

Feature extraction

☑ Trắc nghiệm 12 📁 Thực hành 2 🔗 Chạy trên Kaggle ⬇ PDF bài

Phần thân của một model pretrained biến mỗi ảnh thành một vector, gọi là embedding hay feature. Ảnh giống nhau cho vector gần nhau, nên chỉ cần vector là làm được nhiều việc: tìm ảnh giống một ảnh cho trước, hoặc huấn luyện logistic regression, kNN ở chương 2 trên các vector đó. Cách làm này gọi là feature extraction: không huấn luyện lại model, chỉ dùng nó để lấy vector. Nó nhanh và là baseline nên thử đầu tiên.

MỤC TIÊU

- Lấy embedding từ model pretrained.
- Dùng embedding với logistic regression, kNN và tìm ảnh giống nhau.
- Tính cosine similarity và pooling có mask.
- Tránh leakage khi chọn encoder và xử lý dữ liệu.

Vì sao cần một biểu diễn khác?

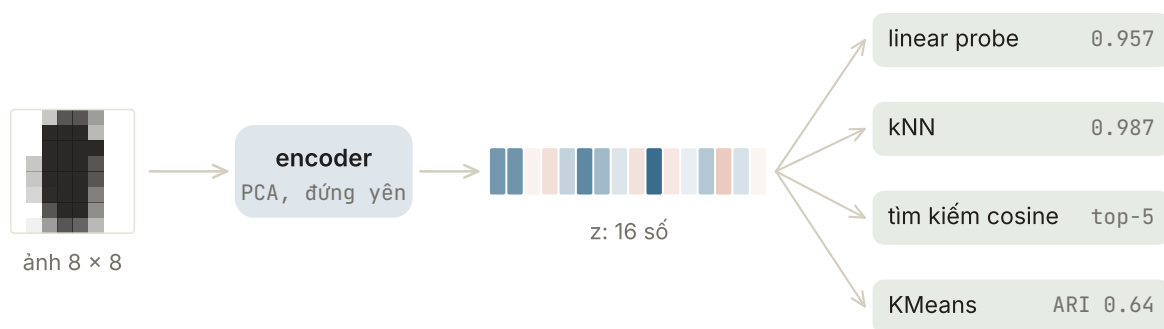
Hai ảnh cùng chiếc bàn có thể lệch vị trí vài pixel. Khoảng cách giữa hai mảng pixel lúc ấy lớn, dù ta coi chúng cùng nội dung. Tương tự, “tìm đường ngắn nhất” và “shortest path” không có nhiều ký tự chung, nhưng gần nhau về chủ đề. Ta muốn biểu diễn giữ những khác biệt có ích và giảm những khác biệt ít liên quan.

Encoder là một hàm f_θ nhận đầu vào $\mathbf{x}$ và trả về vector $\mathbf{z}$ có d thành phần:

$$\mathbf{z} = f_\theta(\mathbf{x}) \in \mathbb{R}^d.$$

θ là trọng số encoder. Feature extraction giữ θ cố định. Embedding là tên thường dùng cho vector biểu diễn này; một số tài liệu gọi chung là feature.

Không có quy định rằng tọa độ thứ 7 phải nghĩa là “có bàn”. Các tọa độ cùng nhau tạo ra hình học; việc gần xa có ích đến đâu còn phụ thuộc mục tiêu mà encoder đã học.



Một encoder đứng yên cho embedding; dùng lại cho phân loại, kNN, tìm kiếm, phân cụm.

Model có head phân loại trả về logits của nhãn cũ. Để lấy đặc trưng, ta thường dùng biểu diễn trước head hoặc API feature của model. Logits cũng là một biểu diễn, nhưng bị buộc vào các nhãn cũ và không mặc nhiên là lựa chọn tốt nhất. Với model có nhiều token, cần một quy tắc pooling để gộp các vector token thành một vector cho cả mẫu (mục cuối bài).

Với ResNet-50 của torchvision, thay `model.fc = torch.nn.Identity()` rồi gọi `features = model(x)` là được vector 2048 chiều trước head.

Trước học sâu, người ta tự thiết kế đặc trưng như tỉ lệ màu, cạnh, kết cấu; encoder học sâu tự học chúng từ dữ liệu. Có encoder tốt, nhiều việc chỉ còn là so vector:

- Tìm sản phẩm giống ảnh chụp đôi giày là tìm embedding gần nhất trong kho.
- Mở khóa bằng khuôn mặt với đúng một ảnh đăng ký là so embedding ảnh mới với embedding đã lưu theo một ngưỡng.

Khi mỗi lớp chỉ có vài ảnh, lấy trung bình embedding của chúng thành một tâm (prototype) cho mỗi lớp rồi gán ảnh mới vào tâm gần nhất.

