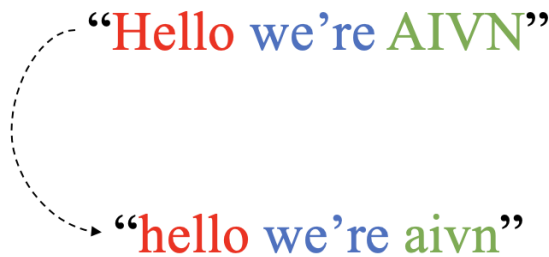


Python Text Libraries – Exercise

Ngày 20 tháng 6 năm 2022

1. Trong các bài toán Máy học liên quan đến dữ liệu văn bản (text), tiền xử lý văn bản (text preprocessing) là một kỹ thuật cực kì quan trọng, góp phần làm cải thiện khả năng biểu diễn, phân tích ngôn ngữ và trên hết là gia tăng hiệu suất mô hình. Một trong số các kỹ thuật tiền xử lý văn bản phổ biến gồm có:

- (a) **Chuyển chữ viết thường (lowercasing):** Vì máy tính coi hai chữ cái 'a' và 'A' là hai ký tự khác nhau mặc dù có cùng ý nghĩa, điều này làm gia tăng độ phức tạp cũng như tài nguyên lưu trữ khi biểu diễn văn bản. Lowercasing là kỹ thuật chuyển toàn bộ các chữ cái trong đoạn văn bản thành kiểu chữ viết thường, sử dụng lowercasing sẽ phần nào giúp ta tránh được trường hợp kể trên.



“Hello we’re AIVN”

“hello we’re aivn”

Hình 1: Minh họa kỹ thuật lowercasing

- (b) **Chuyển chữ viết hoa (uppercasing):** Với mục đích tương tự như kỹ thuật lowercasing, ở kỹ thuật uppercasing các chữ cái có trong văn bản sẽ được chuyển toàn bộ sang kiểu chữ viết hoa.



“Hello we’re AIVN”

“HELLO WE'RE AIVN”

Hình 2: Minh họa kỹ thuật uppercasing

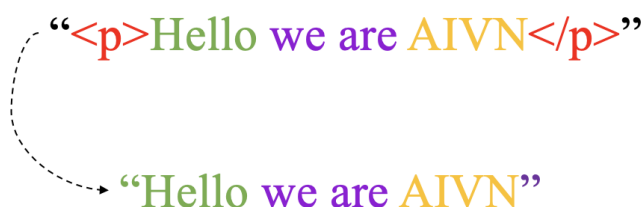
- (c) **Loại bỏ đường dẫn URL (URL removal):** là kỹ thuật loại bỏ các đường dẫn liên kết đến website có trong văn bản. Mục đích chính của việc loại bỏ các url đó là vì chúng không

thật sự biểu diễn đúng ngôn ngữ tự nhiên, vì vậy nhiều khả năng sẽ không chứa những thông tin quan trọng trong việc biểu diễn văn bản.



Hình 3: Minh họa kỹ thuật URL removal

- (d) **Loại bỏ các thẻ html (html tags removal):** là kỹ thuật loại bỏ các chuỗi có dấu <> ở đầu cuối. Đây là các chuỗi có dạng là các thẻ html. Kỹ thuật này sẽ cực kỳ hữu ích khi bộ dữ liệu của chúng ta là các file .html, hoặc văn bản trích dẫn...



Hình 4: Minh họa kỹ thuật html tags removal

Gợi ý: các bạn có thể tìm hiểu thư viện **BeautifulSoup** (sử dụng phương thức `get_text()` của class **BeautifulSoup()**) để thực hiện kỹ thuật này.

- (e) **Loại bỏ dấu câu (punctuations removal):** là kỹ thuật loại bỏ đi các dấu câu như “! ? : ; < > ...” có trong văn bản (nằm ngoài hoặc nằm bên trong một từ). Việc loại bỏ dấu câu có mục đích giảm bớt sự biểu diễn các từ không cần thiết tương tự với kỹ thuật lowercasing, uppercasing.



