ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

CO3117 - Học máy

Báo cáo Học phần Mở rộng

Ứng dụng giải thuật Viterbi vào nhận diện hợp âm bài hát

Giảng viên hướng dẫn: Lê Thành Sách

Sinh viên thực hiện:

Nguyễn Thái Học	2311100
Nguyễn Thái Sơn	2312968
Lương Minh Thuận	2313348
Phạm Trần Minh Trí	2313622

THÀNH PHỐ HỒ CHÍ MINH, 11/2025

Mục lục

1 Giới thiệu

Phần này tóm tắt nhanh nội dung và phương pháp được triển khai trong bộ mã tham khảo (notebook kèm theo) cho bài toán nhận diện hợp âm từ file âm thanh.

1.1 Bối cảnh

Nhận diện hợp âm (chord recognition) là một bài toán thường gặp trong xử lý tín hiệu âm thanh và âm nhạc tính toán. Mục tiêu là dự đoán chuỗi hợp âm xuất hiện theo thời gian trên một bản nhạc. Ứng dụng bao gồm phân tích âm nhạc, tạo nhạc tự động, và hỗ trợ học nhạc.

1.2 Tổng quan phương pháp

Bộ mã tham khảo thực hiện qui trình chính sau:

- Tiền xử lý âm thanh và trích xuất đặc trưng Chroma (chroma features) từ các file âm thanh bằng thư viện `librosa`.
- Chuẩn hoá và ghép nhãn hợp âm (annotation) từ file .lab sang khung thời gian của các frame Chroma.
- Gom nhãn về tập trạng thái rút gọn gồm 24 hợp âm cơ bản (12 nốt x maj,min) cùng trạng thái "No-Chord"và "Complex"cho những hợp âm không trực tiếp ánh xạ.
- Tổng hợp dữ liệu từ nhiều bài hát (concatenate) để tạo gói dữ liệu huấn luyện (X: chuỗi quan sát chroma, y: chuỗi trạng thái, lengths: số frame mỗi bài).
- Tính toán tham số HMM: phân phối ban đầu π, ma trận chuyển trạng thái A (với làm mịn) và mô hình phát xạ B được ước lượng bằng các Gaussian Mixture Models (GMM) cho mỗi trạng thái.
- Sử dụng thuật toán Viterbi để suy đoán chuỗi hợp âm cho file âm thanh mới dựa trên tham số HMM đã huấn luyện.

1.3 Xử lý tín hiệu số (DSP) trong trích xuất đặc trưng

Phần này trình bày chi tiết cách các kỹ thuật xử lý tín hiệu số được áp dụng trong code để chuyển đổi tín hiệu âm thanh thô thành các đặc trưng phù hợp cho mô hình học máy.

1.3.1 Sample Rate (Tần số lấy mẫu)

Sample Rate là số lượng mẫu (samples) được ghi lại mỗi giây từ tín hiệu âm thanh liên tục. Trong code, giá trị được thiết lập là:

```
SAMPLE_RATE = 22050  # Hz (samples/second)
```

Ý nghĩa và lựa chọn:

- Theo định lý Nyquist-Shannon, để tái tạo hoàn hảo tín hiệu gốc, tần số lấy mẫu phải ít nhất gấp đôi tần số cao nhất trong tín hiệu. Với $f_s = 22050$ Hz, ta có thể biểu diễn chính xác các tần số lên đến $\frac{22050}{2} = 11025$ Hz.
- Tai người nghe được từ khoảng 20 Hz đến 20 kHz. Tuy nhiên, với phân tích âm nhạc, đặc biệt là nhận diện hợp âm, các thành phần tần số quan trọng thường nằm dưới 10 kHz. Do đó, 22.05 kHz là một lựa chọn phù hợp giúp giảm khối lượng dữ liệu (so với 44.1 kHz chuẩn CD) mà vẫn giữ được thông tin âm nhạc cần thiết.
- Khi gọi `librosa.load(audio_path, sr=22050)`, nếu file âm thanh gốc có sample rate khác (ví dụ 44.1 kHz), `librosa` sẽ tự động resample (lấy mẫu lại) về 22050 Hz.