HỎI NHANH

Encoder cho mỗi ảnh một vector 512 số, và có 100 ảnh. Ma trận embedding xếp mỗi mẫu một hàng có shape gì?

A (512, 100)

B (100, 512)

C (100,)

D (1, 51200)

HỎI NHANH

ResNet-50 học trên ImageNet có head trả về 1000 logits, và ngay trước head là một vector 2048 số. Câu lạc bộ cần phân loại ảnh thành 5 loại linh kiện máy tính, không loại nào trùng tên lớp ImageNet. Nên lấy đặc trưng nào cho linear probe?

A 1000 logits, vì đó là tầng có ý nghĩa rõ nhất

B Xác suất softmax của 1000 lớp, vì đã chuẩn hóa tổng bằng 1

C Vector 2048 số trước head, rồi học một head 5 lớp mới

D Pixel thô, vì nhãn mới không có trong ImageNet

Thử offline: encoder học từ dữ liệu train

Để mọi bước chạy trên CPU trong vài giây, bài dùng `load_digits` có sẵn trong `scikit-learn`: ảnh chữ số 8×8 , mỗi ảnh trải thành 64 pixel. Encoder là PCA (bài 2.9) giảm còn 16 tọa độ. Nó được học một lần trên tập train rồi đứng yên, đóng vai trò của một model pretrained, chỉ khác là nhỏ và học ngay tại chỗ thay vì tải về.

Các bước phía sau (chia dữ liệu, trích, học head, tìm kiếm, phân cụm) giữ nguyên khi thay PCA bằng một CNN hay Transformer; [script của chương](#) đi đúng quy trình ấy với DistilBERT cho văn bản.

Chia dữ liệu trước khi học PCA. PCA không dùng nhãn, nhưng học các hướng biến thiên từ input; nếu cho nó nhìn validation thì validation đã góp vào cách biểu diễn. Trong bài, tập giữ lại gọi là validation vì ta dùng nó đọc và so sánh cách làm. Không báo nó như một test cuối độc lập sau khi đã thử nhiều phương án.

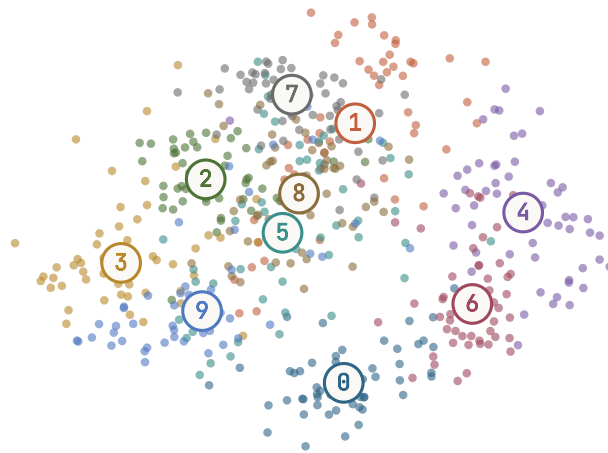
Python

```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split

digits = load_digits()
indices = np.arange(len(digits.target))
train_id, valid_id = train_test_split(
    indices, test_size=0.3, random_state=7, stratify=digits.target
)
X_train = digits.data[train_id] / 16.0
X_valid = digits.data[valid_id] / 16.0
y_train = digits.target[train_id]
y_valid = digits.target[valid_id]

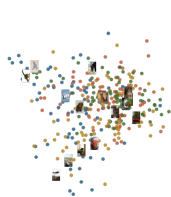
encoder = PCA(n_components=16, svd_solver="full")
Z_train = encoder.fit_transform(X_train)
Z_valid = encoder.transform(X_valid)
print("pixel:", X_train.shape, X_valid.shape)
print("embedding:", Z_train.shape, Z_valid.shape)
print("id tách roi:", len(set(train_id) & set(valid_id)) == 0)
```

Dòng cuối kiểm rằng không ID nào nằm ở cả hai tập. Điều đó chưa đủ để nói hai tập độc lập: `load_digits` không kèm thông tin người viết, nên chữ của cùng một người có thể nằm ở cả hai phía. Với kho ảnh câu lạc bộ, nhiều ảnh chụp liên tiếp trong một buổi hoạt động gần như trùng nhau; nếu mục tiêu là dự đoán cho một buổi mới thì phải chia theo buổi hoặc theo nguồn.



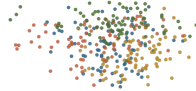
hai tọa độ đầu
trong 16 của z

Hai tọa độ đầu của embedding: cùng chữ số tụ lại gần nhau.



