PERSONAL NOTE – WEEK 02 (16.09 – 21.09)

TEXT PREPROCESSING

XỬ LÝ VĂN BẢN VỚI NLTK

Nội dung

1. Truy cập văn bản từ Web và Disk

a. Truy cập từ Web

- Văn bản được sử dụng có nguồn gốc đa dạng, phong phú nhất vẫn là Web. Các văn bản này thường được truy cập thông qua **URL**:
    - Trong Python, thư viện **urllib** sẽ hỗ trợ các phương thức để giúp ta truy cập text bằng URL, đọc, mã hóa nó,...
    - Dữ liệu sau khi được lấy từ URL sẽ được Python lưu trữ dưới kiểu String.
- Một số loại văn bản thường được lấy sẽ nằm ở dạng:
    - Electronic Book: Đối với văn bản được lấy từ đây, văn bản sẽ tồn tại một số thông tin không cần thiết (gọi là **nhiễu**) như thông tin tác giả, năm xuất bản, bản quyền,… nằm ở phần lớn là **header** và **footer**. Do đó, việc loại bỏ chúng để lấy nội dung gốc của văn bản là cần thiết (điều này có thể thực hiện bằng phương pháp Slicing của Python, tuy nhiên cần phải biết được vị trí của chúng trong văn bản để cho ra kết quả tốt nhất).
        - Trong trường hợp này, có thể sử dụng một số hàm như **rfind()** hoặc **find()** để tìm vị trí của header và footer.
    - HTML: Các văn bản được lấy từ trang web hầu hết thường tồn tại dưới dạng HTML. Tuy nhiên, với cấu trúc của file HTML, khi văn bản được lấy về, nội dung sẽ được dính kèm với các thẻ như <p>, <a>,…
        - Một phương pháp để lấy được nội dung văn bản ra khỏi các thẻ là sự hỗ trợ của thư viện **BeautifulSoup**.
        - Dù văn bản đã được lấy ra khỏi các thẻ, tuy nhiên, nó vẫn sẽ tồn tại một số nhiễu khác (như ở Electronic Book). Vì thế, việc tìm kiếm và chỉnh sửa để lấy được nội dung của văn bản vẫn cần thiết (thực hiện thủ công) → Điều này gây tốn thời gian khi phải chấp nhận việc thử và sai để có thể tìm đầy đủ nội udng của văn bản và giảm thiểu số nhiễu.
    - RSS Feeds: là một định dạng sử dụng để phân phối nội dung tự động từ các trang web, thường là **blog, trang tin tức …** (ít thấy và sử dụng). RSS feeds thường được định dạng dưới dạng XML, giúp các ứng dụng dễ dàng đọc và xử lý thông tin.

b. Truy cập từ disk (local file)

- Đối với các văn bản cục bộ (tức là được lưu trữ trong máy), chỉ cần chú ý đến đường dẫn dẫn đến file.

2. Xử lý văn bản với Python String

- Trong Python, List và String thường được sử dụng phổ biến để giải quyết các vấn đề về NLP, tuy nhiên giữa chúng có một số khác biệt:
    - String và List đều có thể sử dụng phép toán (operation) là + và *
    - String và List đều có thể truy cập từng phần tử
    - Tuy nhiên, List có thể dễ dàng chỉnh sử thông tin của phần tử (nếu phần tử đó là một chuỗi); ngược lại String lại không thể chỉnh sửa được.
        - Điều này được lý giải vì **String** là **immutable** - nghĩa là không thể thay đổi khi đã khởi tạo

- Ngược lại, **List** lại là **mutable** - nên nội dung của nó có thể được chỉnh sửa bất cứ lúc nào → List hỗ trợ các phương thức để chỉnh sửa giá trị ban đầu nhiều hơn là các phương thức tạo ra giá trị mới
- Tuy nhiên, vẫn có thể thay đổi giá trị của String thông qua phương thức **replace()**.

- Một số method phổ biến đối với Python String:

Method	Functionality
`s.find(t)`	index of first instance of string `t` inside `s` (`-1` if not found)
`s.rfind(t)`	index of last instance of string `t` inside `s` (`-1` if not found)
`s.index(t)`	like `s.find(t)` except it raises `ValueError` if not found
`s.rindex(t)`	like `s.rfind(t)` except it raises `ValueError` if not found
`s.join(text)`	combine the words of the text into a string using `s` as the glue
`s.split(t)`	split `s` into a list wherever a `t` is found (whitespace by default)
`s.splitlines()`	split `s` into a list of strings, one per line
`s.lower()`	a lowercased version of the string `s`
`s.upper()`	an uppercased version of the string `s`
`s.title()`	a titlecased version of the string `s`
`s.strip()`	a copy of `s` without leading or trailing whitespace
`s.replace(t, u)`	replace instances of `t` with `u` inside `s`