Ảnh hưởng đến pipeline:

- Sample rate quyết định độ phân giải tần số của tín hiệu số.
- Nó ảnh hưởng trực tiếp đến số lượng mẫu trong tín hiệu: với một bài hát dài T giây, số mẫu là $N = T \times f_s$.
- Ví dụ: một bài hát 3 phút (180 giây) sẽ có $180 \times 22050 = 3{,}969{,}000$ mẫu.

1.3.2 Hop Length (Độ dịch chuyển cửa sổ)

Hop Length xác định số mẫu mà cửa sổ phân tích dịch chuyển giữa các frame liên tiếp khi tính toán biến đổi thời gian-tần số (như STFT hoặc CQT). Trong code:

```
HOP_LENGTH = 512  # samples
```

Ý nghĩa vật lý:

- Hop length quyết định độ phân giải thời gian của các đặc trưng được trích xuất.
- Với $h = 512$ mẫu và $f_s = 22050$ Hz, khoảng thời gian giữa hai frame liên tiếp là:

$$\Delta t = \frac{h}{f_s} = \frac{512}{22050} \approx 0.0232 \text{ giây} \approx 23.2 \text{ ms} \tag{1.1}$$

- Do đó, ta thu được khoảng $\frac{1}{0.0232} \approx 43$ frame mỗi giây (như ghi chú trong code: `# (~43 frames/sec)`).

Trade-off giữa độ phân giải thời gian và tần số:

- *Hop length nhỏ* (ví dụ 128): Nhiều frame hơn mỗi giây $\Rightarrow$ độ phân giải thời gian cao (theo dõi biến đổi nhanh), nhưng tăng khối lượng tính toán và có thể gây nhiễu do các frame chồng lấp nhiều.
- *Hop length lớn* (ví dụ 2048): Ít frame hơn $\Rightarrow$ giảm tính toán, nhưng mất mát thông tin biến đổi nhanh (độ phân giải thời gian thấp).
- Giá trị 512 là một cân bằng phổ biến: đủ chi tiết để bắt được sự thay đổi hợp âm (thường diễn ra trong vài trăm ms) mà không quá nặng về tính toán.

1.3.3 Quá trình trích xuất Chroma từ tín hiệu âm thanh

Trong phương thức `extract_chroma` của lớp `AudioProcessor`, các bước DSP chính như sau:
Bước 1: Nạp và resample tín hiệu

```
y, sr = librosa.load(audio_path, sr=self.sr)
```

- `y`: mảng numpy 1D chứa các giá trị biên độ của tín hiệu âm thanh (thường được chuẩn hoá trong khoảng $[-1, 1]$).
- Chiều dài: $N = \text{len}(y)$ mẫu.

Bước 2: Tính Chromagram bằng CQT (Constant-Q Transform)

```
chroma = librosa.feature.chroma_cqt(y=y, sr=sr,
                                    hop_length=self.hop_length).T
```

- **CQT** là một biến đổi thời gian-tần số tương tự STFT nhưng có độ phân giải tần số theo thang logarit (phù hợp với âm nhạc vì các nốt nhạc cách nhau theo tỷ lệ logarit).
- Chromagram tập hợp năng lượng của tất cả các octave về 12 lớp âm (pitch class): C, C#, D, ..., B. Mỗi hàng của chroma vector biểu diễn năng lượng tương đối của 12 nốt nhạc tại một frame thời gian.
- Đầu ra của `chroma_cqt`: ma trận $(12 \times M)$ với $M = \lfloor \frac{N - n_{\text{fft}}}{h} \rfloor + 1 \approx \frac{N}{h}$ là số frame.
- Phép chuyển vị `.T` biến đổi thành $(M \times 12)$ để mỗi hàng là một quan sát (frame).