PCA của pixel thô (3072 số mỗi ảnh): bốn lớp trộn vào nhau

máy bay
ô tô
mèo
ếch



chuyển từ không gian pixel sang không gian đặc trưng

máy bay
ô tô
mèo
ếch



PCA của vector ResNet-18 (512 số mỗi ảnh): cùng lớp tụ lại

máy bay
ô tô
mèo
ếch



linear probe trên vector ResNet-18: accuracy 5-fold 0.90

máy bay
ô tô
mèo
ếch

Ảnh CIFAR-10 theo PCA của pixel thô, rồi theo PCA của vector đặc trưng ResNet-18.

Ma trận Z được tạo một lần. Khi dùng encoder neural thật, ta đặt encoder ở `eval()` và tắt gradient, chạy theo batch, lưu id cùng embedding. Nếu đầu vào có augmentation ngẫu nhiên, embedding cũng ngẫu nhiên; cần chọn rõ một phép biến đổi cố định, hoặc một tập biến đổi cố định rồi lấy trung bình, và kiểm cách đó trên validation.

HỎI NHANH

Kho ảnh câu lạc bộ có 40 buổi hoạt động, mỗi buổi khoảng 50 ảnh chụp liên tiếp. Chia ngẫu nhiên theo từng ảnh thì kNN trên embedding đạt 0.99 trên validation, nhưng dùng cho một buổi mới thì kém hẳn. Nguyên nhân hợp lý nhất là gì?

- A Encoder đã quá khớp với ảnh của câu lạc bộ
- B Những ảnh gần như trùng nhau của cùng một buổi nằm ở cả hai tập
- C kNN cần k lớn hơn khi kho có hơn 1000 ảnh
- D Embedding chưa được chuẩn hóa L2 trước khi tìm láng giềng

Linear probe hỏi biểu diễn có dễ phân loại không

Ta thử một head đơn giản: logistic regression. Nó học ma trận W và bias $\mathbf{b}$ từ Z , giữ encoder đứng yên:

$$\ell = W\mathbf{z} + \mathbf{b}, \quad p_k = \frac{e^{\ell_k}}{\sum_j e^{\ell_j}}.$$

Nếu $d = 16$ và có $K = 10$ lớp, head có $Kd + K = 170$ tham số. Gọi là linear probe vì nó thăm dò xem nhãn mới có thể đọc ra bằng một phép tuyến tính hay không.

Probe kém chưa chứng minh encoder vô dụng: nhãn mới có thể cần một ranh giới phi tuyến, head có thể chưa được chỉnh, hoặc split có vấn đề. Probe tốt cũng chỉ nói về nhãn và dữ liệu đang thử. Ít dữ liệu mà ảnh lại giống ImageNet là trường hợp hợp nhất để giữ encoder đứng yên và chỉ học head.

Python

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
```

```

probe = LogisticRegression(C=1.0, max_iter=1000)
probe.fit(Z_train, y_train)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(Z_train, y_train)
raw_knn = KNeighborsClassifier(n_neighbors=5)
raw_knn.fit(X_train, y_train)
for name, classifier, features in [
    ("linear probe", probe, Z_valid),
    ("kNN embedding", knn, Z_valid),
    ("kNN pixel", raw_knn, X_valid),
]:
    prediction = classifier.predict(features)
    print(name, "accuracy =", round(accuracy_score(y_valid, prediction), 4))
print("tham so head:", probe.coef_.size + probe.intercept_.size)

