

Tokenization và text classification

☑ Trắc nghiệm 45 ☑ Thực hành 8 ⬇ PDF bài

Muốn phân loại văn bản, ví dụ nhận xét sau buổi học là khen hay chê, trước hết phải biến câu thành số. Tokenization cắt câu thành các đơn vị nhỏ (từ, mảnh của từ hoặc ký tự); sau đó mỗi câu được biểu diễn bằng số lần xuất hiện của từng token. Bài này học các cách cắt token, dựng một baseline phân loại, rồi xem các câu baseline đoán sai để biết nó thiếu gì.

MỤC TIÊU

- Phân biệt các cách tách token: ký tự, từ, subword.
- Làm một bước BPE bằng tay.
- Dựng TF-IDF với logistic regression; so với Naive Bayes.
- Đọc lỗi của model: phủ định, từ chưa gặp, dữ liệu trùng.

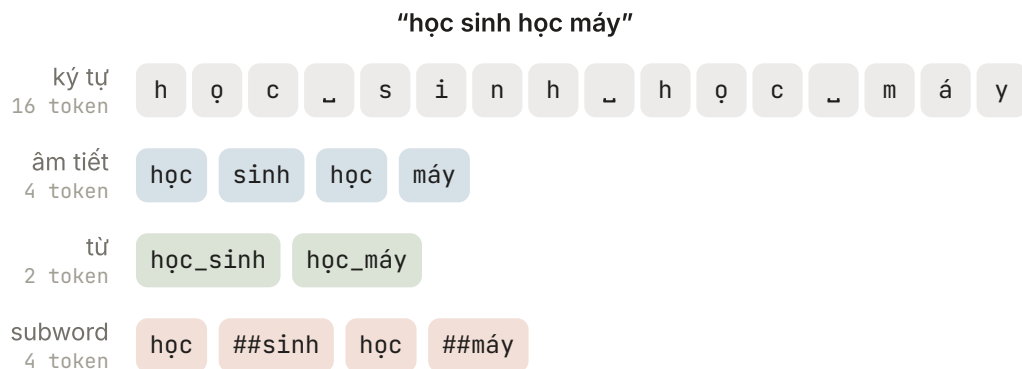
Một khoảng trắng chưa chắc là một ranh giới từ

Trong C++, đọc bằng `cin >> s` lấy các mảnh ngăn bởi khoảng trắng. Ta có thể làm tương tự bằng `split()`, nhưng với câu “học sinh học máy”, kết quả học, sinh, học, máy là bốn âm tiết. Khi nói về người học và lĩnh vực AI, “học sinh” và “học máy” thường là hai đơn vị từ vựng. Tách từ tiếng Việt cần xét ngữ cảnh; không thể ghép mọi cặp âm tiết bằng một quy tắc cố định.

Token là đơn vị mà một hệ thống xử lý, không nhất thiết là từ trong ngôn ngữ học.

Tokenizer có thể dùng từ, ký tự, byte hoặc subword (mảnh dưới từ). Cùng một câu sẽ có số token khác nhau dưới các tokenizer khác nhau.

Ký hiệu $t_1, \dots, t_L$ chỉ chuỗi token có độ dài L ; mỗi token được tra thành một ID nguyên trong từ vựng V . ID là địa chỉ trong bảng, không phải mức độ hay cỡ: token ID 100 không “lớn nghĩa” hơn ID 20.



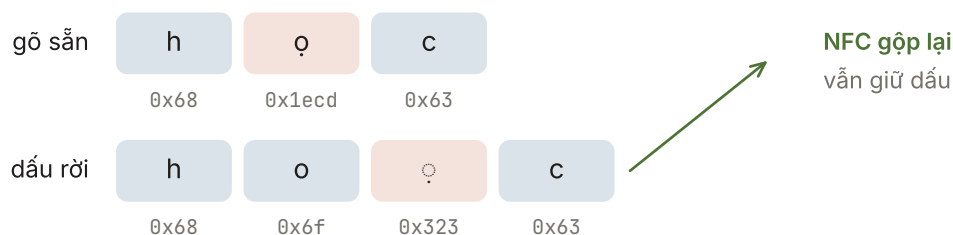
cùng một câu, số token tùy đơn vị tokenizer chọn

Cùng một câu tách theo ký tự, âm tiết, từ hay subword cho số token khác nhau.

Ta chuẩn hóa Unicode về NFC trước khi đếm. Hai chuỗi trông giống hệt nhau có thể khác nhau ở mức mã: chữ “ọ” có thể là một ký tự dựng sẵn, hoặc chữ “o” theo sau là dấu nặng kết hợp U+0323. NFC gộp cách viết thứ hai về cách thứ nhất và vẫn giữ dấu; tự xóa dấu sẽ làm “ma”, “má”, “mà” mất khác biệt.

Viết thường cũng là một lựa chọn theo nhiệm vụ: phù hợp với baseline nhận xét ở đây, nhưng có thể làm mất thông tin tên riêng trong NER ở bài 6.3.