Bước 3: Chuẩn hoá Chroma

```
chroma_sum = np.sum(chroma, axis=1, keepdims=True)
chroma_sum[chroma_sum == 0] = 1  # Tránh chia cho 0
chroma_features = chroma / chroma_sum
```

- Mỗi vector chroma (hàng) được chuẩn hoá sao cho tổng các thành phần bằng 1. Điều này giúp đặc trưng không bị ảnh hưởng bởi độ lớn tổng thể của tín hiệu (loudness).
- Kết quả: mỗi frame có vector 12 chiều biểu diễn *phân phối xác suất* của 12 pitch class.

1.3.4 Căn chỉnh nhãn với frame

Sau khi có ma trận chroma $(M \times 12)$, cần ánh xạ nhãn hợp âm (từ file `.lab`) sang từng frame:

```
frame_times = librosa.frames_to_time(np.arange(n_frames),
                                     sr=self.sr,
                                     hop_length=self.hop_length)
```

- Hàm `frames_to_time` tính thời gian (tính bằng giây) của tâm mỗi frame:

$$t_i = \frac{i \times h}{f_s}, \quad i = 0, 1, \ldots, M-1 \tag{1.2}$$

- File `.lab` chứa các dòng dạng: `start_time end_time chord_label`.
- Với mỗi khoảng thời gian $[\text{start}, \text{end})$ trong file nhãn, ta tìm các frame có $t_i \in [\text{start}, \text{end})$ và gán nhãn tương ứng.

Ví dụ minh hoạ:

- Giả sử một đoạn hợp âm C:maj kéo dài từ giây thứ 0.5 đến 2.0.
- Với hop length 512 và sample rate 22050, frame thứ i tương ứng thời gian $t_i = 0.0232i$ giây.
- Các frame có index từ $\lceil 0.5/0.0232 \rceil = 22$ đến $\lfloor 2.0/0.0232 \rfloor = 86$ sẽ được gán nhãn `C:maj`.

1.4 Triển khai chi tiết và các lựa chọn kỹ thuật

Một số điểm kỹ thuật đáng chú ý trong notebook:

1.4.1 Kiến trúc class AudioProcessor

- **AudioProcessor**: lớp xử lý âm thanh được khởi tạo với hai tham số DSP chính:

```
def __init__(self, sr: int = 22050, hop_length: int = 512)
```

- Phương thức `extract_chroma(audio_path)`: trích xuất đặc trưng chroma độc lập, trả về ma trận $(M \times 12)$.
- Phương thức `extract_and_align(audio_path, lab_path)`: kết hợp trích xuất chroma và căn chỉnh nhãn, trả về tuple `(chroma_features, aligned_labels_id)`.
- Thiết kế module này cho phép tái sử dụng dễ dàng: cùng một instance `AudioProcessor` có thể xử lý nhiều file với tham số DSP nhất quán.

1.4.2 Đơn giản hoá không gian trạng thái

- Nhãn gốc trong dataset có thể rất đa dạng (ví dụ: `C:maj7`, `Db:min`, `G:sus4`, `F#:dim`).
- Phương thức `simplify_chord` ánh xạ về 26 trạng thái:
  - 24 hợp âm cơ bản: 12 nốt $\times$ {major, minor}
  - Trạng thái `N`: không có hợp âm (silence, noise)
  - Trạng thái `Complex`: các hợp âm diminished, augmented, suspended không thể phân loại đơn giản
- Ánh xạ enharmonic: Db $\rightarrow$ C#, Eb $\rightarrow$ D#, Gb $\rightarrow$ F#, Ab $\rightarrow$ G#, Bb $\rightarrow$ A# để tránh trùng lặp.
- Lý do: giảm độ phức tạp mô hình (26 trạng thái thay vì hàng trăm), giúp GMM huấn luyện tốt hơn với dữ liệu hạn chế.

1.4.3 Tính toán tham số HMM

- **Ma trận chuyển trạng thái** A: được khởi tạo với giá trị smoothing nhỏ (10^{-5}) để tránh xác suất bằng 0 (Laplace smoothing), sau đó cộng dồn số lần chuyển từ trạng thái i sang j và chuẩn hoá theo hàng.
- **Phân phối ban đầu** π: đếm trạng thái đầu tiên của mỗi bài hát, cộng dồn và chuẩn hoá.