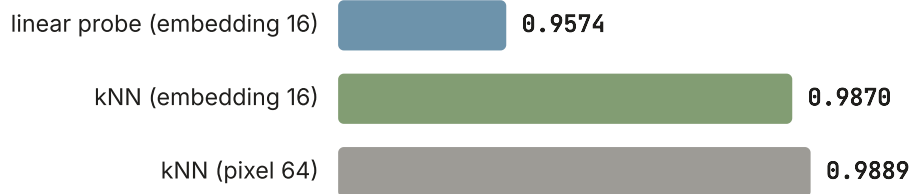
```

Ba dòng accuracy đo trên **cùng 540 ảnh validation**. kNN trên 16 tọa độ PCA đạt 0.9870, gần bằng kNN trên đủ 64 pixel (0.9889): chênh 0.0019, đúng một ảnh. Bỏ 48 trong 64 chiều mà accuracy của kNN gần như giữ nguyên. Feature extraction cần đúng điều đó: một vector ngắn hơn, rẻ hơn để lưu và tìm, vẫn giữ quan hệ gần xa có ích.

Linear probe thấp hơn (0.9574) dù dùng cùng embedding: các chữ số gom thành cụm cục bộ tốt, nhưng các siêu phẳng tuyến tính trong 16 chiều chưa tách chúng gọn như cách lằng giằng làm.

Vài điều cần nhớ khi đọc các con số:

- Chênh một ảnh trên một split không đủ để xếp hạng hai cách làm; muốn so chắc cần cross-validation hoặc nhiều split.
- 170 là tham số của head, chưa tính 16×64 hệ số và 64 giá trị trung bình mà PCA đã học.
- Nếu chỉnh số thành phần PCA theo chính validation này thì điểm của nó không còn là phép đo độc lập.



trục bắt đầu từ 0.94; 0.0019 $\approx$ đúng một ảnh trên 540

kNN trên 16 tọa độ PCA gần bằng kNN trên đủ 64 pixel.

HỎI NHANH

Trong linear probe, những tham số nào được học bằng nhãn của bài mới?

- A Chỉ W và b của head, encoder giữ nguyên
- B Cả encoder lẫn head, encoder với learning rate nhỏ
- C Chỉ encoder, còn head là logistic regression cố định
- D Không tham số nào, vì probe chỉ đo khoảng cách

HỎI NHANH

Một bạn đọc bảng trên và kết luận: “kNN trên pixel đạt 0.9889, cao hơn kNN trên embedding là 0.9870, vậy trích đặc trưng làm hại, nên bỏ encoder”. Phản biện nào đúng nhất?

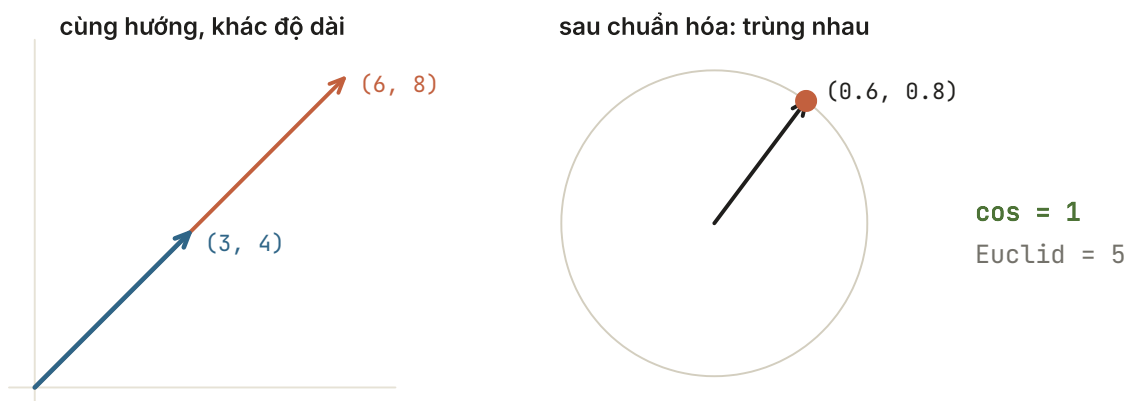
- A Kết luận đúng, vì cùng validation thì số lớn hơn là cách tốt hơn
- B Kết luận sai, vì embedding luôn tốt hơn pixel khi có encoder
- C Hai cách chỉ chênh đúng một ảnh trên 540, một cách chia chưa đủ để kết luận
- D Kết luận sai, vì phải so linear probe với kNN trên pixel mới công bằng

Tìm kiếm tương đồng: độ dài và hướng

Tìm kiếm tương đồng cần chọn khoảng cách. Hai vector $(3, 4)$ và $(6, 8)$ chỉ cùng một hướng nhưng khác độ dài. Tích vô hướng bằng $3 \cdot 6 + 4 \cdot 8 = 50$. Độ dài lần lượt bằng 5 và 10. Cosine similarity chia đi ảnh hưởng độ dài:

$$\cos(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a}^\top \mathbf{b}}{\|\mathbf{a}\|_2 \|\mathbf{b}\|_2} = 1.$$

Khoảng cách Euclid giữa chúng bằng 5. Vì vậy hai cách đo trả lời hai câu hỏi khác nhau. Cosine thường có ích khi hướng là điều ta muốn so; Euclid có thể có ích khi cả độ dài mang thông tin. Không chuẩn hóa theo thói quen nếu model hoặc validation cho thấy độ dài hữu ích.



$(3, 4)$ và $(6, 8)$ cùng hướng nên cosine bằng 1, dù khoảng cách Euclid là 5.

Python

