | {token.dep_:<10} | {token.head.text:<12} | {token.head.pos_:<
```

Kết quả mong đợi:

```
TEXT         | DEP        | HEAD TEXT    | HEAD POS | CHILDREN
---------------------------------------------------------------------
Apple        | nsubj      | looking      | VERB     | []
is           | aux        | looking      | VERB     | []
looking      | ROOT       | looking      | VERB     | ['Apple', 'is', 'at', '.']
at           | prep       | looking      | VERB     | ['buying']
buying       | pcomp      | at           | ADP      | ['startup']
U.K.         | compound   | startup      | NOUN     | []
startup      | dobj       | buying       | VERB     | ['U.K.', 'for']
for          | prep       | startup      | NOUN     | ['billion']
$            | quantmod   | billion      | NUM      | []
1            | compound   | billion      | NUM      | []
billion      | pobj       | for          | ADP      | ['$', '1']
```

Giải thích các thuộc tính:

- `token.text`: Văn bản của token.
- `token.dep_`: Nhãn quan hệ phụ thuộc của token này với head của nó.
- `token.head.text`: Văn bản của token `head`.
- `token.head.pos_`: Part-of-Speech tag của token `head`.
- `token.children`: Một iterator chứa các token con (dependent) của token hiện tại.

Phần 4: Duyệt cây phụ thuộc để trích xuất thông tin

Bây giờ, chúng ta sẽ sử dụng các thuộc tính đã học để giải quyết các bài toán cụ thể.

4.1. Bài toán: Tìm chủ ngữ và tân ngữ của một động từ

Chúng ta muốn tìm các cặp `(chủ ngữ, động từ, tân ngữ)` trong câu.

```python
text = "The cat chased the mouse and the dog watched them."
doc = nlp(text)

for token in doc:
    # Chỉ tìm các động từ
    if token.pos_ == "VERB":
        verb = token.text
        subject = ""
```

```
        obj = ""

        # Tìm chủ ngữ (nsubj) và tân ngữ (dobj) trong các con của động từ
        for child in token.children:
            if child.dep_ == "nsubj":
                subject = child.text
            if child.dep_ == "dobj":
                obj = child.text

        if subject and obj:
            print(f"Found Triplet: ({subject}, {verb}, {obj})")
```

Kết quả mong đợi:

```
Found Triplet: (cat, chased, mouse)
```

4.2. Bài toán: Tìm các tính từ bổ nghĩa cho một danh từ

```
text = "The big, fluffy white cat is sleeping on the warm mat."
doc = nlp(text)

for token in doc:
    # Chỉ tìm các danh từ
    if token.pos_ == "NOUN":
        adjectives = []
        # Tìm các tính từ bổ nghĩa (amod) trong các con của danh từ
        for child in token.children:
            if child.dep_ == "amod":
                adjectives.append(child.text)

        if adjectives:
            print(f"Danh từ '{token.text}' được bổ nghĩa bởi các tính từ: {adjectives}")
```

Kết quả mong đợi:

```
Danh từ 'cat' được bổ nghĩa bởi các tính từ: ['big', 'fluffy', 'white']
Danh từ 'mat' được bổ nghĩa bởi các tính từ: ['warm']
```

Phần 5: Bài tập tự luyện

Bài 1: Tìm động từ chính của câu

Động từ chính của câu thường có quan hệ `ROOT`. Viết một hàm `find_main_verb(doc)` nhận vào một đối tượng `Doc` của spaCy và trả về `Token` là động từ chính.

Bài 2: Trích xuất các cụm danh từ (Noun Chunks)

`spaCy` đã có sẵn thuộc tính `.noun_chunks` để trích xuất các cụm danh từ. Tuy nhiên, hãy thử tự viết một hàm để làm điều tương tự.

Gợi ý: Một cụm danh từ đơn giản là một danh từ và tất cả các từ bổ nghĩa cho nó (như `det`, `amod`, `compound`). Bạn có thể bắt đầu từ một danh từ và duyệt ngược lên `head` hoặc duyệt xuống các `children` của nó.

Bài 3: Tìm đường đi ngắn nhất trong cây

Viết một hàm `get_path_to_root(token)` để tìm đường đi từ một token bất kỳ lên đến gốc (ROOT) của cây. Hàm nên trả về một danh sách các token trên đường đi.

Tổng kết

Trong bài thực hành này, chúng ta đã đi từ những khái niệm cơ bản đến các ứng dụng thực tế của phân tích cú pháp phụ thuộc. Bạn đã học cách sử dụng `spaCy` để phân tích câu, trực quan hóa kết quả, và quan trọng nhất là duyệt cây phụ thuộc để trích xuất các thông tin ngữ nghĩa quan trọng. Nắm vững kỹ thuật này sẽ mở ra rất nhiều khả năng cho các ứng dụng NLP phức tạp hơn.