- **Mô hình phát xạ** B: mỗi trạng thái được mô hình hoá bởi một GMM với 40 thành phần Gaussian (có thể điều chỉnh):
  - GMM cho phép mô hình hoá phân phối phức tạp của chroma vector trong mỗi trạng thái (do có nhiều cách chơi cùng một hợp âm, âm sắc nhạc cụ khác nhau, nhiễu).
  - Covariance type `full`: ma trận hiệp phương sai đầy đủ để bắt được tương quan giữa các pitch class.
  - Nếu một trạng thái không có đủ quan sát (ví dụ hợp âm hiếm), GMM không được huấn luyện (gán `None`) để tránh overfitting.

1.4.4 Thuật toán Viterbi trong miền log

- Viterbi tìm chuỗi trạng thái có xác suất cao nhất $\arg\max_{s_1,\ldots,s_M} P(s_1,\ldots,s_M \mid x_1,\ldots,x_M)$.
- Để tránh underflow khi nhân nhiều xác suất nhỏ, tất cả tính toán được thực hiện trong miền log:

$$\log P(s_1,\ldots,s_M,x_1,\ldots,x_M) = \log \pi_{s_1} + \sum_{t=1}^{M-1} \log A_{s_t,s_{t+1}} + \sum_{t=1}^{M} \log B_{s_t}(x_t) \tag{1.3}$$

- GMM cung cấp `score_samples()` trả về log-likelihood trực tiếp.
- Ma trận $V[t,j]$ lưu log-xác suất cao nhất để đạt trạng thái j tại thời điểm t.

1.4.5 Ảnh hưởng của tham số DSP đến hiệu năng

- **Sample rate cao hơn** (ví dụ 44.1 kHz): tăng độ chính xác tần số nhưng tăng gấp đôi kích thước dữ liệu và thời gian tính toán mà lợi ích cho nhận diện hợp âm không đáng kể.
- **Hop length nhỏ hơn** (ví dụ 256): độ phân giải thời gian cao hơn (khoảng 86 frame/giây), có thể bắt được sự chuyển hợp âm nhanh hơn nhưng tăng số frame cần xử lý trong Viterbi (độ phức tạp $O(M \times N_{\text{states}}^2)$).
- **Số thành phần GMM**: 40 thành phần là kết quả sau khi thử nghiệm với BIC/AIC (mã tham khảo có phần comment về grid search). Quá ít thành phần dẫn đến underfitting, quá nhiều dẫn đến overfitting và tăng thời gian huấn luyện.

1.5 Ghi chú và hướng dẫn chạy

- Notebook tham khảo: file notebook đính kèm (tên file trong phần đính kèm). Notebook chứa cả phần tiền xử lý, huấn luyện GMM, tính toán tham số HMM, và ví dụ áp dụng

Viterbi.

- Để tái tạo kết quả trên máy của bạn: chuẩn bị thư mục chứa các file âm thanh (.wav/.mp3) và file nhãn (.lab) tương ứng; chỉnh biến đường dẫn trong notebook; chạy tuần tự các cell theo thứ tự (extract -> aggregate -> train -> predict).
- Nếu muốn, tôi có thể chèn một hình minh hoạ (ví dụ heatmap chroma hoặc sơ đồ ma trận A) và/hoặc thêm tham chiếu BibTeX cho các thư viện (librosa, sklearn). Hãy cho biết yêu cầu cụ thể (kích thước, vị trí ảnh, tên file ảnh) nếu cần.

1.6 Kết luận ngắn

Quy trình trong notebook là một pipeline điển hình cho bài toán nhận diện hợp âm: từ trích xuất chroma đến mô hình hóa phát xạ bằng GMM và suy giải bằng HMM/Viterbi. Các phần tiếp theo sẽ trình bày chi tiết về cơ sở lý thuyết HMM, thuật toán Viterbi, kết quả thực nghiệm và đánh giá.