Hình 5: Minh họa kỹ thuật punctuations removal

- (f) **Loại bỏ stopwords (stopwords removal):** Là kỹ thuật loại bỏ các từ "stopwords" có trong văn bản. Stopwords là một tập các từ được các nhà khoa học nghiên cứu và định nghĩa là các từ không mang nhiều thông tin quan trọng trong quá trình phân tích văn bản, vì vậy có thể loại bỏ chúng khi biểu diễn văn bản.



Hình 6: Minh họa kỹ thuật stopwords removal

Gợi ý: các bạn có thể truy cập vào bộ stopwords sử dụng thư viện **nltk** ([link](#)).

- (g) **Loại bỏ các từ có tần suất xuất hiện nhiều nhất (frequent words removal):** trong một số trường hợp, việc loại bỏ các từ xuất hiện quá nhiều trong bộ dữ liệu (bộ dữ liệu sẽ bao gồm rất nhiều văn bản khác nhau) sẽ cải thiện hiệu suất mô hình. Kỹ thuật frequent words removal sẽ loại bỏ k ($k > 0$) từ có tần suất xuất hiện cao nhất. Trong phạm vi bài tập này, ta sẽ thực hiện kỹ thuật này dưới mức độ trên một đoạn văn bản duy nhất (tương đương với bộ dữ liệu chỉ có một văn bản).



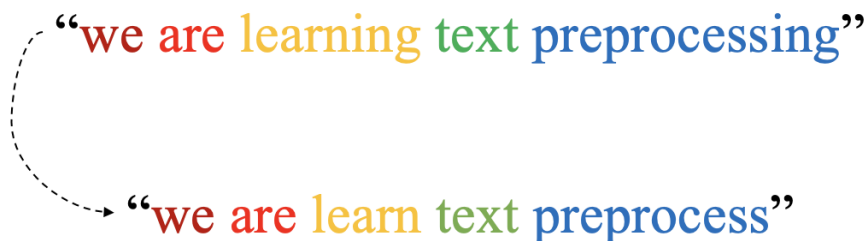
Hình 7: Minh họa kỹ thuật frequent words removal với $k = 2$

- (h) **Sửa lỗi chính tả (spelling correction):** là kỹ thuật thay thế các từ sai chính tả trong văn bản thành từ đúng. **Gợi ý:** các bạn có thể sử dụng class **Speller()** trong thư viện **autocorrect** ([link](#)) để thực hiện kỹ thuật này.



Hình 8: Minh họa kỹ thuật spelling correction

- (i) **Stemming:** trong tiếng anh, một số từ thường sẽ thay đổi đuôi của từ gốc của chúng dựa theo *từ loại* (intelligence, intelligent...) hoặc *thì* đối với động từ (change, changes, changing, changed...). Điều này dẫn đến một ý tưởng kỹ thuật đưa tất cả các từ trong văn bản về dạng gốc của chúng (stem form) nhằm biểu diễn văn bản hiệu quả và đỡ phức tạp hơn dựa trên một tập các luật đổi từ do các nhà khoa học tạo ra. **Gợi ý:** các bạn có thể sử dụng class



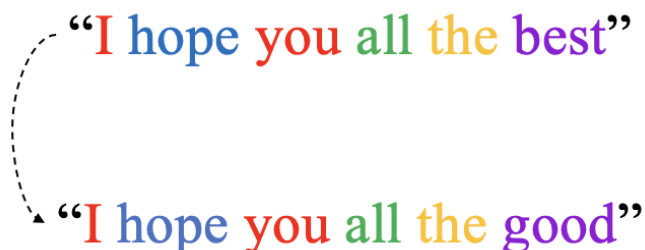
“we are learning text preprocessing”

“we are learn text preprocess”

Hình 9: Minh họa kỹ thuật stemming (kết quả từ thư viện **nltk**)

PorterStemmer() trong thư viện **nltk** ([link](#)) để thực hiện kỹ thuật này.

- (j) **Lemmatization:** tương tự với mục đích ở kỹ thuật stemming, lemmatization cũng là một kỹ thuật đưa các từ trong văn bản về dạng gốc của nó (trong tiếng anh) nhưng theo cách phức tạp hơn. **Gợi ý:** các bạn hãy sử dụng thuộc tính **lemma_** của các token mà **spacy** model tính được từ văn bản đầu vào trong thư viện **spacy** ([link](#)).