```
a = np.array([3., 4.])
b = np.array([6., 8.])
cosine = (a @ b) / (np.linalg.norm(a) * np.linalg.norm(b))
print("dot:", a @ b)
print("cosine:", cosine)
print("Euclid:", np.linalg.norm(a - b))
```

Cosine nằm từ -1 đến 1 với vector khác 0. Nó không phải xác suất: cosine 0.8 không nghĩa là 80% chắc chắn cùng chủ đề. Vector 0 không có hướng nên phải xử lý riêng.

Sau khi chuẩn hóa mọi hàng về độ dài 1, xếp hạng cosine chỉ cần tích vô hướng, còn bình phương khoảng cách Euclid bằng $2 - 2 \cos$. Hai cách xếp hạng lúc đó tương đương. Chuẩn hóa mọi hàng bằng `V / np.sqrt((V ** 2).sum(axis=1, keepdims=True))`; sau đó `W @ W.T` là ma trận cosine, đường chéo toàn 1.

Tích vô hướng thô ưu ái vector dài: với $\mathbf{w} = (1, 1, 0)$, $A = (3, 3, 3)$ có tích vô hướng 6, lớn hơn $B = (1, 1.2, 0)$ với 2.2, nhưng cosine của B là 0.996, cao hơn 0.816 của A .

HỎI NHANH

Mọi vector đã được chuẩn hóa về độ dài 1. Hàng A có cosine 0.9 với truy vấn, hàng B có cosine 0. Bình phương khoảng cách Euclid từ truy vấn tới A và tới B lần lượt là bao nhiêu?

A 0.1 và 1.0

B 1.8 và 0

C 0.9 và 0

D 0.2 và 2.0

Ta dùng một ảnh validation làm truy vấn và tìm trong tập train. Chọn một hàng truy vấn không dùng nhãn để xếp hạng; nhãn chỉ giúp đọc kết quả sau đó.

Python

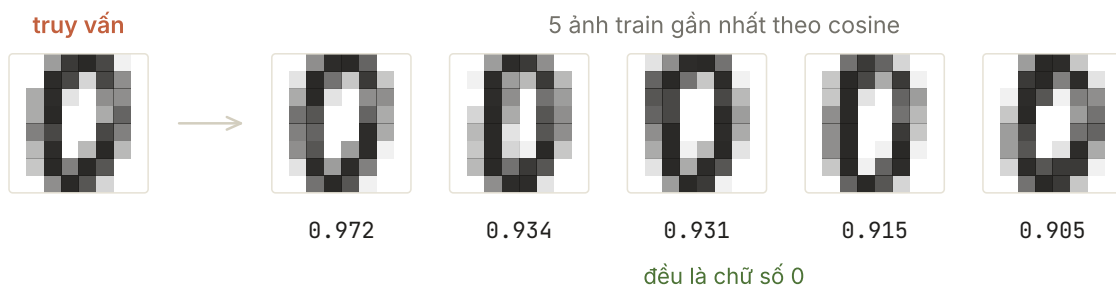
```
def normalize_rows(matrix):
    norms = np.linalg.norm(matrix, axis=1, keepdims=True)
    return matrix / np.maximum(norms, 1e-12)

database = normalize_rows(Z_train)
query = normalize_rows(Z_valid[:1])
similarity = (query @ database.T)[0]
nearest = np.argsort(-similarity, kind="stable")[:5]
print("nhân truy vấn:", int(y_valid[0]))
print("nhân 5 hàng gần:", y_train[nearest].tolist())
```

```
print("cosine:", similarity[nearest].round(3).tolist())
print("id database:", train_id[nearest].tolist())
```

Nhìn nhãn của năm hàng gần nhất cùng cosine để xem phép đo đang trả về gì. Một truy vấn thành công không đủ đo cả bộ tìm kiếm. Khi có nhãn liên quan cho nhiều truy vấn, dùng metric như recall@k hoặc MAP@k theo yêu cầu bài.

Chi phí tìm tuyến tính cho một truy vấn là $O(nd)$ với n vector trong kho, mỗi vector có d số; kho lớn cần xem lại thời gian và bộ nhớ trước khi dùng chỉ mục gần đúng.



Năm ảnh train gần nhất với một chữ số 0 theo cosine: cả năm đều là 0.

HỎI NHANH

`similarity = np.array([0.2, 0.9, -0.4, 0.7])` là cosine giữa truy vấn và bốn hàng của kho. `np.argsort(-similarity)[:2]` trả về gì?

A [1, 3]

B [2, 0]

C [0.9, 0.7]

D [3, 1]

Pooling văn bản phải tôn trọng padding

Đổi từ ảnh sang chữ, encoder có thể trả `last_hidden_state` shape (B, L, D) : một vector D số cho mỗi token. Muốn lấy một vector câu bằng trung bình, chỉ cộng các token hợp lệ. Với mask m_t bằng 1 ở token thật và 0 ở padding:

$$\mathbf{z} = \frac{\sum_{t=1}^L m_t \mathbf{h}_t}{\sum_{t=1}^L m_t}.$$

Ở ví dụ tay, câu đầu có hai vector $(1, 3)$ và $(3, 1)$ nên trung bình là $(2, 2)$. Vị trí padding có giá trị bất kỳ; nó không được làm lệch trung bình. Code dưới đây dựng tay một hidden nhỏ, đặt 999 ở vị trí padding để lỗi lộ rõ. Với model thật, hidden là `last_hidden_state` và mask là `attention_mask` mà tokenizer trả về.

Python