2 Cơ sở lý thuyết

2.1 Mô hình Markov ẩn (Hidden Markov Model - HMM)

2.1.1 Định nghĩa và thành phần

Hidden Markov Model là một mô hình xác suất thống kê được sử dụng để mô hình hoá các chuỗi có cấu trúc tuần tự, trong đó trạng thái thực sự (hidden state) không quan sát được trực tiếp, mà chỉ có thể suy luận thông qua các quan sát (observations).
Một HMM được định nghĩa bởi bộ năm thành phần $\lambda = (S, O, A, B, \pi)$:

- **Tập trạng thái ẩn** S: $S = \{s_1, s_2, \ldots, s_N\}$ với N là số trạng thái. Trong bài toán của chúng ta, $N = 26$ trạng thái (24 hợp âm + N + Complex).
- **Tập quan sát** O: Không gian các giá trị quan sát có thể. Trong trường hợp này, mỗi quan sát là một vector chroma 12 chiều: $o_t \in \mathbb{R}^{12}$.
- **Ma trận chuyển trạng thái** A: Ma trận $N \times N$ với $A_{ij} = P(s_{t+1} = j \mid s_t = i)$ biểu diễn xác suất chuyển từ trạng thái i sang trạng thái j. Thoả mãn:

$$\sum_{j=1}^{N} A_{ij} = 1, \quad \forall i \in \{1, \ldots, N\} \tag{2.1}$$

- **Mô hình phát xạ** B: Tập hợp các phân phối xác suất $B = \{b_1, b_2, \ldots, b_N\}$, trong đó $b_i(o)$ là xác suất (hoặc mật độ xác suất) quan sát được o khi đang ở trạng thái i. Trong implementation của chúng ta:

$$b_i(o) = P(o_t = o \mid s_t = i) \text{ được mô hình hoá bởi } \text{GMM}_i(o) \tag{2.2}$$

- **Phân phối trạng thái ban đầu** π: Vector N chiều với $\pi_i = P(s_1 = i)$ là xác suất trạng thái ban đầu là i. Thoả mãn:

$$\sum_{i=1}^{N} \pi_i = 1 \tag{2.3}$$

2.1.2 Giả định Markov

HMM dựa trên hai giả định quan trọng:
1. Tính Markov bậc nhất (First-order Markov Property):

$$P(s_t \mid s_1, s_2, \ldots, s_{t-1}) = P(s_t \mid s_{t-1}) \tag{2.4}$$

Trạng thái hiện tại chỉ phụ thuộc vào trạng thái trước đó, không phụ thuộc vào các trạng thái xa hơn trong quá khứ.

Ý nghĩa trong nhận diện hợp âm: Hợp âm hiện tại có xu hướng liên quan mật thiết đến hợp âm ngay trước đó (ví dụ: trong tiến trình I-IV-V-I). Mặc dù giả định này là đơn giản hoá (trong thực tế, cấu trúc âm nhạc có thể phụ thuộc vào ngữ cảnh dài hơn), nhưng nó đủ hiệu quả cho nhiều ứng dụng và giúp giảm độ phức tạp tính toán.

2. Tính độc lập quan sát (Output Independence):

$$P(o_t \mid s_1, \ldots, s_T, o_1, \ldots, o_{t-1}, o_{t+1}, \ldots, o_T) = P(o_t \mid s_t) \tag{2.5}$$

Quan sát tại thời điểm t chỉ phụ thuộc vào trạng thái tại t, độc lập với các quan sát và trạng thái khác.

Ý nghĩa: Vector chroma tại một frame chỉ được sinh ra từ hợp âm đang diễn ra tại frame đó, không chịu ảnh hưởng trực tiếp từ chroma của frame trước/sau.

