Personal Note Week 03 - Text Feature Extraction

1. Loading features from dicts

- Sử dụng **DictVectorizer** của **Scikit-learn** để chuyển đổi dữ liệu từ dạng `Python dict` thành các ma trận đặc trưng `feature matrix` → kiểu dữ liệu phù hợp với các mô hình Machine Learning
- Khi khởi tạo `DictVectorizer` , tham số `sparse` sẽ đại diện cho việc chuyển đổi và lưu trữ `feature matrix` dưới dạng **ma trận thưa** hay **đầy đủ**
  - **Ma trận thưa**: chỉ lưu trữ các phần tử khác 0, nó thể hiện (**vị trí**) và **giá trị** của phần tử đó
  - Ví dụ:

```
# ma trận đầy đủ
[[ 0,  0,  1],
 [ 0,  0,  0],
 [ 2,  0,  0]]

# ma trận thưa
(0, 2)  1
(2, 0)  2
```

- Với `DictVectorizer` được khởi tạo, nó tồn tại một số phương thức chính:
  - `fit()` : được dùng để **học** các tham số từ dữ liệu, nhưng **không** thực hiện việc chuyển đổi dữ liệu ngay. Nó sẽ học được danh sách các đặc trưng **(feature names)** từ dữ liệu đầu vào (các cặp thuộc tính-giá trị trong dictionary). Nó xác định các cột tương ứng trong ma trận đầu ra nhưng không trả về ma trận đã được chuyển đổi.
  - `transform()` : trả về ma trận đã được chuyển đổi (transform), tức là dữ liệu được biến đổi thành ma trận đặc trưng phù hợp cho mô hình học máy. (thường được dùng sau fit)
  - `fit_transform()` : thực hiện luôn cả quá trình học và chuyển đổi trong một lần gọi, trả về ma trận đặc trưng ngay lập tức.

    - `get_feature_names_out()` : cho phép xem các feature names của dữ liệu
    - `.inverse_transform()`: Để in lại dữ liệu ban đầu từ ma trận đã mã hóa.

    → Sau khi biến đổi, mỗi cột sẽ tương ứng cho một **feature**

- `DictVectorizer` cũng là một công cụ chuyển đổi biểu diễn hữu ích để huấn luyện các trình phân loại trình tự trong các mô hình Xử lý ngôn ngữ tự nhiên, thường hoạt động bằng cách trích xuất các **cửa sổ đặc trưng (features window)** xung quanh một từ quan tâm cụ thể.

2. Feature Hasing

- **Định nghĩa:**
    - Là một kỹ thuật được sử dụng để chuyển đổi các đặc trưng phân loại hoặc các đặc trưng dạng từ vựng (văn bản) thành các vector số cố định kích thước.
    - Đây là một phương pháp mã hóa đặc trưng nhanh và hiệu quả cho các mô hình học máy, đặc biệt trong các tình huống dữ liệu có rất nhiều đặc trưng hoặc các giá trị đầu vào không biết trước.
- **Cách hoạt động:**
    - Sử dụng **hàm băm (hash function)** để **ánh xạ (mapping) các đặc trưng** vào **một không gian vector có kích thước cố định**, thay vì phải **tạo ra một vector lớn** với **số chiều bằng tổng số** các **đặc trưng có thể có** (như với one-hot encoding hoặc DictVectorizer)
    - Các step chính (tổng quan):
        - **Băm giá trị đặc trưng:** được băm vào một chỉ số trong không gian vector.
        - **Định hướng giá trị**: Để tránh xung đột (collision) khi nhiều đặc trưng được băm vào cùng một chỉ số, một giá trị định hướng (thường là -1 hoặc +1) được gán cho đặc trưng băm dựa trên một hàm băm thứ hai.
        - **Ánh xạ giá trị**: Giá trị của đặc trưng được cộng hoặc trừ (tùy thuộc vào giá trị định hướng) vào vị trí chỉ số đã băm trong vector kết quả.
- **Collision (xung đột băm):**
    - Đối với Hasing, **collision (xung đột)** khi thực hiện là vấn đề quan trọng nhất:

    - **Nguyên nhân:** do số lượng vị trí trong vector đặc trưng cố định, các đặc trưng khác nhau có thể bị ánh xạ vào cùng một chỉ số trong không gian vector.
    - Để giảm thiểu lỗi do collision, **signed hash function** (hàm băm có dấu) là phương pháp được sử dụng phổ biến:
        - Thay vì chỉ ánh xạ đặc trưng vào một chỉ số cố định, hàm băm này sẽ gán thêm một **dấu (+ hoặc -)** cho giá trị của đặc trưng.
        - Dấu của giá trị được **xác định** bằng **một hàm băm thứ hai**, và **giá trị đặc trưng** sẽ **được cộng** vào hoặc **trừ** đi **tại chỉ số đã băm trong vector đặc trưng** (dựa trên dấu đó).
        - Khi hai đặc trưng khác nhau được ánh xạ vào cùng một chỉ số, chúng có thể có dấu khác nhau (một đặc trưng cộng vào, một đặc trưng trừ đi) → Điều này giúp các xung đột có thể triệt tiêu lẫn nhau thay vì cộng dồn lỗi.
        - Vì thế, **giá trị kỳ vọng** của bất kỳ đặc trưng nào trong vector đầu ra là **bằng 0**, giúp giảm khả năng sai lệch do xung đột băm.
        - Trong thư viện **scikit-learn**, cơ chế này được bật mặc định với tham số `alternate_sign=True`
    - Tuy nhiên, cơ chế dấu trong hàm băm vẫn có thể tắt khi **kích thước bảng băm lớn**, có thể **tắt** cơ chế này để hỗ trợ các mô hình yêu cầu đầu vào không âm như **MultinomialNB** hoặc các bộ chọn đặc trưng **chi-squared**.
- **FeatureHasher:**
    - **FeatureHasher** chấp nhận dữ liệu đầu vào ở nhiều dạng khác nhau (dictionary, cặp (feature, value), hoặc chuỗi ký tự).
    - Dữ liệu chuỗi được ánh xạ thành giá trị mặc định là 1.
    - Nếu một đặc trưng xuất hiện nhiều lần trong cùng một mẫu, các giá trị của nó sẽ được cộng lại.
    - Kết quả của quá trình hashing là một **ma trận thưa** trong định dạng **CSR**, giúp quản lý dữ liệu hiệu quả khi số lượng đặc trưng lớn.
- Feature Hashing có thể được sử dụng trong phân loại tài liệu, tuy nhiên nó không thực hiện các bước như tách từ (tokenize) hay các nhiệm vụ tiền xử lý khác, **chỉ thực hiện giải mã từ Unicode sang UTF-8**