“I hope you all the best”

“I hope you all the good”

Hình 10: Minh họa kỹ thuật lemmatization (kết quả từ thư viện **spacy**)

Với khái niệm của các kỹ thuật trên, các bạn hãy viết chương trình Python khai báo class **TextPreprocessor()** với một số yêu cầu sau:

- Trong phương thức khởi tạo (constructor), cài đặt các thuộc tính về **stopwords**, **stemmer**, **spelling_correction**.
- Khai báo các phương thức mô phỏng các kỹ thuật tiền xử lý đã miêu tả ở đầu bài, với input là văn bản cần xử lý và output là văn bản đã được xử lý.
- Khai báo một phương thức có tên **preprocess_text()**, đây là phương thức sẽ áp dụng các kỹ thuật tiền xử lý đã khai báo trước đó:
 - Bộ tham số đầu vào của phương thức gồm có:
 - * 1 tham số bắt buộc **text**. Tham số này chứa văn bản cần tiền xử lý.
 - * 10 tham số mặc định bao gồm **lowercase**, **uppercase**, **remove_punct**, **remove_stopwords**, **remove_freqs**, **stemming**, **lemmatize**, **remove_url**, **remove_tags**, **word_correct**. Đây là các tham số giúp người dùng có thể chỉ định kỹ thuật họ muốn sử dụng (các bạn tùy chọn giá trị mặc định cho các tham số này).
 - Chỉ cho phép người dùng sử dụng một trong hai kỹ thuật **lowercasing** hoặc **uppercasing**. Nếu người dùng chỉ định sử dụng cả hai, in ra dòng thông báo lỗi **Cannot do both lowercasing and uppercasing. Please specify only one of them.** và trả về giá trị **None**.

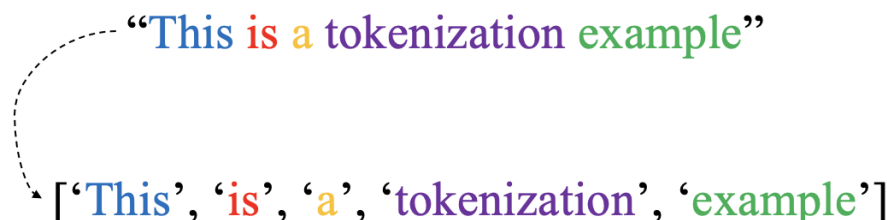
- Chỉ cho phép người dùng sử dụng một trong hai kỹ thuật stemming hoặc lemmatization. Nếu người dùng chỉ định sử dụng cả hai, in ra dòng thông báo lỗi **Cannot do both stemming and lemmatization. Please specify only one of them.** và trả về giá trị **None**.
- Với các kỹ thuật được người dùng chỉ định sử dụng, áp dụng chúng lên văn bản đầu vào và cuối cùng trả về văn bản đã được xử lý. **Lưu ý:** áp dụng các kỹ thuật theo thứ tự liệt kê ở đầu bài.
- Như vậy, Input/Output của chương trình là (sau khi đã khai báo đối tượng **TextPreprocessor()** và bắt đầu gọi phương thức **preprocess_text()**):
 - **Input:**
 - (a) text (bắt buộc).
 - (b) lowercase, uppercase, remove_punct, remove_stopwords, remove_freqs, stemming, lemmatize, remove_url, remove_tags, word_correct (không bắt buộc).
 - **Output:** Văn bản đã được tiền xử lý.
 - **Ví dụ:**

```

1 preprocessor = TextPreprocessor()
2 text = """Hello AIO-2022, we're AIVN.
3 Follow us at: https://www.facebook.com/aivietnam.edu.vn
4 #aivn #aio2022"""
5
6 print(preprocessor.preprocess_text(text))
7 # >>> hello ain follow us ain
8
9 print(preprocessor.preprocess_text(text, remove_punct=False, remove_url=
10      False, spelling_correct=False))
11 # >>> aio-2022, we'r aivn. follow us at: https://www.facebook.com/
12      aivietnam.edu.vn #aivn #aio2022
13
14 print(preprocessor.preprocess_text(text, lowercase=True, uppercase=True))
15 # >>> Cannot do both lowercasing and uppercasing. Please specify only one
16      of them.
```

2. Sau khi thực hiện tiền xử lý văn bản, người ta thường biểu diễn lại văn bản thành vector chứa các con số biểu diễn cho từng từ trong văn bản (text vectorization) nhằm giúp máy tính có thể thực hiện các phép tính toán cũng như sẽ giúp gia tăng hiệu suất mô hình nếu biểu diễn chúng đủ tốt. Kỹ thuật vector hóa văn bản sẽ bao gồm có các thành phần như sau:


- (a) **Tách token (tokenization):** tách các từ (token) trong một chuỗi thành các chuỗi riêng lẻ dựa vào khoảng trắng và lưu chúng vào một list (vector từ).