```
import torch

hidden = torch.tensor([
    [[1., 3.], [3., 1.], [999., 999.]],
    [[2., 0.], [0., 2.], [2., 2.]],
])
mask = torch.tensor([[1, 1, 0], [1, 1, 1]])
weights = mask.unsqueeze(-1).float()
pooled = (hidden * weights).sum(dim=1) / weights.sum(dim=1)
reference = []
for row, valid_tokens in zip(hidden.numpy(), mask.numpy()):
    reference.append(row[valid_tokens == 1].mean(axis=0))
print("shape:", tuple(pooled.shape))
print("masked mean:", pooled.numpy().round(3))
print("mean sai:", hidden.mean(dim=1).numpy().round(3))
print("doi chieu:", np.allclose(pooled.numpy(), np.array(reference)))
```

Hàng đầu của `mean sai` bị padding kéo đi rất xa. Dòng đối chiếu dùng cách lọc token rồi trung bình, độc lập với broadcasting của bản tensor.

Với model thật, còn phải quyết định special token như `[CLS]`, `[SEP]` có được pooling hay không. Mean pooling trên DistilBERT cho một baseline; model được huấn luyện riêng cho tương đồng câu thường hợp với tìm kiếm hơn. Chọn cái nào thì đo trên validation.

HỎI NHANH

Mask của một câu là (1, 1, 1, 0, 0), và tổng ba vector của các token thật là (6, 3). Masked mean bằng bao nhiêu?

- A (1.2, 0.6)
- B (6, 3)
- C Còn tùy giá trị của hai vector PAD
- D (2, 1)

HỎI NHANH

hidden có shape (32, 20, 768), mask có shape (32, 20). Dòng `(hidden * mask).sum(dim=1)` làm gì?

- A Chạy đúng, cho tổng các token thật với shape (32, 768)
- B Báo lỗi shape, phải dùng `mask.unsqueeze(-1)`
- C Chạy được, nhưng cộng cả padding vào tổng
- D Cho shape (32, 20), vì nhân theo từng token

Phân cụm và đọc giới hạn của nó

Nếu nhiều tài liệu chưa có nhãn, ta có thể phân cụm embedding để xem các nhóm. KMeans học tâm cụm trên train, còn validation được gán vào tâm gần nhất. Chỉ số cụm không phải tên chủ đề; đổi số thứ tự các cụm không đổi chất lượng. Dữ liệu của bài có nhãn chữ số nên dùng adjusted Rand index, một phép đo không phụ thuộc việc hoán vị tên cụm.

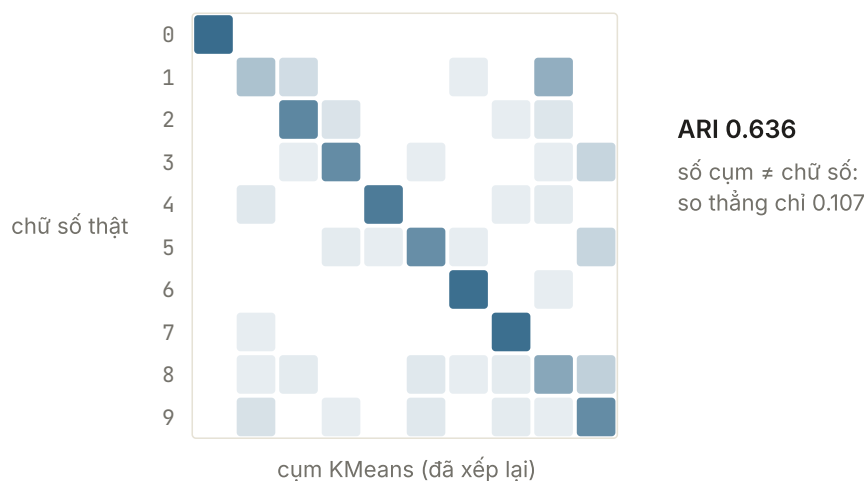
Python