Bước cắt văn bản thành token gọi là tách token (tokenization); dấu câu thường thành token riêng. Đổi “ko” thành “không”, “dc” thành “được”, hay đưa “Hà Nội” và “hà nội” về cùng một dạng, là chuẩn hóa văn bản. Tiếng Trung, tiếng Nhật không có khoảng trắng giữa các từ nên tách còn khó hơn; với tiếng Việt, chuỗi như “học sinh học sinh học” có nhiều cách ghép âm tiết thành từ, phải dựa vào ngữ cảnh.



Hai chuỗi cùng hiện là “học” nhưng khác số ký tự; NFC đưa về một dạng.

Python

```
import unicodedata
import numpy as np

def normalize_text(text):
    text = unicodedata.normalize("NFC", text)
    return " ".join(text.lower().split())

sentence = "Học sinh học máy"
print(normalize_text(sentence).split())
nfc = unicodedata.normalize("NFC", "ho\u0323c")
print(nfc == "học")
```

KẾT QUẢ

```
['học', 'sinh', 'học', 'máy']
True
```

Dòng đầu là bốn âm tiết cắt theo khoảng trắng, chưa phải kết quả tách từ. Dòng thứ hai cho thấy chữ “o” cộng dấu nặng rồi, sau NFC, trùng với “học” gõ sẵn. Với model pretrained, phải dùng đúng tokenizer và quy trình chuẩn hóa đi kèm model. Chẳng hạn, PhoBERT nhận văn bản đã tách từ, các âm tiết trong một từ nối bằng dấu gạch dưới (học_sinh); các model đa ngôn ngữ như mBERT hay XLM-R nhận văn bản thô và tự cắt subword.

HỎI NHANH

"học sinh".split() trả về hai mảnh. Từ đó ta rút ra được điều gì?

- A Trong tiếng Việt, giữa hai khoảng trắng luôn là đúng một từ
- B Đây là hai âm tiết, còn theo ngữ cảnh có thể chúng là một từ
- C Phải bỏ dấu trước, rồi mới ghép được thành một từ
- D Model nào cũng cần nối các âm tiết bằng dấu _

HỎI NHANH

Chuỗi `b` in ra trông như “học”, nhưng `b == "học"` lại là `False`, và `[hex(ord(ch)) for ch in b]` cho `['0x68', '0x6f', '0x323', '0x63']`. Trước khi đếm token, nên xử lý thế nào?

- A Dấu nặng đang tách rời khỏi chữ “o”, nên chuẩn hóa NFC
- B Ký tự `0x323` là rác, nên xóa mọi ký tự không phải chữ cái
- C Chuỗi bị lỗi mã hóa, nên bỏ dấu cả hai chuỗi rồi so
- D Hai chuỗi khác chữ hoa thường, nên gọi `lower()` rồi so lại

Subword: từ lạ được ghép từ những mảnh quen

Một từ vựng chứa mọi tên riêng, từ mới và lỗi gõ sẽ rất lớn. Chỉ dùng ký tự thì từ vựng nhỏ, nhưng chuỗi dài. Subword dung hòa hai điều: giữ các mảnh thường gặp thành token, chia những từ ít gặp thành nhiều mảnh.

BPE (byte-pair encoding) bắt đầu từ các ký tự, đếm cặp token kề nhau trên train, gộp cặp có tần suất cao nhất thành một token mới, rồi lặp lại cho đến khi từ vựng đủ kích thước. Tần suất tính có trọng số theo số lần từ xuất hiện.

Để nhìn thuật toán mà không vướng dấu, dùng ba từ tiếng Anh do ta tự tạo corpus: `low` xuất hiện 3 lần, `lower` 2 lần, `new` 1 lần. Cặp `l, o` và `o, w` đều có 5 lần. Ta quy định phá hòa theo thứ tự từ điển, vì vậy bước đầu gộp `l, o`.

Tokenizer của GPT-2 chạy cùng thuật toán trên byte UTF-8 thay vì ký tự (byte-level BPE): 256 giá trị byte là từ vựng gốc, nên chuỗi nào cũng mã hóa được mà không cần token unknown.

Python

```
from collections import Counter
```

```

word_counts = {"low": 3, "lower": 2, "new": 1}
pieces = {word: list(word) for word in word_counts}
pair_counts = Counter()
for word, tokens in pieces.items():
    for left, right in zip(tokens, tokens[1:]):
        pair_counts[(left, right)] += word_counts[word]

best_pair = sorted(pair_counts, key=lambda pair: (-pair_counts[pair], pair))[0]

def merge_pair(tokens, pair):
    merged = []
    i = 0
    while i < len(tokens):
        if i + 1 < len(tokens) and (tokens[i], tokens[i + 1]) == pair:
            merged.append(tokens[i] + tokens[i + 1])
            i += 2
        else:
            merged.append(tokens[i])
            i += 1
    return merged

print(best_pair, pair_counts[best_pair])
for word in word_counts:
    print(word, merge_pair(pieces[word], best_pair))

```

KẾT QUẢ

```

('l', 'o') 5
low ['lo', 'w']
lower ['lo', 'w', 'e', 'r']
new ['n', 'e', 'w']

```

Đối chiếu con số 5 bằng tay: cả low và lower chứa l, o, nên $3 + 2 = 5$. Nếu đếm mỗi loại từ một lần, cặp này chỉ còn 2 và thứ tự gộp có thể đổi: từ phổ biến mất ưu thế. Mỗi bước gộp thêm một token vào từ vựng và một luật vào danh sách. Khi mã hóa câu mới, tokenizer áp các luật đã học theo đúng thứ tự; validation và test chỉ được mã hóa, không được học thêm luật.