2.1.3 Xác suất kết hợp

Với một chuỗi quan sát $O = (o_1, o_2, \ldots, o_T)$ và một chuỗi trạng thái ẩn $S = (s_1, s_2, \ldots, s_T)$, xác suất kết hợp được tính như sau:

$$P(O, S \mid \lambda) = \pi_{s_1} \cdot b_{s_1}(o_1) \cdot \prod_{t=2}^{T} A_{s_{t-1}, s_t} \cdot b_{s_t}(o_t) \tag{2.6}$$

Trong miền log (để tránh underflow):

$$\log P(O, S \mid \lambda) = \log \pi_{s_1} + \log b_{s_1}(o_1) + \sum_{t=2}^{T} \left[\log A_{s_{t-1}, s_t} + \log b_{s_t}(o_t) \right] \tag{2.7}$$

2.2 Ba bài toán cơ bản của HMM

2.2.1 Bài toán 1: Đánh giá (Evaluation)

Vấn đề: Cho mô hình λ và chuỗi quan sát O, tính $P(O \mid \lambda)$.

Giải pháp: Thuật toán Forward-Backward (hoặc chỉ Forward).

Ứng dụng: Đánh giá mức độ phù hợp của một chuỗi quan sát với mô hình.

2.2.2 Bài toán 2: Giải mã (Decoding)

Vấn đề: Cho mô hình λ và chuỗi quan sát O, tìm chuỗi trạng thái $S^* = \arg\max_S P(S \mid O, \lambda)$ có xác suất cao nhất.
Giải pháp: Thuật toán Viterbi (sẽ trình bày chi tiết ở phần sau).
Ứng dụng: Đây chính là bài toán chúng ta giải quyết trong nhận diện hợp âm.

2.2.3 Bài toán 3: Học (Learning)

Vấn đề: Cho tập chuỗi quan sát, ước lượng tham số λ sao cho $P(O \mid \lambda)$ đạt cực đại.
Giải pháp: Thuật toán Baum-Welch (Expectation-Maximization cho HMM).
Trong project này: Chúng ta không dùng Baum-Welch vì có sẵn nhãn (supervised learning). Thay vào đó, tham số được ước lượng trực tiếp từ dữ liệu có nhãn:

- $\hat{\pi}_i = \frac{\text{số bài hát bắt đầu bằng hợp âm } i}{\text{tổng số bài hát}}$
- $\hat{A}_{ij} = \frac{\text{số lần chuyển từ } i \text{ sang } j}{\text{số lần xuất hiện trạng thái } i}$
- $\hat{b}_i(o)$: huấn luyện GMM trên tất cả các vector chroma tương ứng với trạng thái i

2.3 Gaussian Mixture Model (GMM) cho mô hình phát xạ

2.3.1 Lý do sử dụng GMM

Vector chroma của cùng một hợp âm có thể có phân phối phức tạp do:

- Nhiều cách voicing khác nhau (inversion, doubling notes)
- Âm sắc nhạc cụ đa dạng
- Nhiễu, harmonics phụ

GMM cho phép mô hình hoá phân phối multimodal (nhiều đỉnh) bằng cách kết hợp nhiều Gaussian thành phần.

2.3.2 Định nghĩa GMM

Một GMM với K thành phần có mật độ xác suất:

$$b_i(o) = \sum_{k=1}^{K} w_{ik} \cdot \mathcal{N}(o \mid \mu_{ik}, \Sigma_{ik}) \tag{2.8}$$

trong đó:

- w_{ik}: trọng số (mixing coefficient) của thành phần k trong GMM của trạng thái i, với $\sum_{k=1}^{K} w_{ik} = 1$
- $\mu_{ik} \in \mathbb{R}^{12}$: vector trung bình của thành phần k
- $\Sigma_{ik} \in \mathbb{R}^{12\times 12}$: ma trận hiệp phương sai (covariance matrix) của thành phần k
- $\mathcal{N}(o \mid \mu, \Sigma)$: phân phối Gaussian đa biến:

$$\mathcal{N}(o \mid \mu, \Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(o-\mu)^T \Sigma^{-1}(o-\mu)\right) \tag{2.9}$$