3. Xử lý văn bản với Unicode

a. Unicode

- Unicode hỗ trợ hơn một triệu ký tự, mỗi ký tự được gán với một con số, được gọi là **code point**.
- Trong Python, **Code Point** được viết dưới dạng ***\uXXXX***, với ***XXXX*** là 4 chữ số dưới dạng hexa.
- UTF8 được sử dụng để mã hóa các ký tự bằng cách sử dụng nhiều bytes → Điều này giúp cho nó biểu diễn đầy đủ các ký tự trong Unicode.
- Quá trình minh họa cho encoding và decoding trong NLP:
    - Text trong file thường ở dạng **mã hóa (encoding)**, chúng ta cần một công cụ để chuyển hóa nó thành Unicode.
    - Việc chuyển đổi Unicode này được gọi là **decoding (giải mã).**
    - Ngược lại, để có thể viết Unicode ra file hoặc terminal, chúng ta cần thực hiện chuyển đổi nó thành mã phù hợp, việc này được gọi là **encoding (mã hóa)**

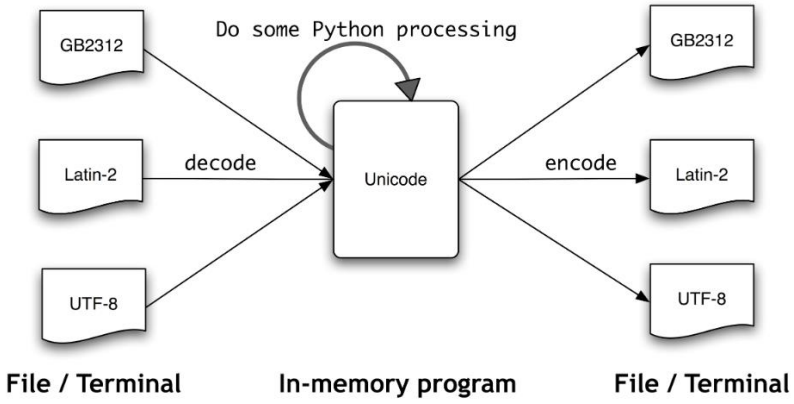


***Figure 3.3**: Unicode Decoding and Encoding*

b. Extracting Encoded text from files

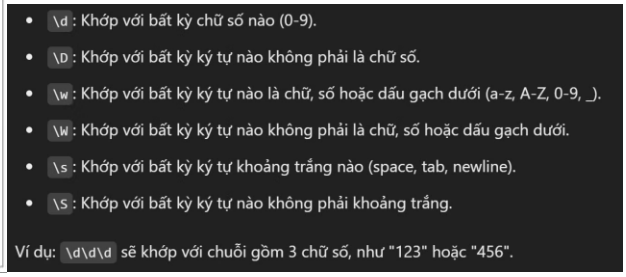
- Việc sử dụng **encoding** làm tham số cho phương thức **open()** là quan trọng để thông báo cho chương trình biết rằng văn bản đang được mã hóa ở bảng mã nào khi thực hiện đọc file.

4. Regular Expression

a. Regular Expression

- Là một tập hợp các ký tự và ký hiệu đặc biệt được sử dụng để mô tả các mẫu chuỗi văn bản. Chúng rất linh hoạt và mạnh mẽ trong việc tìm kiếm, so khớp và thao tác chuỗi.
- Một số yếu tố quan trọng trong Regular Expression:
    - **Ký tự đại diện (wildcards):** là dấu '.' (dấu chấm), tương ứng với bất kỳ ký tự nào
    - **Ký tự mỏ neo:**
        - ^ đại diện cho sự bắt đầu của chuỗi: **^abc** đại diện cho chuỗi bắt đầu bằng **abc**
        - **$** đại diện cho sự kết thúc của chuỗi: **$abc** đại diện cho chuỗi kết thúc bằng **abc**
    - **Ký hiệu []** : đại diện cho phạm vi: **[abc]** là các ký tự a hoặc b hoặc c
    - **Ký hiệu - :** đại diện cho khoảng của các ký tự: [a-d] đại diện cho các ký tự [abcd]
    - **Ký hiệu +:** đại diện cho việc xuất hiện một hoặc nhiều lần (nghĩa là ít nhất một lần). Và các pattern được truyền vào khi thực hiện tìm kiếm phải nằm **liên tiếp nhau**

Operator	Behavior
.	Wildcard, matches any character
^abc	Matches some pattern *abc* at the start of a string
abc$	Matches some pattern *abc* at the end of a string
[abc]	Matches one of a set of characters
[A-Z0-9]	Matches one of a range of characters
ed|ing|s	Matches one of the specified strings (disjunction)
*	Zero or more of previous item, e.g. a*, [a-z]* (also known as *Kleene Closure*)
+	One or more of previous item, e.g. a+, [a-z]+
?	Zero or one of the previous item (i.e. optional), e.g. a?, [a-z]?
{n}	Exactly *n* repeats where n is a non-negative integer
{n,}	At least *n* repeats
{,n}	No more than *n* repeats
{m,n}	At least *m* and no more than *n* repeats
a(b|c)+	Parentheses that indicate the scope of the operators