HỎI NHANH

Sau bước gộp l, o, corpus gồm lo w 3 lần, lo w e r 2 lần, và n e w 1 lần. Bước BPE thứ hai gộp cặp nào?

A lo, w, xuất hiện 5 lần

B o, w, xuất hiện 5 lần

C w, e, xuất hiện 2 lần

D e, r, xuất hiện 2 lần

	ký tự	gộp l, o (5 lần)	gộp lo, w (5 lần)
low × 3	l o w	lo w	low
lower × 2	l o w e r	lo w e r	low e r
new × 1	n e w	n e w	n e w

tần suất có trọng số: l, o có trong low (3) và lower (2), cộng 5

BPE gộp dần cặp hay gặp nhất: l, o rồi lo, w, nên “low” thành một token.

h o c s i n h l u y ê n t o k e n i z a t i o n v à o v e r f i t t i n g t r ư ớ c k ỹ t h i i o a i .
học sinh luyện tokenization và overfitting trước kỳ thi ioai .

từ vựng 149 mảnh (chỉ có ký tự): câu thành 51 token

học sinh luyện tokenization và overfitting trước kỳ thi ioai .

từ vựng 300 mảnh: câu thành 33 token

học sinh luyện tokenization và overfitting trước kỳ thi ioai .

từ vựng 1200 mảnh: câu thành 17 token

từ vựng 4000 mảnh: câu thành 11 token

BPE học trên lời giảng của giáo trình: từ vựng càng lớn, câu càng ít token.

WordPiece, tokenizer của BERT, cũng học một từ vựng subword nhưng khác BPE ở hai chỗ:

- **Khi học**, nó không chọn cặp xuất hiện nhiều nhất mà chọn cặp có điểm $\frac{f(a,b)}{f(a)f(b)}$ cao nhất (f là tần suất), tức ưu tiên hai mảnh hiếm khi đứng riêng.
- **Khi mã hóa**, nó chỉ dùng từ vựng cuối, không dùng luật gộp: lấy mảnh dài nhất có trong từ vựng khớp đầu từ, rồi lặp lại với phần còn lại. Tiền tố **##** đánh dấu mảnh nối tiếp. Nếu đến một chỗ không mảnh nào khớp, cả từ thành **[UNK]**.

Cùng một từ có thể bị hai tokenizer cắt khác nhau, nên với model pretrained, in token thật do tokenizer của model trả về thay vì đoán từ tên model.

Gặp từ hiếm, tokenizer subword không bỏ đi mà tách thành nhiều mảnh quen; dấu **##** của WordPiece đánh dấu mảnh nối tiếp phần trước của cùng một từ, như `play + ##ing`. BPE mức byte không bao giờ cần **[UNK]**, vì chuỗi nào cũng là một dãy byte;

Tokenization giải thích vài lỗi lạ của mô hình ngôn ngữ. Model thấy “strawberry” là vài mảnh chứ không thấy từng chữ cái, nên đếm chữ r dễ sai; số dài bị cắt thành những mảnh không theo hàng đơn vị, nên cộng số dài khó. Tokenizer học chủ yếu trên tiếng Anh tách tiếng Việt vụn hơn: cùng một nội dung có thể tốn gấp đôi số token, tức gấp đôi chi phí và chỗ trong ngữ cảnh.

HỎI NHANH

Từ vựng WordPiece chỉ có `un`, `una`, `##able` và `[UNK]`. Mã hóa từ `unab!e` thì được gì?

A `un, ##able`

B `una, ##ble`

C `una, ##b, ##le`

D `[UNK]`

từ vựng

un

una

##able

[UNK]

unable

una

+ ble ?

##ble, ##bl, ##b đều không có

→

[UNK]

tham lam lấy mảnh dài nhất, không quay lui thử un + ##able

WordPiece lấy mảnh dài nhất trước; phần còn lại không khớp nên cả từ thành UNK.

Từ chuỗi token đến TF-IDF

Baseline cổ điển không cần subword: ta đếm âm tiết hoặc cụm âm tiết xuất hiện trong nhận xét. Bag-of-words đặt mỗi mục từ thành một cột; mỗi câu thành một hàng. Nó giữ tần suất nhưng bỏ vị trí. Vì thế “An giúp Bình” và “Bình giúp An” có cùng vector unigram. Bigram (hai token liền nhau) khôi phục một phần thứ tự, nhưng vẫn không biểu diễn toàn bộ cấu trúc câu.

TF-IDF giảm trọng số của những mục xuất hiện ở quá nhiều câu. Với cấu hình mặc định làm tròn IDF của scikit-learn, nếu có n câu train và mục j xuất hiện trong df_j câu, ta dùng

$$\text{idf}_j = \log \frac{1 + n}{1 + df_j} + 1.$$

Trước chuẩn hóa, đặc trưng là số lần xuất hiện nhân IDF. Sau đó, mặc định mỗi hàng được chuẩn hóa có độ dài L2 bằng 1 nếu khác vector 0. Từ lặp nhiều được nhấn mạnh trong câu, từ quá phổ biến được giảm tương đối. Không có “hiểu nghĩa” nào tự xuất hiện từ công thức này: hai từ đồng nghĩa khác chữ vẫn ở hai cột.