```
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score

clusterer = KMeans(n_clusters=10, n_init=10, random_state=0)
clusterer.fit(Z_train)
valid_clusters = clusterer.predict(Z_valid)
print("ARI validation:", round(adjusted_rand_score(y_valid, valid_clusters), 4))
print("accuracy so cum voi nhan:", round(accuracy_score(y_valid, valid_clusters), 4))
```

ARI 0.6364 cho thấy cụm khớp nhãn chữ số khá tốt nhưng còn xa mức 1. Dòng thứ hai so thẳng số thứ tự cụm với nhãn và chỉ được 0.1074, gần mức đoán bừa giữa 10 lớp: KMeans đánh số cụm tùy ý, cụm 0 không có lý do gì trùng chữ số 0. Accuracy của classifier và ARI của clustering có định nghĩa khác nhau nên không xếp chung một cột điểm.

Với tài liệu câu lạc bộ, phải đọc vài mẫu mỗi cụm và kiểm những cụm lẫn nhiều chủ đề; một hình scatter hai chiều đẹp không chứng minh phân cụm tốt trong toàn bộ không gian.



Chữ số thật × cụm KMeans: phần lớn nằm trên đường chéo, vài chữ số bị tách hoặc gộp.

HỎI NHANH

Muốn biết mỗi cụm ứng với chữ số nào để tính một accuracy dễ đọc cho phép phân cụm ở trên, ta làm thế nào cho đúng?

- A Gán cho mỗi cụm nhãn chiếm đa số trên train, rồi áp lên validation
- B Gán cho mỗi cụm nhãn chiếm đa số trên chính validation rồi chấm
- C Chạy lại KMeans với random_state khác tới khi accuracy cao
- D Không làm được, vì KMeans không nhận nhãn khi học

Khi đi thi

- Trích embedding một lần, lưu kèm ID; học logistic regression hoặc kNN trên đó làm baseline trước khi fine-tune.
- PCA, scaler hay head đều fit trên phần train của từng fold.
- Đọc luật về model pretrained: có đề cấm hẳn, khi đó encoder phải tự học từ dữ liệu được cấp.

Tóm tắt

- Encoder biến mỗi mẫu thành một vector; feature extraction giữ encoder cố định, chỉ học phần phía trên.
- Linear probe, kNN, tìm kiếm và phân cụm dùng lại cùng ma trận embedding.
- Cosine so hướng, không phải xác suất; pooling văn bản phải bỏ PAD bằng mask.
- Chia dữ liệu trước khi học encoder; ảnh gần trùng nhau phải nằm cùng một phía.
- Thay PCA bằng CNN hay Transformer pretrained thì quy trình không đổi; chất lượng vẫn đo trên validation.

Kiểm tra cuối bài

CÂU 1

Linear probe hai lớp là logistic regression trên $\mathbf{z} = f_{\theta}(\mathbf{x})$, với f_{θ} là một CNN pretrained đã bị đóng băng, và dự đoán lớp 1 khi $p \geq 0.5$. Ranh giới quyết định có dạng gì?

- A Siêu phẳng trong không gian pixel, vì logistic regression là tuyến tính
- B Siêu phẳng theo embedding, còn theo pixel thì có thể cong tùy ý
- C Không là siêu phẳng ở đâu cả, vì sigmoid là hàm phi tuyến
- D Siêu phẳng theo cả embedding lẫn pixel, vì encoder đã bị đóng băng

PHÒNG THEO CMU 10-601 · Exam 2 Practice · Q2.3(b)

CÂU 2

Muốn thêm một bước PCA trước linear probe trên embedding của train, và chọn số thành phần bằng 5-fold cross-validation. Làm thế nào để giữ đúng ranh giới giữa các fold?

- A Fit PCA một lần trên toàn bộ train, vì encoder đã bị đóng băng
- B Ở mỗi fold, fit PCA trên phần validation rồi chiếu phần train
- C Ở mỗi fold, fit PCA riêng cho phần train và cho phần validation
- D Ở mỗi fold, fit PCA trên phần train rồi chiếu phần validation

CÂU 3

Trong ví dụ của bài, thay $n_components=16$ bằng $n_components=64$, tức giữ đủ mọi thành phần, rồi chạy lại kNN 5 láng giềng trên cùng cách chia. Accuracy trên validation so với các số trong bảng sẽ thế nào?

- A Cao hơn 0.9889, vì PCA lọc bớt nhiều khối pixel
- B Bằng 0.9870, vì vẫn là embedding PCA như trước
- C Đúng bằng 0.9889 của kNN trên pixel
- D Thấp hơn 0.9870, vì 64 chiều nhiều hơn 16 chiều

PHÒNG THEO CMU 10-601 · Exam 3 Practice · Q9.2(ii)

CÂU 4

