PART OF SPEECH

1. **Ý nghĩa:**

* Là bước tiếp theo của Tokenize
* Được sử dụng để gán nhãn các từ theo mức độ kiểu từ vựng (động từ, tính từ, danh từ,…)
* Tăng tính ngữ nghĩa của feature -> feature mang tính đặc trưng cao hơn
* Cần chọn một bộ nhãn (tagset) chuẩn để gán nhãn, được sử dụng phổ biến là bộ nhãn từ vựng của Penn TreeBank (48 nhãn)

https://[www.ling.upenn.edu/courses/Fall\_2003/ling001/penn\_treebank\_pos.html](https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html)

* Các phương pháp gán nhãn phổ biến là: Hidden Markov Model (HMM) and the Maximum Entropy Markov Model (MEMM).
* Làm tiền đề cho việc xác định các thực thể có nghĩa (NER)

**Ứng dụng:**

* Phục vụ cho các bài toán xử lý văn bản: tạo ra các đặc trưng mang ngữ nghĩa cao hơn

Ví dụ: **book - động** **từ** và **book – danh từ** được tính là 2 đặc trưng khác nhau

* Ứng dụng trong nhận diện giọng nói: Đầu vào là giọng nói, đầu ra là đoạn văn bản

**Khó khăn:**

* Mơ hồ, không rõ ràng: Một từ có thể thuộc nhiều loại từ và mang nghĩa khác nhau.

Ví dụ:

+ Book:

* Danh từ: quyển sách
* Động từ: đặt phòng/ lịch…

Độ chính xác phụ thuộc vào hai yếu tố:

* Tần suất xuất hiện loại từ của một từ nào đó.

Ví dụ: act có 2 loại từ là động từ và danh từ, nhưng động từ thường được sử dụng nhiều hơn danh từ.

* Ngữ cảnh của câu:

Ví dụ:

* I am waiting for the **train**

Sau mạo từ the, train luôn được dùng như danh từ

* She can **train** me on cooking

Sau modal verb, train chắc chắn là động từ

1. **POS Tagging trong StanfordCoreNLP**

- Gồm các bước cơ bản:

+ Tokenize

+ Ssplit (đoạn văn bản có nhiều câu)

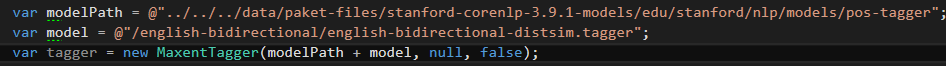
+ Gán nhãn

* Sử dụng các hàm trong lớp MaxentTagger
* Cung cấp sẵn model đã được train (có hỗ trợ hàm tự train trên dữ liệu khác của người dùng)
* Hàm khởi tạo MaxentTagger: class này sử dụng để chạy, train và kiểm tra việc dán nhãn POS. Có 2 kiểu dán nhãn Tiếng Anh là:

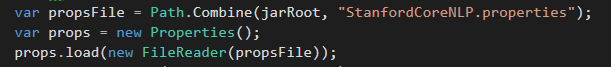
+ bi-directional dependency network tagger (gán nhãn mạng lưới phụ thuộc hai chiều): sử dụng file english-bidirectional-distsim.tagger. Độ chính xác là 97.32%.

+ Gán nhãn sử dụng chuỗi thông tin một chiều từ trái sang nhưng gán chính xác ít từ vô danh hơn cũng như ít các đặc trưng từ vựng hơn, sử dụng file english-left3words-distsim.tagger. Độ chính xác là 96.92%, bù lại kiểu dãn nhãn này nhanh hơn hẳn, và được khuyến khích sử dụng.

+ Input: đường dẫn đến tập tin model, ngoài ra có thể config một tham số kiểu Properties (các thuộc tính phục vụ cho quá trình gán nhãn)



Properties có thể được load bằng file:



Hoặc khởi tạo trực tiếp:



Và khởi tạo MaxentTagger với tham số Properties:



* Tokenize: Tạo thành một mảng các từ riêng biệt từ một đoạn văn bản

+ Input: tham số kiểu Reader (thư viện java.io)



Có thể chuyển từ kiểu string ở c# sang kiểu StringReader bằng hàm khởi tạo StringReader

+ Output: interface List(thư viện java), là mảng các token đã được tách từ chuỗi ban đầu

* TagSentence: Gán nhãn cho một mảng các từ riêng biệt

+ Input: tham số có kiểu là interface List (ArrayList)



+ Output: interface List, là mảng các từ đã được gắn tag



1. **Các tài liệu tham khảo:**

<https://ongxuanhong.wordpress.com/2016/09/02/gan-nhan-tu-loai-part-of-speech-tagging-pos/>

<https://nlp.stanford.edu/software/pos-tagger-faq.html>

<https://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/tagger/maxent/MaxentTagger.html>