Vài thuật ngữ đi kèm:

- Kho văn bản dùng làm dữ liệu gọi là **corpus**; mỗi tài liệu thành một vector, nên 12 tài liệu cho 12 vector.

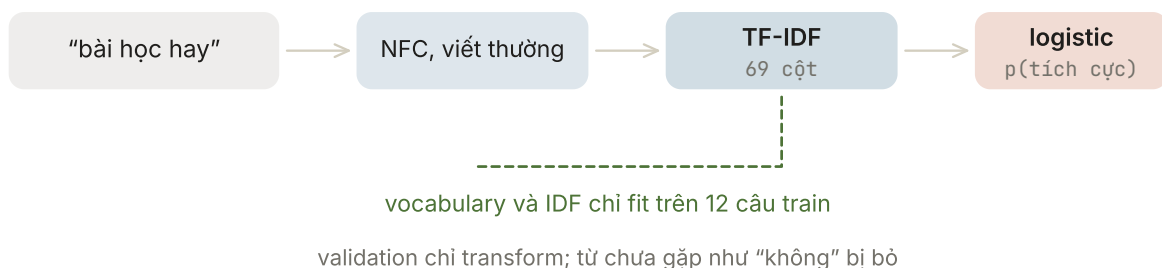
- Những từ rất phổ biến như “là”, “và”, “của” gọi là **từ dừng** (stopword); bỏ chúng thu nhỏ từ vựng và bớt nhiễu, nhưng phải cẩn thận với từ mang nghĩa như “không”.

Ở dạng gốc không làm trơn, tf là số lần xuất hiện chia độ dài văn bản và $idf = \log(N/df)$: “mèo” xuất hiện 3 lần trong bài 150 từ có tf 0.02; “và” có mặt trong cả 1000 văn bản có $idf = \log 1 = 0$ nên TF-IDF bằng 0. Lấy những mục có TF-IDF cao nhất của một bài cũng là cách đơn giản để trích từ khóa.

HỎI NHANH

Sau TF-IDF có chuẩn hóa L2 mặc định, nhận xét “hay” và nhận xét “hay hay hay” cho ra hai vector thế nào?

- A Vector thứ hai gấp ba lần vector thứ nhất
- B Hai vector bằng nhau
- C Vector thứ hai gấp $\sqrt{3}$ lần vector thứ nhất
- D Hai vector khác nhau, vì IDF tính theo số lần lặp trong câu



Chuẩn hóa văn bản, TF-IDF fit trên train, rồi logistic regression.

Với phân loại nhị phân, logistic regression cho điểm $z = \mathbf{w}^\top \mathbf{x} + b$, rồi xác suất $p = 1/(1 + e^{-z})$. Ở đây nhãn 1 là nhận xét tích cực và nhãn 0 là nhận xét tiêu cực. Thư viện học $\mathbf{w}$, b bằng loss phân loại và regularization. Ta đã học logistic regression ở chương 2; điểm mới là $\mathbf{x}$ được rút từ văn bản.

Một baseline cổ điển khác là Naive Bayes đa thức (multinomial). Nó giả định các token độc lập với nhau khi đã biết nhãn, nên xác suất của cả câu là tích xác suất từng token:

$$\log P(y \mid \mathbf{x}) = \log P(y) + \sum_j x_j \log P(j \mid y) + \text{const}, \quad P(j \mid y) = \frac{c_{jy} + \alpha}{\sum_k c_{ky} + \alpha|V|}.$$

Ở đây x_j là số lần mục j có trong câu, c_{jy} là tổng số lần mục j xuất hiện trong các câu train nhãn y , còn α (thường bằng 1, gọi là làm trơn Laplace) giữ cho mục chưa gặp ở một lớp không có xác suất 0: từ “trúng thưởng” chưa từng có trong 100 từ của các thư thường, từ vựng 50 mục, có xác suất $(0 + 1)/(100 + 50) = 1/150$ trong lớp ấy. Học chỉ là đếm, nên Naive Bayes chạy rất nhanh và thường mạnh khi ít dữ liệu.

Giả định độc lập sai với văn bản: “theo” gần như luôn đi cùng “dõi”, nên cùng một bằng chứng bị cộng hai lần và xác suất bị đẩy về gần 0 hoặc 1. Hiệu log-xác suất của hai lớp là tổng có trọng số của các x_j , nên Naive Bayes cũng là một bộ phân loại tuyến tính trên vector đếm.