- Kết quả của `hasher.transform()` là một `scipy.sparse` (ma trận thưa):
  - Số dòng là số lượng tokens được phân tách
  - Số cột là số lượng đặc trưng được phân tích từ tokens

3. Text Feature Extraction

a. Bag Of Words

- Chiến lược mô hình Bag Of Words:
  - Tách từ (tokenize) → Đếm tần suất → Chuẩn hóa và biến đổi văn bản thành ma trận số

  → Giúp các thuật toán có thể hiểu và phân tích văn bản một cách hiệu quả.
- `CountVectorizer` : là một công cụ giúp chuyển đổi văn bản thô thành ma trận đặc trưng dưới dạng "Bag of Words":
  - `CountVectorizer` đếm số lần xuất hiện của các từ (token) trong tập dữ liệu văn bản và biểu diễn chúng dưới dạng ma trận số.
  - **Mỗi hàng** trong ma trận **là một tài liệu**, và **mỗi cột** là một **từ (hoặc cụm từ)** trong **tập từ vựng**.
  - **Giá trị** tại mỗi vị trí trong ma trận thể hiện **số lần từ đó xuất hiện trong tài liệu tương ứng.**
- Các bước thực hiện:
  - **Tokenization**: Văn bản được tách thành các từ (token) hoặc n-grams (cụm từ).
  - **Build Vocabulary**: Tạo ra một danh sách từ vựng (duy nhất), bao gồm tất cả các từ xuất hiện trong tập dữ liệu.
  - **Count Occurrences**: Đếm số lần mỗi từ xuất hiện trong từng tài liệu và tạo ra một ma trận đặc trưng.
- Đối với các văn bản tiếng Việt, việc sử dụng `CountVectorizer` có thể gặp khó khăn, do đó, có thể thực hiện phép đếm thủ công với hàm `Counter` để tính tần suất xuất hiện của một **word (trong vocab)** trong một tài liệu là bao nhiêu
- Kết quả sẽ là ma trận có kích thước (n_samples x n_features): mỗi dòng đại diện cho sample tương ứng, mỗi cột đại diện cho feature (ở đây là word

trong vocabulary), giá trị tại đó là tần suất xuất hiện của từ trong sample tương ứng

- Trong mô hình Bag Of Words, số lượng token được sử dụng để xây dựng vocab có thể được định nghĩa bằng phương thức n-grams.
  - Với **unigram**, mỗi token được xem là một word.
  - Với **bigram,** hai token ghép lại với nhau (theo thứ tự).
  - Lưu ý: Việc phân tách word (xem token là một word hay character,...) tùy thuộc vào mục đích của project hướng đến. Mỗi project sẽ áp dụng việc phân tách khác nhau (như các bài sửa lỗi chính tả sẽ phân tách xem một character là một word,...)

b. TFIDF

- Là phương pháp được sử dụng để đánh giá **mức độ quan trọng** của **một từ** trong **một tài liệu** so với **toàn bộ tập tài liệu.**
- TF-IDF kết hợp hai yếu tố chính: tần suất của từ trong một tài liệu cụ thể (**TF**) và tần suất ngược của từ trong toàn bộ tập tài liệu (**IDF**).
- Công thức:
  - **TF:** Đo lường tần suất xuất hiện của **một từ** trong **một tài liệu**

$$TF(t,d) = \frac{\text{Số lần từ } t \text{ xuất hiện trong tài liệu } d}{\text{Tổng số từ trong tài liệu } d}$$

  - **IDF:** Đo lường mức độ phổ biến của từ trong tập tài liệu. Nếu từ xuất hiện trong nhiều tài liệu, giá trị IDF sẽ nhỏ, nghĩa là từ này ít mang tính phân biệt.

$$IDF(t,D) = log\left(\frac{\text{Tổng số tài liệu}}{\text{Số tài liệu chứa từ } t}\right)$$

  - **TF-IDF:** Giá trị này là tích của **TF** và **IDF**

$$TFIDF(t,d,D) = TF(t,d) \times IDF(t,D)$$

- Giải thích:

- **TF**: Mô tả tần suất của một từ trong tài liệu, nhưng không phản ánh tầm quan trọng của từ đó so với tập tài liệu.
- **IDF**: Giảm trọng số của những từ phổ biến xuất hiện ở hầu hết các tài liệu và tăng trọng số cho các từ ít xuất hiện, giúp tập trung vào các từ mang nhiều thông tin.
- **TF-IDF**: Tăng trọng số cho những từ xuất hiện nhiều trong một tài liệu cụ thể nhưng ít xuất hiện trong các tài liệu khác.