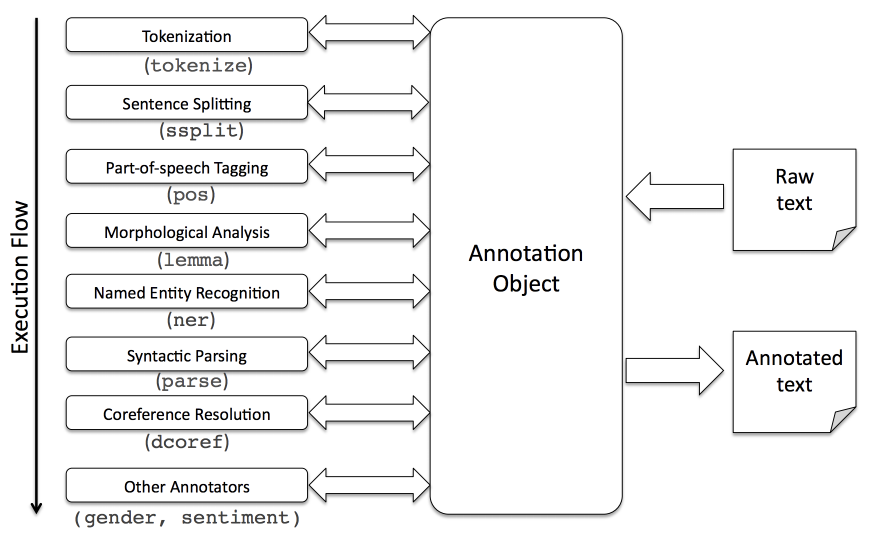
<https://sergey-tihon.github.io/Stanford.NLP.NET//samples.html#Stanford-Log-linear-Part-Of-Speech-POS-Tagger>

<https://en.wikipedia.org/wiki/Part-of-speech_tagging>

1. **Pipeline**

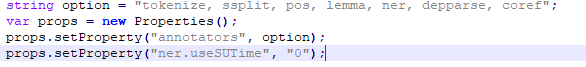
Ngoài cách sử dụng hàm như trên, thư viện có hỗ trợ pipeline để người dùng tiện sử dụng, không cần phải nhớ tên các lớp và hàm. Pipeline có thể hiểu như một luồng xử lý gồm một hoặc nhiều bước. Người dùng cần khai báo các bước xử lý, có thể lấy kết quả của một bước bất kỳ. Nếu một bước xử lý cần kết quả của các bước xử lý khác, người dùng buộc phải khai báo các bước đó, nếu không sẽ bị lỗi.

Một số bước xử lý mà pipeline của thư viện có hỗ trợ là: tokenize, ssplit, pos, lemma, ner, depparse, coref.



Để sử dụng pipeline, cần khởi tạo với tham số là kiểu Properties. Biến này có thể được tạo ra trực tiếp trong chương trình hoặc đọc từ file. Trong gói thư viện StanfordCoreNLP có hỗ trợ một file properties mẫu, ta có thể sử dụng file này hoặc chỉnh sửa tùy ý.

Ở ví dụ này, nhóm khởi tạo biến kiểu Properties ngay trong chương trình:



Các thuộc tính cần để khởi tạo pipeline bắt buộc phải có là annotators, đây có thể hiểu là các bước xử lý, khởi tạo kiểu string với format là các bước xử lý cách nhau 1 dấu “,”.

Khởi tạo pipeline:



Để xử lý một đoạn văn bản, ta tạo một biến kiểu Annotation (lưu các phân tích của từng bước xử lý của văn bản):



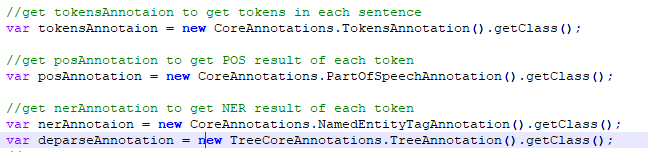
Sau đó, gán annotation cần xử lý vào pipeline:



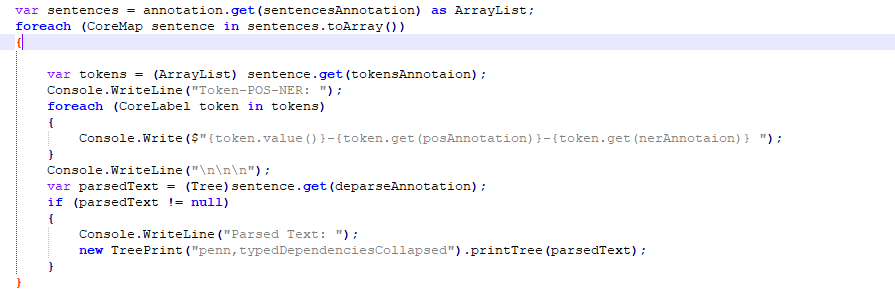
Sau đó, khi cần lấy kết quả của bước xử lý nào, cần phải lấy lớp của bước xử lý tương ứng. Ở đây, kiểu Annotation chuẩn lưu trữ các phân tích của bước “Sentence Splitting”.



Tương tự với các bước xử lý còn lại:



Từ annotation của mỗi bước, có thể lấy annotation của bước tiếp theo, ví dụ:



Từ annotation ban đầu, ta thấy ra được các câu riêng lẻ nhờ sentenceAnnotation. Từ các sentences này, ta lấy ra được các token riêng lẻ nhờ tokenAnnotation, tương tự cho các bước xử lý sau.