Tính thử TF-IDF và logistic regression trên một train có ba câu: “hay dễ”, “hay rõ”, “khó”. Mục “hay” xuất hiện trong hai câu nên IDF bằng $\log(4/3) + 1 \approx 1.288$; “dễ” xuất hiện trong một câu nên bằng $\log(4/2) + 1 \approx 1.693$. Vector chưa chuẩn hóa của “hay dễ” trên hai cột này là $(1.288, 1.693)$. Độ dài của nó xấp xỉ 2.127, nên sau chuẩn hóa là $(0.605, 0.796)$. Tích vô hướng với trọng số giả định $(1, -1)$ và bias 0 cho $z \approx -0.191$, tức $p \approx 0.452$. Trọng số $(1, -1)$ được chọn để luyện tính, không lấy từ baseline.

Python

```
raw = np.array([np.log(4 / 3) + 1, np.log(4 / 2) + 1])
unit = raw / np.linalg.norm(raw)
z_hand = unit @ np.array([1.0, -1.0])
print("idf:", raw.round(3))
print("tfidf:", unit.round(3))
print("p:", round(1 / (1 + np.exp(-z_hand)), 3))
```

KẾT QUẢ

```
idf: [1.288 1.693]
tfidf: [0.605 0.796]
p: 0.452
```

Code cho đúng ba số đã tính tay. Nếu “hay” có mặt ở cả ba câu, IDF của nó là $\log(4/4) + 1 = 1$: mục phổ biến chỉ bị giảm tương đối, không bị xóa khỏi biểu diễn.

HỎI NHANH

Giữ IDF như trên, trọng số giả định là $(1, -1)$ và bias bằng 0. Câu “hay hay dễ” cho p xấp xỉ bao nhiêu?

- A 0.452, giống câu “hay dễ”
- B 0.707
- C 0.571
- D 0.500, vì chuẩn hóa xóa hiệu ứng lặp

Dựng baseline trên dữ liệu nhỏ

Đoán một nhận xét là khen hay chê gọi là phân tích cảm xúc (sentiment analysis); xếp câu “cho mình đặt bàn 4 người lúc 7 giờ” vào nhóm “đặt chỗ” là phân loại ý định. Cả hai đều là phân loại văn bản. Ta tự viết 12 câu train và 6 câu validation, không tải dữ liệu.

Sáu câu đủ để soi từng lỗi nhưng quá ít để so hai model: mỗi câu đổi accuracy $1/6 \approx 0.167$. Validation cố ý chứa hai câu phủ định để tìm giới hạn của biểu diễn. Các câu cùng nhãn không được mặc định là độc lập nếu chúng là bản sao hoặc cùng một người viết.

Python

```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix

train_texts = [
    "bài học hay dễ hiểu", "nội dung rõ ràng bổ ích",
    "bài giảng tốt dễ theo dõi", "ví dụ hay hữu ích",
    "khóa học thú vị", "lời giải rõ ràng",
```

```

    "bài học khó hiểu nhầm chán", "nội dung kém rồi",
    "bài giảng tệ khó theo dõi", "ví dụ sai vô ích",
    "khóa học nhầm chán", "lời giải rồi sai",
]
train_labels = np.array([1] * 6 + [0] * 6)
valid_texts = [
    "bài học hay", "lời giải sai", "bài giảng không hay",
    "ví dụ rõ ràng", "nội dung rồi", "khóa học không tệ",
]
valid_labels = np.array([1, 0, 0, 1, 0, 1])

baseline = Pipeline([
    ("tfidf", TfidfVectorizer(preprocessor=normalize_text, ngram_range=(1, 2))),
    ("classifier", LogisticRegression(C=4.0, random_state=0, max_iter=1000)),
])
baseline.fit(train_texts, train_labels)
valid_pred = baseline.predict(valid_texts)
print("shape train:", baseline["tfidf"].transform(train_texts).shape)
print("accuracy:", f"{accuracy_score(valid_labels, valid_pred):.3f}")
print("macro-F1:", f"{f1_score(valid_labels, valid_pred, average='macro'):.3f}")
print(confusion_matrix(valid_labels, valid_pred, labels=[0, 1]))

```

KẾT QUẢ

```

shape train: (12, 69)
accuracy: 0.667
macro-F1: 0.667
[[2 1]
 [1 2]]

```

Shape (12, 69) nghĩa là 12 câu train và 69 mục: 31 âm tiết cùng 38 bigram đã gặp trong train. Confusion matrix có hàng là nhãn thật, cột là nhãn đoán, theo thứ tự [0, 1]. Hai ô đường chéo là số câu đúng, hai ô còn lại chỉ hướng nhầm.

Accuracy đếm tổng số đúng chia 6; macro-F1 lấy trung bình F1 của hai lớp để mỗi lớp có vai trò ngang nhau. Trên dữ liệu lệch lớp, accuracy cao vẫn có thể che một lớp bị bỏ qua.

	đoán 0	đoán 1	
thật 0	2	1	"bài giảng không hay"
thật 1	1	2	"khóa học không tệ"

accuracy 4/6 = 0.667, macro-F1 0.667

Hai câu đoán sai đều chứa “không”.

Trong Pipeline, TF-IDF được fit trên train, rồi dùng cùng vocabulary và IDF khi gọi `predict(valid_texts)`. Ta không gọi `fit_transform` riêng trên validation: thứ tự cột có thể đổi và thống kê của validation sẽ lọt vào thí nghiệm. Có cuộc thi cho phép fit vectorizer trên cả văn bản test không nhãn; khi luật cho phép, cách làm ấy phải áp dụng như nhau cho mọi thí nghiệm để các điểm còn so được với nhau.