2.3.3 Huấn luyện GMM

Trong code, mỗi GMM được huấn luyện bằng thuật toán Expectation-Maximization (EM) từ `sklearn.mixture.GaussianMixture`:

```
gmm = GaussianMixture(n_components=40,
                      random_state=0,
                      covariance_type='full')
gmm.fit(observations_for_state)
```

- `n_components=40`: số thành phần Gaussian (có thể tuning)
- `covariance_type='full'`: ma trận hiệp phương sai đầy đủ (không giả định độc lập giữa các chiều)
- Phương thức `gmm.score_samples(o)`: trả về $\log b_i(o)$ cho vector quan sát o

2.4 Thuật toán Viterbi

2.4.1 Bài toán

Cho:

- Mô hình HMM $\lambda = (S, O, A, B, \pi)$
- Chuỗi quan sát $O = (o_1, o_2, \ldots, o_T)$

Tìm chuỗi trạng thái $S^* = (s_1^*, s_2^*, \ldots, s_T^*)$ sao cho:

$$S^* = \arg\max_S P(S \mid O, \lambda) = \arg\max_S P(S, O \mid \lambda) \tag{2.10}$$

2.4.2 Ý tưởng chính

Viterbi sử dụng quy hoạch động (dynamic programming) để tránh liệt kê tất cả N^T chuỗi trạng thái có thể.
Định nghĩa biến trung gian:

$$V_t(j) = \max_{s_1,\ldots,s_{t-1}} P(s_1,\ldots,s_{t-1},s_t = j,o_1,\ldots,o_t \mid \lambda) \tag{2.11}$$

$V_t(j)$ là xác suất cao nhất của đường đi từ thời điểm 1 đến t kết thúc tại trạng thái j và sinh ra các quan sát $o_1,\ldots,o_t$.

2.4.3 Công thức đệ quy

Khởi tạo (t = 1):

$$V_1(j) = \pi_j \cdot b_j(o_1), \quad \forall j \in \{1,\ldots,N\} \tag{2.12}$$

Trong miền log:

$$\log V_1(j) = \log \pi_j + \log b_j(o_1) \tag{2.13}$$

Đệ quy (t = 2, ..., T):

$$V_t(j) = \max_i \left[V_{t-1}(i) \cdot A_{ij}\right] \cdot b_j(o_t) \tag{2.14}$$

Trong miền log:

$$\log V_t(j) = \max_i \left[\log V_{t-1}(i) + \log A_{ij}\right] + \log b_j(o_t) \tag{2.15}$$

Đồng thời, lưu lại con trỏ (backpointer) để truy vết ngược:

$$\mathrm{Ptr}_t(j) = \arg\max_i \left[\log V_{t-1}(i) + \log A_{ij}\right] \tag{2.16}$$

Kết thúc: Trạng thái cuối cùng có xác suất cao nhất:

$$s_T^* = \arg\max_j \log V_T(j) \tag{2.17}$$

Truy vết ngược (Backtracking):

$$s_t^* = \mathrm{Ptr}_{t+1}(s_{t+1}^*), \quad t = T-1, T-2, \ldots, 1 \tag{2.18}$$

2.4.4 Độ phức tạp tính toán

- Thời gian: $O(T \times N^2)$ (với T là số frame, N là số trạng thái)

- Không gian: $O(T \times N)$

So với việc liệt kê tất cả đường đi ($O(N^T)$), Viterbi hiệu quả hơn rất nhiều.