b. Ứng dụng

- Các ví dụ trên đều liên quan đến việc tìm kiếm các từ w khớp với một số biểu thức chính quy **regexp** bằng cách sử dụng **re.search(regexp, w).**
- Ngoài việc kiểm tra xem một biểu thức chính quy có khớp với một từ hay không, chúng ta có thể sử dụng regular expression để trích xuất tài liệu từ các words hoặc để sửa đổi chúng theo những cách cụ thể.
- Phân biệt giữa **re.search()** và **re.findall()** :
    - **re.search()** : chỉ cần tìm lần khớp đầu tiên và không quan tâm đến những kết quả khác. Trả về đối tượng match nếu tìm thấy, hoặc None nếu không có gì khớp.
    - **re.findall()** : khi muốn lấy tất cả các kết quả khớp trong một chuỗi. Trả về danh sách các kết quả khớp (non-overlapping)
- Áp dụng cho **trích xuất các từ**: Sử dụng tìm kiếm với re.findall(pattern, string) (không thực hiện tìm kiếm chồng lên nhau), thực hiện tìm kiếm tất cả các **regular repression** trong **string**
- **Thực hiện nén từ (đối với English):** là việc loại bỏ các nguyên âm ở giữa các từ, chỉ giữ lại các cụm nguyên âm ở đầu và cuối từ cùng với các phụ âm.

- **Tìm kiếm từ gốc (bằng thủ công):** Sử dụng các **regexp** để loại bỏ các hậu tố trong từ, đưa nó về từ gốc (word stem)
- **Sử dụng để phân tách từ (tokenize)**

5. Stemmer

- Là việc loại bỏ tiền tố/ hậu tố của một từ, đưa nó về từ gốc ban đầu.
- Việc thực hiện loại bỏ các phụ tố này có thể được làm bằng thủ công nhờ **regexp**, tuy nhiên các Stemmer được Python hỗ trợ lại làm rất tốt công việc này, trong đó có 2 loại chính: **Porter** và **Lancaster**.
    - Mỗi bộ Stemmer sẽ tuân thủ quy tắc riêng để loại bỏ các phụ tố của từ, việc sử dụng nó tùy thuộc vào mục đích.
    - **Porter Stemmer** là một lựa chọn tốt nếu bạn đang lập chỉ mục một số văn bản và muốn hỗ trợ tìm kiếm bằng các dạng từ thay thế

6. Lemmatizer

- Dù cùng thực hiện việc chuyển một từ về từ gốc, tuy nhiên, **Lemmatizer** sử dụng một từ điển ngữ pháp (lexicon) và ngữ cảnh ngữ pháp của từ để giảm từ về dạng gốc có nghĩa, gọi là "lemma".
- Khác với **Stemmer**, Lemmatizer trả về từ có nghĩa trong từ điển và có sự chính xác cao hơn vì nó phân tích cấu trúc ngữ pháp của từ trong ngữ cảnh.
- Các loại Lemmatizer phổ biến: **WordNet Lemmatizer**: Dựa trên WordNet (từ điển ngữ nghĩa).
- Phân biệt đơn giản giữa **Lemmatizer** và **Stemmer**

Stemmer	Lemmatizer
Cắt hậu tố để tạo ra "stem" (gốc)	Sử dụng từ điển ngữ nghĩa để tìm "lemma" (từ gốc)
Dựa trên quy tắc đơn giản, không ngữ cảnh	Phân tích ngữ pháp và ngữ cảnh của từ
Kết quả có thể không có nghĩa	Kết quả luôn là từ có nghĩa
Nhanh và đơn giản	Chậm hơn, phức tạp hơn

7. Practice

- **Yêu cầu:** Tiền xử lý văn bản được cung cấp: https://www.kaggle.com/datasets/haitranquangofficial/vietnamese-online-news-dataset?authuser=0
    - Dataset được cung cấp là các article viết bằng tiếng Việt, lưu dưới dạng file JSON.
    - Số lượng articles trong tập dataset khá lớn (khoảng 184541 bài)
- **Những việc cần làm:**
    - Lấy nội dung từ file JSON: lấy title và content của mỗi article để đại diện cho nội dung của article đó
    - Chuyển đổi các ký tự trong dữ liệu thành chữ thường
    - Chuẩn hóa văn bản
    - Làm sạch văn bản: loại bỏ các ký tự không cần thiết có trong văn bản

- Loại bỏ stop-word: vì các thư viện không hỗ trợ Stop-word cho tiếng Việt, nên cần sử dụng một tập văn bản (txt) Stop Word (có thể download từ Web) để sử dụng.
- Phân tách từ (Tokenize): Sử dụng phương thức **word_tokenize()** của thư viện **underthesea** để thực hiện (thư viện này hỗ trợ tốt cho các văn bản tiếng Việt).

- **Những lưu ý khi xử lý văn bản tiếng Việt:**
  - Việc thực hiện Stemming và Lemmatization là không cần thiết và có thể bỏ qua.
  - Thực hiện loại bỏ stop words trước khi tokenize (do tồn tại những stop-word nhiều hơn một từ), nên việc thực hiện theo thứ tự này đảm bảo các stop words được loại bỏ hoàn toàn
  - Tokenize luôn là bước thực hiện cuối cùng.