HỎI NHANH

Ta muốn chạy 5-fold cross-validation cho baseline trên cả 18 câu có nhãn. Làm cách nào thì điểm mới tin được?

- A Fit `TfidfVectorizer` trên cả 18 câu, rồi CV logistic trên ma trận
- B Fit `TfidfVectorizer` riêng trên phần validation của từng fold
- C Giữ vectorizer đã fit trên 12 câu train, chỉ CV phần logistic
- D Đưa cả Pipeline cùng văn bản thô vào `cross_val_score`

Đọc câu sai trước khi tăng độ phức tạp

Ta cần xem từng nhận xét, thay vì chỉ nhìn một điểm tổng. Cột xác suất dưới đây là xác suất model gán cho nhãn 1, không phải xác suất rằng model “hiểu câu”.

Python

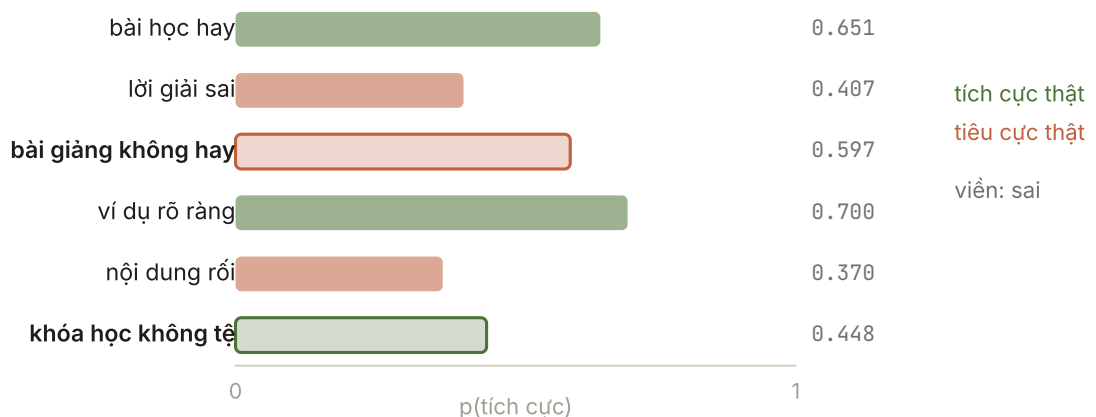
```
positive_index = list(baseline["classifier"].classes_).index(1)
positive_prob = baseline.predict_proba(valid_texts)[:, positive_index]
for text, true, pred, prob in zip(valid_texts, valid_labels, valid_pred, positive_prob):
    print(f"thật={true} đoán={pred} p1={prob:.3f} | {text}")
print("'không' có trong vocabulary:", "không" in baseline["tfidf"].vocabulary_)
```

KẾT QUẢ

```
thật=1 đoán=1 p1=0.651 | bài học hay
thật=0 đoán=0 p1=0.407 | lời giải sai
thật=0 đoán=1 p1=0.597 | bài giảng không hay
thật=1 đoán=1 p1=0.700 | ví dụ rõ ràng
thật=0 đoán=0 p1=0.370 | nội dung rối
thật=1 đoán=0 p1=0.448 | khóa học không tệ
'không' có trong vocabulary: False
```

Hai câu chứa “không” cần được xem xét. Từ này không có trên train nên bị bỏ trong biểu diễn; không hay không thể được nhận diện bằng bigram đã học. Model còn nhìn thấy “hay” hoặc “tệ”, rồi suy ra hướng ngược với nghĩa cả câu. Baseline đã có bigram, nhưng một bigram chỉ thành cột khi đã xuất hiện trong train. Nói chung, từ chưa gặp ở train không có cột nên TF-IDF bỏ qua nó khi transform.

Danh sách stopwords chứa “không” còn tệ hơn: “phim không hay” thành “phim hay”, đổi hẳn nghĩa. Những câu mỉa mai như “giao hàng nhanh ghê, đúng 3 tuần là tới” cần model đọc được ngữ cảnh như ở bài 6.3.



Hai câu sai chứa “không”, từ chưa có trong vocabulary.

Liệu một bộ phân loại khác có cứu được? Giữ nguyên dữ liệu, thay logistic regression bằng Naive Bayes trên vector đếm:

Python

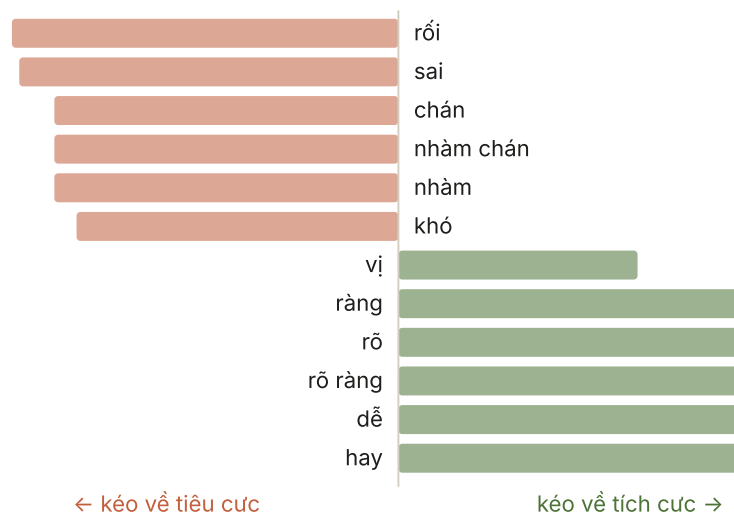
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB

naive_bayes = Pipeline([
    ("count", CountVectorizer(preprocessor=normalize_text, ngram_range=(1, 2))),
    ("classifier", MultinomialNB(alpha=1.0)),
])
naive_bayes.fit(train_texts, train_labels)
print("NB đoán:", naive_bayes.predict(valid_texts))
print("thật:   ", valid_labels)
```

KẾT QUẢ

```
NB đoán: [1 0 1 1 0 0]
thật:     [1 0 0 1 0 1]
```

Naive Bayes sai đúng hai câu có “không”: hai bộ phân loại khác nhau, một đọc TF-IDF, một đọc vector đếm, nhưng cùng một vocabulary đã mất từ phủ định, nên nhầm như nhau. Cách cải thiện bắt đầu từ biểu diễn và dữ liệu: thêm câu phủ định đủ đa dạng vào train, đánh giá trên người viết hoặc chủ đề chưa thấy, rồi mới cân nhắc model ngữ cảnh ở bài 6.3.



Các từ có trọng số lớn nhất đều là từ khen, chê trực tiếp; không có cột nào cho “không”.

HỎI NHANH

Cả logistic regression lẫn Naive Bayes đều đoán ngược hai câu có chữ “không”. Bước tiếp theo nào nhắm đúng nguyên nhân?

- A Thử thêm một classifier mạnh hơn, như gradient boosting
- B Hạ threshold để các câu phù định rơi về lớp 0
- C In ra các feature còn lại của câu sau transform
- D Tăng C để model chú ý hơn tới các từ hiếm

Khi đi thi

- Dựng TF-IDF cùng logistic regression trước: vài giây trên CPU là có mốc và một file nộp hợp lệ.
- Chia validation giống cách tạo test, theo người viết hay theo thời gian nếu cần; loại bản trùng trước khi chia.
- Với model pretrained, đếm độ dài bằng đúng tokenizer của nó trước khi chọn `max_length`.

Tóm tắt

- Token là đơn vị của tokenizer; tiếng Việt có khoảng trắng giữa âm tiết nên phải kiểm ranh giới từ.
- BPE học luật gộp theo tần suất có trọng số và áp lại đúng thứ tự; WordPiece mã hóa tham lam bằng mảnh dài nhất, không khớp thì [UNK].
- TF-IDF là biểu diễn từ vừng; logistic regression và Naive Bayes là bộ phân loại tuyến tính đọc nó; cả pipeline fit trên train.

- Điểm tổng đi cùng câu sai và confusion matrix; hai model cùng sai một câu thì xem biểu diễn và dữ liệu trước khi đổi model. Phủ định, từ lạ và bản trùng là những phép thử cần có.

Kiểm tra cuối bài

CÂU 1

Bình luận sản phẩm có nhiều tên hãng mới và lỗi gõ như “đẹppp”. Tokenizer theo từ đã có 300 nghìn mục mà test vẫn đầy từ lạ. Chuyển sang subword như BPE hay WordPiece thì được lợi chính ở đâu?

- A Chạy nhanh hơn, vì chuỗi subword luôn ngắn hơn chuỗi từ
- B Cài đặt đơn giản hơn, vì không phải học từ vựng trên train
- C Không có lợi với tiếng Việt, vì subword chỉ học được trên tiếng Anh
- D Từ lạ được ghép từ mảnh quen, từ vựng nhỏ, từ cùng gốc dùng chung mảnh

PHÒNG THEO NOAI 2026 Singapore · Vòng sơ loại · Câu 235

CÂU 2

Đề giới hạn mỗi bình luận 128 token, đếm theo tokenizer của model cho sẵn. Đếm bằng `split()` thì bình luận dài nhất có 110 âm tiết. Bình luận ấy có vừa giới hạn không?

- A Chắc chắn vừa, vì mỗi âm tiết là đúng một token
- B Chắc chắn vừa, vì tokenizer gộp âm tiết nên token ít hơn
- C Chưa biết, phải đếm bằng chính tokenizer của model
- D Chắc chắn tràn, vì mỗi ký tự là một token

PHÒNG THEO [NOAI 2026 Singapore · Vòng sơ loại · Câu 227](#)

CÂU 3

Corpus là list từ `["bun"]*5 + ["sun", "sing", "king", "ring"]`. Một bạn đếm các cặp ký tự kề nhau trong vòng `for word in set(corpus)`, và phá hòa theo thứ tự từ điển. Bước gộp BPE đầu tiên đúng ra là gì, và code này gộp gì?

- A Đúng ra là `i,n` với 3 lần, code gộp `u,n` với 2 lần
- B Đúng ra là `u,n` với 6 lần, code gộp `i,n` với 3 lần
- C Cả hai đều gộp `u,n`, chỉ khác số đếm
- D Cả hai đều gộp `i,n`, chỉ khác số đếm

PHÒNG THEO [USA-NA-AIO 2026 · Vòng 1 · Part 5.1–5.2](#)

CÂU 4

Đây là hàm mã hóa câu cho một model sinh văn bản.

