**Bài 11: NLTK Package phần 1 - Lập trình AI bằng Python**

Trong bài này, chúng ta sẽ học cách bắt đầu với Natural Language Toolkit Package.

**1. Điều kiện tiên quyết :**

Nếu chúng ta muốn xây dựng các ứng dụng với xử lý Ngôn ngữ Tự nhiên thì việc thay đổi ngữ cảnh sẽ gây khó khăn nhất. Yếu tố ngữ cảnh ảnh hưởng đến cách máy hiểu một câu cụ thể. Do đó, chúng ta cần phát triển các ứng dụng Ngôn ngữ tự nhiên bằng cách sử dụng các phương pháp tiếp cận máy học để máy móc cũng có thể hiểu cách con người có thể hiểu ngữ cảnh.

Để xây dựng các ứng dụng như vậy, chúng tôi sẽ sử dụng gói Python có tên NLTK (Natural Language Toolkit Package).

**2. Importing NLTK :**

Cài đặt bằng câu lệnh sau :

**pip** install nltk

Nếu bạn sử dụng conda thì sẽ sử dụng câu lệnh :

conda install -c anaconda nltk

Bây giờ sau khi cài đặt gói NLTK, chúng ta cần nhập nó qua dấu nhắc lệnh python. Chúng ta có thể nhập nó bằng cách viết lệnh sau trên dấu nhắc lệnh Python:

>>> **import** nltk

**3. Download dữ liệu NLTK :**

Bây giờ sau khi nhập NLTK, chúng ta cần tải xuống dữ liệu cần thiết. Nó có thể được thực hiện với sự trợ giúp của lệnh sau trên dấu nhắc lệnh Python:

>>> nltk.download()

**4. Cài đặt các package khác :**

Để xây dựng các ứng dụng xử lý ngôn ngữ tự nhiên bằng NLTK, chúng ta cần cài đặt các package cần thiết. Các package như sau:

**a. gensim**

Nó là một thư viện mô hình ngữ nghĩa mạnh mẽ hữu ích cho nhiều ứng dụng. Chúng ta có thể cài đặt nó bằng cách thực hiện lệnh sau:

**pip** install gensim

**b. pattern**

Nó được sử dụng để làm cho gói gensim hoạt động bình thường. Chúng ta có thể cài đặt nó bằng cách thực hiện lệnh sau

**pip** install pattern

**5. Khái niệm Tokenization, Stemming, và Lemmatization :**

**a. Tokenization :**

Nó có thể được định nghĩa là quá trình chia nhỏ văn bản đã cho, tức là chuỗi ký tự thành các đơn vị nhỏ hơn được gọi là mã thông báo. Các mã có thể là các từ, số hoặc dấu chấm câu. Nó còn được gọi là phân đoạn từ. Sau đây là một ví dụ đơn giản về mã hóa -

**Input** − Mango, banana, pineapple and apple all are fruits.

**Output :**



Quá trình phá vỡ văn bản đã cho có thể được thực hiện với sự trợ giúp của việc xác định các ranh giới từ. Sự kết thúc của một từ và sự bắt đầu của một từ mới được gọi là ranh giới từ. Hệ thống chữ viết và cấu trúc kiểu chữ của các từ ảnh hưởng đến ranh giới.

Trong mô-đun Python NLTK, ta có các gói khác nhau liên quan đến tokenization mà ta có thể sử dụng để chia văn bản thành các mã thông báo theo yêu cầu  Một số package như sau:

**sent\_tokenize package**

Như tên gợi ý, gói này sẽ chia văn bản đầu vào thành các câu

from nltk.tokenize **import** sent\_tokenize

**word\_tokenize package**

Package này chia văn bản đầu vào thành các từ.

from nltk.tokenize **import** word\_tokenize

**WordPunctTokenizer package**

package này chia văn bản đầu vào thành các từ cũng như các dấu chấm câu

from nltk.tokenize **import** WordPuncttokenizer

**b. Stemming :**

Trong khi làm việc với các từ, chúng ta bắt gặp rất nhiều biến thể do lý do ngữ pháp. Khái niệm biến thể ở đây có nghĩa là ta phải đối phó với các hình thức khác nhau của cùng một từ như democracy**,**democratic và democratization. Nó rất cần thiết để máy móc hiểu rằng những từ khác nhau này có cùng một dạng cơ sở. Bằng cách này, sẽ rất hữu ích khi trích xuất các dạng cơ bản của các từ trong khi chúng ta phân tích văn bản.

Chúng ta có thể đạt được điều này bằng cách xuất phát. Theo cách này, chúng ta có thể nói rằng tạo gốc là quá trình tự suy nghĩ để rút ra các dạng cơ sở của từ bằng cách cắt bỏ các đầu của từ.

Trong mô-đun Python NLTK, chúng tôi có các gói khác nhau liên quan đến việc tạo gốc. Các gói này có thể được sử dụng để lấy các dạng cơ bản của từ. Các gói này sử dụng các thuật toán. Một số gói như sau:

**PorterStemmer package :**

Package Python này sử dụng thuật toán Porter để trích xuất biểu mẫu cơ sở

from nltk.stem.porter **import** PorterStemmer

Ví dụ: nếu chúng ta cung cấp từ ‘writing’ làm đầu vào cho trình tạo gốc này, chúng ta sẽ nhận được từ ‘write’ sau khi tạo gốc.