Hình 11: Minh họa kỹ thuật tokenization

- (b) **Tạo bộ từ điển (create dictionary)**: với mỗi văn bản trong bộ dữ liệu, lưu các token tách được từ các văn bản vào một list, nếu từ đó đã xuất hiện trong list rồi thì không cần phải đưa vào nữa.
- (c) **Mã hóa văn bản (text encoding)**: là kỹ thuật biểu diễn văn bản đầu vào thành vector với phần tử là các con số đại diện cho các từ bên trong văn bản.
- **Count vectorizer**: giả sử với bộ từ điển gồm n từ, lúc này ta có thể tạo một vector n phần tử có giá trị ban đầu là 0. Sau đó, nếu một từ trong văn bản đầu vào có trong bộ từ điển, vị trí của từ đó trong vector n phần tử sẽ được cộng thêm 1. Như vậy, với một văn bản ta có thể biểu diễn được thành một vector n phần tử.

“this is a a vectorizer example”

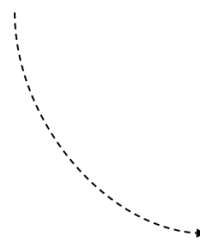


this	that	is	are	a	vectorizer	example
1	0	1	0	2	1	1

Hình 12: Minh họa kỹ thuật count vectorizer. Văn bản đầu vào được chuyển thành một vector n phần tử dựa vào bộ từ điển (trong ảnh là bộ từ điển giả định đã có sẵn)

- **One-hot encoding**: giả sử với bộ từ điển gồm n từ, ta biểu diễn mỗi từ trong văn bản đầu vào bằng một vector n phần tử dựa vào bộ từ điển, với vị trí của từ đang xét trong vector sẽ có giá trị = 1 và tất cả các vị trí phần tử còn lại sẽ có giá trị = 0. Như vậy, với một văn bản có m từ, ta có m vector n phần tử.

“this is a vectorizer example”



this	is	a	vectorizer	example
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

Hình 13: Minh họa kỹ thuật one hot encoding. Mỗi một từ trong văn bản đầu vào sẽ là một vector n phần tử.

Với những khái niệm về các bước vector hóa văn bản, các bạn hãy viết chương trình Python thực hiện khởi tạo một class **TextVectorizer()** với một số yêu cầu như sau:

- Class **TextVectorizer()** sẽ bao gồm 5 phương thức: phương thức khởi tạo, phương thức **tokenize()**, phương thức **create_dictionary()**, phương thức **one_hot_encoding()** và

phương thức `count_vectorizer()`. Trong class sẽ có một thuộc tính là **dictionary** với giá trị mặc định là **None** nếu người dùng không cung cấp bộ từ điển nào.