```
tokens = [BOS] + [vocab.get(w, UNK) for w in text.split()] + [EOS]
if len(tokens) < max_len:
    tokens += [PAD] * (max_len - len(tokens))
else:
    tokens = tokens[:max_len]
```

Với `max_len = 8` và một câu 9 từ, lỗi nghiêm trọng nhất là gì?

- A PAD mang ID 0, nên loss tính cả các vị trí đệm
- B Tokenizer theo từ làm vocabulary quá lớn, phải đổi sang BPE
- C BOS chiếm một chỗ, nên câu ngắn bị pad thiếu
- D Chuỗi bị cắt chỉ còn BOS và 7 từ, mất hẳn EOS

PHÒNG THEO NOAI 2026 Singapore · Chung kết · Câu 19

CÂU 5

Một đội lọc tin nhắn rác bằng Naive Bayes trên bag-of-words. Hai từ “khuyến” và “mãi” gần như luôn đi cùng nhau, nhưng mô hình tính chúng như hai bằng chứng độc lập. Hệ quả là gì?

- A Phép tính gọn, xác suất ra quá tự tin, nhưng nhãn thường vẫn đúng
- B Không có hệ quả gì, vì giả định độc lập luôn đúng khi đã tách từ cẩn thận
- C Không có hệ quả gì, vì Naive Bayes tự học tương tác giữa các từ
- D Mô hình không cần train, chỉ cần danh sách từ rác cho sẵn

PHÒNG THEO NOAI 2026 Singapore · Vòng sơ loại · Câu 169

CÂU 6

Vectorizer fit trên train có ba cột [dở, hay, tệ]. Một vectorizer khác fit riêng trên validation cho [dở, hay, ổn]. Shape đều đúng ba cột. Vì sao không đưa ma trận validation này vào classifier cũ được?

- A Vì TF-IDF fit trên validation không chứa được giá trị 0
- B Vì cột thứ ba đã đổi nghĩa, dù shape vẫn bằng nhau
- C Vì classifier nhị phân chỉ nhận đúng một cột đặc trưng
- D Vì số câu validation phải bằng số câu train

CÂU 7

Validation có nhiều lỗi gõ như “hayyy”, “haay”, không có trong vocabulary từ của train. Nên thử đổi biểu diễn theo hướng nào trước?

- A Dùng char n-gram, ví dụ analyzer="char_wb"
- B Tăng ngram_range theo từ lên (1, 3) để có thêm cột
- C Giảm c để model bớt tin vào từ chưa gặp
- D Bỏ dấu toàn bộ văn bản để gộp các biến thể lại

CÂU 8

Lớp cần tìm là nhận xét chê, nhãn 0, chiếm 20% validation. Đội báo $f1_score(y_true, y_pred)$ bằng 0.93. Con số này đo gì?

- A F1 của lớp chê, đúng thứ cần báo
- B F1 của lớp khen, vốn là lớp dương mặc định
- C Trung bình F1 của hai lớp, như macro-F1
- D Accuracy, vì hai nhãn đều là số nguyên

CÂU 9

Mỗi học sinh viết khoảng 20 nhận xét với văn phong riêng. Split ngẫu nhiên cho macro-F1 0.91, còn GroupKFold theo học sinh cho 0.74. Test là nhận xét của một khóa học sinh mới. Nên báo và chọn model theo số nào?

- A 0.74, vì test là những người viết chưa từng thấy
- B 0.825, trung bình hai cách để bớt lệch
- C 0.91, vì split ngẫu nhiên có nhiều dữ liệu train hơn
- D 0.91, rồi trừ đi một khoảng an toàn cố định

CÂU 10

Câu validation “quá đỉnh luôn” có mọi từ nằm ngoài vocabulary TF-IDF. Logistic regression đã học có bias $b = -0.2$. Model trả về xác suất lớp 1 bằng bao nhiêu?

- A 0.500, vì không có thông tin nên chia đều
- B 0.000, vì vector rỗng không thuộc lớp nào
- C Khoảng 0.450
- D Không có, vì code báo lỗi khi chuẩn hóa vector 0

Đọc thêm

- Jurafsky, Martin — [Speech and Language Processing, Words and Tokens](#) và [Text Classification](#). Đọc để nối token với đặc trưng và nhiệm vụ phân loại.
- Sennrich và cộng sự — [Neural Machine Translation of Rare Words with Subword Units](#). Nguồn gốc ứng dụng BPE cho từ hiếm trong NLP.
- Hugging Face — [BPE](#), [WordPiece](#). Xem sự khác nhau giữa học vocabulary và mã hóa câu mới.
- scikit-learn — [Text feature extraction](#). Kiểm chính xác IDF, chuẩn hóa và token_pattern của thư viện.
- VinAI — [PhoBERT](#). Đọc yêu cầu tách từ tiếng Việt trước khi dùng tokenizer này.