**LancasterStemmer package :**

Package Python này sẽ sử dụng thuật toán Lancaster để trích xuất biểu mẫu cơ sở

from nltk.stem.lancaster **import** LancasterStemmer

**SnowballStemmer package :**

Package Python này sẽ sử dụng thuật toán của quả cầu tuyết để trích xuất dạng cơ sở

from nltk.stem.snowball **import** SnowballStemmer

Tất cả các thuật toán này có mức độ nghiêm ngặt khác nhau. Nếu chúng ta so sánh ba loại stemmers này thì Porter là ít nghiêm ngặt nhất và Lancaster là nghiêm ngặt nhất. Snowball stemmer sử dụng tốt về tốc độ cũng như độ chặt chẽ.

**c. Lemmatization**

Ta cũng có thể trích xuất dạng cơ sở của từ bởi lemmatization. Về cơ bản, nó thực hiện nhiệm vụ này với việc sử dụng phân tích từ vựng và hình thái của các từ, thông thường chỉ nhằm mục đích loại bỏ các kết thúc không theo chiều hướng. Dạng cơ sở này của bất kỳ từ nào được gọi là lemma.

Sự khác biệt chính giữa stemming và lemmatization là việc sử dụng từ vựng và phân tích hình thái của các từ. Một điểm khác biệt nữa là việc bổ đề thường thu gọn các từ liên quan đến đạo hàm trong khi bổ đề thường chỉ thu gọn các dạng vô hướng khác nhau của một bổ đề. Ví dụ: nếu chúng ta cung cấp từ saw làm từ đầu vào thì việc đặt gốc có thể trả về từ "s" nhưng việc bổ sung sẽ cố gắng trả lại từ see hoặc saw tùy thuộc vào việc sử dụng mã thông báo là động từ hay danh từ.

Trong mô-đun Python NLTK, mình có package sau liên quan đến quá trình lemmatization mà mình có thể sử dụng để lấy các dạng cơ bản của từ:

**WordNetLemmatizer package**

Package  này sẽ trích xuất dạng cơ sở của từ tùy thuộc vào việc nó được sử dụng như một danh từ hay một động từ

from nltk.stem **import** WordNetLemmatizer

**6. Chunking: Chia dữ liệu thành các Chunks**

Nó là một trong những quá trình quan trọng trong xử lý ngôn ngữ tự nhiên. Công việc chính của phân khúc là xác định các phần của lời nói và các cụm từ ngắn như cụm danh từ. Ta đã nghiên cứu quá trình mã hóa, việc tạo ra mã thông báo. Chunking về cơ bản là ghi nhãn của những token đó. Nói cách khác, phân khúc sẽ cho chúng ta thấy cấu trúc của câu.

tiếp theo, chúng ta sẽ tìm hiểu về các loại Chunking.

**Các loại chunking :**

Có 2 loại chunking bao gồm :

*a. Chunking up :*

Trong quá trình phân chia này, đối tượng, sự vật, v.v ... sẽ trở nên tổng quát hơn và ngôn ngữ trở nên trừu tượng hơn. Có nhiều cơ hội thỏa thuận hơn. Trong quá trình này, mình sẽ thu nhỏ. Ví dụ, nếu chúng ta đặt câu hỏi “ô tô dùng cho mục đích gì”?ta có thể nhận được câu trả lời là "vận chuyển".

*b. Chunking down :*

Trong quá trình phân loại này, đối tượng, sự vật, v.v ... sẽ trở nên cụ thể hơn và ngôn ngữ được thâm nhập nhiều hơn. Cấu trúc sâu hơn sẽ được xem xét khi phân tích. Trong quá trình này, ta sẽ mở rộng thông tin. Ví dụ: nếu ta rút ngắn câu hỏi "Hãy kể cụ thể về một chiếc ô tô"? Ta sẽ nhận được những thông tin nhỏ hơn về chiếc xe.

**Ví dụ :**

chúng ta sẽ thực hiện phân đoạn Danh từ-Cụm từ, một thể loại phân loại sẽ tìm các cụm danh từ trong câu, bằng cách sử dụng mô-đun NLTK trong Python -

*Bước 1 :*chúng ta cần xác định ngữ pháp để phân khúc. Nó sẽ bao gồm các quy tắc mà chúng ta cần tuân theo.

*Bước 2 :*chúng ta cần tạo một trình phân tích cú pháp chunk. Nó sẽ phân tích ngữ pháp và đưa ra kết quả.

*Bước 3 :*đầu ra được tạo ra ở định dạng cây.

Import NLTK :

**import** nltk

Bây giờ, chúng ta cần xác định câu. Ở đây, DT nghĩa là định thức, VBP nghĩa là động từ, JJ nghĩa là tính từ, IN nghĩa là giới từ và NN nghĩa là danh từ.

sentence=[("a","DT"),("clever","JJ"),("fox","NN"),("was","VBP"),

("jumping","VBP"),("over","IN"),("the","DT"),("wall","NN")]

Ta cần đưa ra ngữ pháp. Ở đây, mình sẽ đưa ra ngữ pháp ở dạng biểu thức chính quy

grammar = "NP:{<DT>?<JJ>\*<NN>}"

Ta cần xác định một trình phân tích cú pháp sẽ phân tích ngữ pháp.

parser\_chunking = nltk.RegexpParser(grammar)

Trình phân tích cú pháp phân tích câu như sau:

**parser\_chunking**.parse(sentence)

Tiếp theo,  ta cần lấy đầu ra. Đầu ra được tạo trong biến đơn giản được gọi là output\_chunk.

Output\_chunk = parser\_chunking.parse(sentence)

Khi thực thi đoạn code sau, ta có thể vẽ đầu ra của mình ở dạng cây.

**output**.draw()