Trong ví dụ của bài, W là Z_train sau khi chuẩn hóa từng hàng về độ dài 1, có shape $(1257, 16)$. Ma trận cosine giữa mọi cặp ảnh train $S = W @ W.T$ có shape $(1257, 1257)$. Hạng (rank) của S lớn nhất là bao nhiêu?

- A 16
- B 1257
- C 1241
- D 256

PHÒNG THEO USA-NA-AIO 2026 · Vòng 1 · Part 5.10

CÂU 5

Kho tìm kiếm có 50 000 vector, mỗi vector 100 số float32. Lưu sẵn ma trận cosine S cho mọi cặp thì tốn bao nhiêu bộ nhớ, so với chính ma trận embedding W?

- A S khoảng 20 MB, bằng W
- B S khoảng 20 GB, gấp 1000 lần W
- C S khoảng 10 GB, gấp 500 lần W
- D S khoảng 2.5 GB, gấp 125 lần W

PHÒNG THEO [USA-NA-AIO 2026 · Vòng 1 · Part 5.14](#)

CÂU 6

Matrix factorization trong hệ gợi ý học vector cho người dùng và vector cho phim, rồi so chúng trực tiếp bằng tích vô hướng. Một đội muốn làm tương tự: lấy embedding ảnh từ một ResNet và embedding mô tả từ DistilBERT, dùng PCA đưa cả hai về 128 chiều, rồi tìm ảnh bằng cosine với mô tả. Câu nào đúng?

- A Làm được, vì sau PCA hai loại vector cùng có 128 chiều
- B Làm được, nếu chuẩn hóa L2 cả hai loại vector trước
- C Không được, vì tìm láng giềng chỉ dùng Euclid, không dùng cosine
- D Không được, vì hai encoder không học chung một không gian

PHÒNG THEO [CMU 10-601 · Exam 3 Practice · Q7.4](#)

CÂU 7

Một đội đổi processor từ “resize 224” sang “resize 256 rồi cắt giữa 224”, nhưng vẫn dùng cache embedding của tập train tạo bằng processor cũ, còn tập test thì trích bằng processor mới. Cả hai đều có shape $(n, 2048)$. Hệ quả là gì?

- A Hai tập không còn dùng cùng một hàm trích, phải trích lại train
- B Không sao, vì cùng model và cùng 2048 chiều nên vẫn so được
- C Chỉ cần chuẩn hóa L2 lại cả hai phía là hết lệch
- D Chỉ cần đổi tên cache để ghi nhận processor mới

CÂU 8

Một đội trích embedding bằng đúng transform dùng khi train: RandomResizedCrop và lật ngang ngẫu nhiên. Chạy lại toàn bộ, accuracy kNN trên validation đổi từ 0.91 sang 0.89, dù không sửa dòng nào. Sửa thế nào là hợp lý nhất?

- A Tăng k của kNN để trung bình bớt dao động
- B Trích bằng một transform cố định, như resize rồi cắt giữa
- C Đặt `requires_grad=False` cho mọi tham số của encoder
- D Chuẩn hóa L2 embedding trước khi tìm láng giềng

CÂU 9

Trích embedding cho 20 000 ảnh mất 12 phút, còn học một logistic regression trên embedding mất 20 giây. Cần thử 6 cấu hình head. Cách thứ nhất trích lại embedding trước mỗi lần thử. Cách thứ hai trích một lần rồi dùng cache. Tổng thời gian của hai cách là bao nhiêu?

- A Trích lại mất 72 phút, dùng cache mất 14 phút
- B Trích lại mất 74 phút, dùng cache mất 12 phút
- C Trích lại mất 74 phút, dùng cache mất 14 phút
- D Trích lại mất 74 phút, dùng cache mất 2 phút

CÂU 10

Trên cùng embedding 16 chiều, linear probe đạt 0.9574, còn kNN đạt 0.9870. Giả thuyết: các chữ số gom cụm tốt, nhưng khó tách bằng siêu phẳng. Thí nghiệm nào kiểm trực tiếp giả thuyết này?

- A Thay PCA bằng một encoder pretrained lớn, rồi chạy lại probe
- B Tăng `n_neighbors` của kNN tới khi điểm bằng probe
- C Chạy lại probe với `max_iter` lớn hơn trên cùng Z
- D Thử một head phi tuyến, như SVM RBF, trên cùng Z và cùng cách chia

Đọc thêm

- [scikit-learn — PCA](#): kiểm biến đổi học trên train và số thành phần.
- [scikit-learn — nearest neighbors](#): khoảng cách và chi phí truy vấn trên embedding.
- [DistilBERT model card](#): xem nhiệm vụ học trước, ngôn ngữ, license và cách lấy hidden state.
- [Torchvision — feature extraction](#): lấy output ở các tầng khác nhau thay vì nhằm với logits.

- Dive into Deep Learning – fine-tuning: nối việc giữ encoder với việc học hoặc cập nhật head.

Tải gói recipe của chương – gồm các module phụ và hướng dẫn chuẩn bị asset.