- Đối với phương thức **khởi tạo**, cho phép người dùng truyền vào tham số dictionary (list), là một bộ từ điển. Nếu không chỉ định bộ từ điển, gán giá trị mặc định là **None**.
- Đối với phương thức `tokenize()`, input là một văn bản (str), output là một list các token (list).
- Đối với phương thức `create_dictionary()`, input là một list các văn bản (list các str), output là một từ điển (list) với các phần tử bên trong là các token đã trích được từ các văn bản.
- Đối với phương thức `count_vectorizer()`, input là một văn bản (str), output là vector count của văn bản đầu vào (list).
- Đối với phương thức `one_hot_encoding()`, input là một văn bản (str), output là vector one-hot của văn bản đầu vào (list các list).
- Như vậy, Input/Output của chương trình là:
 - **Input:**
 - * text cần vector hóa (bắt buộc).
 - * bộ từ điển hoặc list các văn bản dùng để tạo bộ từ điển.
 - **Output:** vector one-hot của text đầu vào.
 - **Ví dụ:**

```

1 # Case 1: with corpus
2 vectorizer = TextVectorizer()
3 corpus = [
4     "hello my name is robo",
5     "this is the world of happiness",
6     "welcome to machine learning",
7     "please be aware of errors",
8     "deep learning is fun",
9     "welcome to aivn",
10    "hope you all the best aio2022"
11 ]
12
13 vectorizer.create_dictionary(corpus)
14 text = "welcome aio2022"
15 print(vectorizer.one_hot_encoding(text))
16 # >>> [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
17         0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
18         0, 0, 0, 0, 0, 0, 1]]
19
20 print(vectorizer.count_vectorizer(text))
21 # >>> [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
22         0, 0, 0, 1]
23
24 # Case 2: with dictionary
25 dictionary = [
26     'hello',
27     'my',
28     'name',
29     'robo',
30     'world',
31     'welcome',
32     'happiness'
33 ]
34
35 vectorizer = TextVectorizer(dictionary)

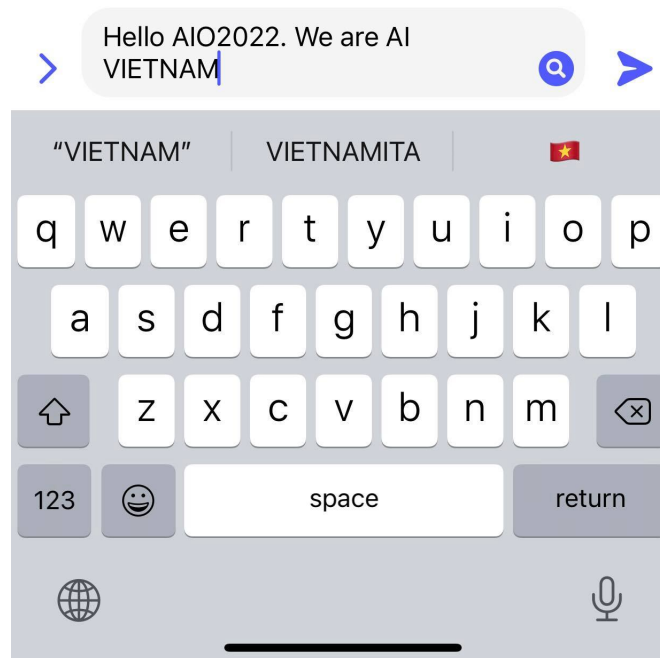
```

```

32 print(vectorizer.count_vectorizer(text))
33 # >>> [0, 0, 0, 0, 0, 1, 0]
34
35 print(vectorizer.one_hot_encoding(text))
36 # >>> [[0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 0]]
37

```

3. Khi nhắn tin trên điện thoại hoặc khi đánh văn bản trên google docs, microsoft word..., ta thường bắt gặp một cửa sổ nhỏ hiện lên với các từ. Các từ này chính là các từ hoàn thiện của từ ta đang nhập đang dở mà hệ thống dự đoán rằng đây là từ hoàn chỉnh mà người dùng dự định nhập, tính năng này được gọi là Autocomplete (hoặc word completion).



Hình 14: Tính năng Autocomplete trên iPhone

Các bạn hãy viết chương trình Python mô phỏng lại tính năng Autocomplete phiên bản đơn giản dựa trên các bước hướng dẫn sau:

- Xây dựng một bộ từ điển từ một thư mục chứa các tài liệu văn bản. Các bạn tải thư mục này tại [đây](#). **Gợi ý:** để có thể đọc các file văn bản từ một thư mục nào đó, các bạn có thể sử dụng module `os` của Python ([link](#)).
- Dựa trên bộ từ điển đã có, tìm kiếm các từ trong từ điển mà có chuỗi bắt đầu tương đồng với đoạn text đầu vào. Khi đó, các từ tìm được chính là các từ mà hệ thống Autocomplete của chúng ta dự đoán.

Như vậy, Input/Output của chương trình là:

- **Input:** một từ bất kì (`text`).
- **Output:** danh sách các từ hoàn chỉnh được dự đoán.
- **Ví dụ:**

```

1 text = "hel"
2 problem03(text)

```



```
3  """
4  >>> Suggest words:
5  1. held
6  2. hell
7  3. help
8  4. helpless
9  """
10
```

- *Hết* -