

Để giúp bạn so sánh hiệu suất phân lớp văn bản tiếng Việt không sử dụng đặc trưng topological (TDA) với trường hợp sử dụng TDA (như trong câu trả lời trước), tôi sẽ cung cấp các đoạn code tương tự nhưng loại bỏ phần trích xuất và tích hợp đặc trưng topological. Các bước vẫn bao gồm **tăng cường dữ liệu** (data augmentation) và **học sâu** (deep learning), sử dụng PhoBERT để phân loại cảm xúc (tích cực/tiêu cực) trên cùng bộ dữ liệu ví dụ (5,000 bình luận tiếng Việt). Code sẽ được thiết kế để chạy trên cả CPU và GPU, tương tự như yêu cầu trước.

Bài toán

- **Mục tiêu:** Phân loại cảm xúc của bình luận tiếng Việt (tích cực vs. tiêu cực).
 - **Dữ liệu:** 5,000 bình luận tiếng Việt, ví dụ:
 - Tích cực: "Sản phẩm rất tốt, pin bền, camera đẹp!"
 - Tiêu cực: "Máy nhanh nóng, hiệu năng kém, thất vọng."
 - **Phương pháp:**
 - **Tăng cường dữ liệu:** Sử dụng thay thế từ đồng nghĩa và back-translation để tăng dữ liệu lên 15,000 mẫu.
 - **Học sâu:** Sử dụng PhoBERT để trích xuất đặc trưng ngữ nghĩa (CLS token) và huấn luyện mô hình phân lớp.
 - **Không dùng TDA:** Loại bỏ đặc trưng topological (Betti numbers, số cạnh, v.v.) và chỉ dựa vào đặc trưng từ PhoBERT.
-

Các bước chi tiết

1. Tăng cường dữ liệu (Data Augmentation)

File này giống hệt file `data_augmentation.py` trước đó, vì tăng cường dữ liệu không liên quan đến TDA. Tôi sẽ giữ nguyên để đảm bảo tính nhất quán khi so sánh.



Show inline

- Kết quả: Tạo file `augmented_test_15k.csv` với 15,000 mẫu từ 5,000 mẫu gốc.
-

2. Học sâu (Deep Learning) mà không dùng đặc trưng Topological

File này thay thế cho `topo_features.py` và `deep_learning_with_topo.py`. Nó sử dụng PhoBERT để trích xuất đặc trưng ngữ nghĩa (CLS token) và huấn luyện mô hình phân lớp mà không cần đặc trưng topological. Code được thiết kế để chạy trên cả CPU và GPU.



deep_learning_no_topo.py
python

Show inline

- Sửa đổi so với phiên bản có TDA:

- **Loại bỏ topo_features:** Không trích xuất attention weights hoặc tính đặc trưng topological (Betti numbers, số cạnh).
- **Dataset đơn giản hơn:** `TextDataset` chỉ chứa `input_ids`, `attention_mask`, và `label`, không có `topo_features`.
- **Mô hình đơn giản hơn:** `PhoBERTClassifier` chỉ sử dụng CLS token từ PhoBERT, không nối với đặc trưng topological. Lớp `classifier` giảm kích thước đầu vào từ `768 + hidden_dim` xuống `768`.
- **Tương thích CPU/GPU:** Sử dụng `device = torch.device("cuda" if torch.cuda.is_available() else "cpu")` và chuyển mọi tensor/model sang `device`.

So sánh với phiên bản sử dụng TDA

Tiêu chí	Có TDA	Không TDA
Tăng cường dữ liệu	Giữ nguyên: Thay thế từ đồng nghĩa, back-translation, tạo 15,000 mẫu.	Giữ nguyên: Tương tự.
Đặc trưng đầu vào	Kết hợp CLS token (PhoBERT) và đặc trưng topological (Betti-0, Betti-1, số cạnh).	Chỉ dùng CLS token (PhoBERT).
Mô hình	PhoBERT + MLP (nối CLS token với topo_features).	PhoBERT + Linear (chỉ dùng CLS token).
Hiệu suất dự kiến	Độ chính xác ~90% (nhờ thông tin cấu trúc từ TDA).	Độ chính xác ~85% (thiếu thông tin cấu trúc).
Thời gian chạy	Chậm hơn do trích xuất attention weights và tính TDA.	Nhanh hơn, chỉ cần forward pass qua PhoBERT.
Yêu cầu tài nguyên	Cao hơn (bộ nhớ cho attention matrices và TDA).	Thấp hơn (chỉ cần bộ nhớ cho PhoBERT).
Tính tương thích	Chạy trên cả CPU/GPU với điều chỉnh device.	Chạy trên cả CPU/GPU với điều chỉnh device.

Kết quả mong đợi

- Không TDA:
 - Độ chính xác: ~85% (do chỉ dựa vào đặc trưng ngữ nghĩa từ PhoBERT).
 - Ví dụ:
 - "Sản phẩm rất tốt, pin bền!" → Tích cực (0.90).
 - "Máy nhanh nóng, hiệu năng kém." → Tiêu cực (0.88).
 - Thời gian huấn luyện nhanh hơn do không cần tính toán topological.
- Có TDA (từ câu trả lời trước):
 - Độ chính xác: ~90% (nhờ bổ sung thông tin cấu trúc từ Betti numbers, số cạnh).
 - Cải thiện khả năng phân biệt các mẫu phức tạp (ví dụ: câu có ngữ nghĩa ẩn).

Cách chạy

¹ Cài đặt môi trường:

```
bash
pip install torch transformers pandas py-vncorenlp sklearn
```

- Lưu ý: Không cần `riprser` hoặc `stats_count` vì không dùng TDA.

² Chạy từng file:

```
bash
python data_augmentation.py
python deep_learning_no_topo.py
```

³ Kiểm tra thiết bị:

```
python
import torch
print(torch.cuda.is_available()) # True nếu có GPU
print(torch.device("cuda" if torch.cuda.is_available() else "cpu"))
```

Lưu ý

- **So sánh hiệu suất:**

- Chạy cả hai phiên bản (có TDA và không TDA) trên cùng bộ dữ liệu để đo độ chính xác và thời gian chạy.
- Sử dụng `accuracy_score` (đã tích hợp trong code) để so sánh.

- **Tối ưu trên CPU:**

- Giảm `batch_size` (ví dụ: từ 16 xuống 8) nếu chạy trên CPU để tiết kiệm bộ nhớ.
- Giảm `max_len` (từ 128 xuống 64) để tăng tốc xử lý.

- **Dữ liệu thực tế:**

- Nếu bạn có bộ dữ liệu cụ thể, thay `small_gpt_web/test_5k.csv` bằng file của bạn và đảm bảo cột `sentence` và `label`.

Nếu bạn cần thêm phân tích so sánh (ví dụ: biểu đồ độ chính xác) hoặc muốn chạy thử trên dữ liệu cụ thể, hãy cung cấp thêm thông tin!