2.4.5 Ví dụ minh hoạ nhỏ

Giả sử:

- 2 trạng thái: C:maj (state 0), G:maj (state 1)
- 3 quan sát: o_1, o_2, o_3
- $\pi = [0.7, 0.3]$
- $A = \begin{bmatrix} 0.8 & 0.2 \\ 0.4 & 0.6 \end{bmatrix}$
- Emission probabilities đã biết: $b_0(o_1), b_1(o_1), \ldots$

Bước 1 (t=1):

$$\log V_1(0) = \log 0.7 + \log b_0(o_1) \tag{2.19}$$
$$\log V_1(1) = \log 0.3 + \log b_1(o_1) \tag{2.20}$$

Bước 2 (t=2):

$$\log V_2(0) = \max \begin{cases} \log V_1(0) + \log 0.8 \\ \log V_1(1) + \log 0.4 \end{cases} + \log b_0(o_2) \tag{2.21}$$

$$\log V_2(1) = \max \begin{cases} \log V_1(0) + \log 0.2 \\ \log V_1(1) + \log 0.6 \end{cases} + \log b_1(o_2) \tag{2.22}$$

(Tương tự cho $t = 3$, sau đó backtrack để tìm chuỗi trạng thái tối ưu)

3 Kết quả và đánh giá

3.1 Cấu hình thực nghiệm

3.1.1 Dataset

- Dữ liệu huấn luyện: Tập các file âm thanh (.mp3/.wav) kèm file annotation (.lab)
- Mỗi file .lab chứa thông tin: `start_time end_time chord_label`
- Sau khi aggregate, dữ liệu được lưu trong file `hmm_data_package.npz` gồm:
  - `X_data`: Ma trận chroma $(M_{total} \times 12)$, với M_{total} là tổng số frame từ tất cả các bài
  - `y_data`: Vector nhãn $(M_{total},)$ chứa state ID
  - `lengths`: Mảng chứa số frame của từng bài, để phân tách các chuỗi độc lập

3.1.2 Tham số mô hình

- Số trạng thái HMM: $N = 26$ (24 hợp âm + N + Complex)
- Sample rate: 22050 Hz
- Hop length: 512 samples ($\approx$ 23.2 ms, $\approx$ 43 frames/giây)
- Số thành phần GMM mỗi trạng thái: 40 (có thể điều chỉnh dựa trên BIC/AIC analysis)
- Covariance type: `full`

3.2 Quy trình huấn luyện

3.2.1 Bước 1: Trích xuất và căn chỉnh

```
all_chroma_vectors, all_label_vectors =
    aggregate_all_data(DATA_DIR, audio_processor)
```

Kết quả: Danh sách các cặp (chroma features, aligned labels) cho mỗi bài hát.

3.2.2 Bước 2: Tính toán tham số HMM

- **Ma trận** A: Đếm số lần chuyển trạng thái, thêm smoothing 10^{-5}, chuẩn hoá
- **Vector** π: Đếm trạng thái đầu tiên của mỗi bài, chuẩn hoá
- **GMM cho mỗi trạng thái**: Tập hợp tất cả chroma vector thuộc trạng thái đó, huấn luyện GMM

Output:

```
INITIAL_STATE_DISTRIBUTION shape: (26,)
TRANSITION_MATRIX shape: (26, 26)
EMISSION_MODELS: 26 GMM models
```

3.3 Quy trình dự đoán

Với một file âm thanh mới (không có nhãn):

3.3.1 Bước 1: Trích xuất chroma

```
features = audio_processor.extract_chroma(audio_file)
```

Output: Ma trận chroma $(M \times 12)$ với M là số frame.

3.3.2 Bước 2: Áp dụng Viterbi

```
predicted_state_ids = viterbi(features,
                              INITIAL_STATE_DISTRIBUTION,
                              TRANSITION_MATRIX,
                              EMISSION_MODELS)
```

Output: Mảng state ID $(M,)$.

3.3.3 Bước 3: Chuyển đổi về nhãn hợp âm

```
predicted_chords = [ID_TO_CHORD[state_id]
                    for state_id in predicted_state_ids]
```

Output: Danh sách nhãn hợp âm (ví dụ: `['C:maj', 'C:maj', 'F:maj', ...]`).

3.3.4 Bước 4: Lưu kết quả

Mỗi frame tương ứng một nhãn, được ghi ra file `.txt`:

```
C:maj
C:maj
F:maj
G:maj
...
```

Nếu muốn chuyển về dạng khoảng thời gian (segment), cần gom các frame liên tiếp có cùng nhãn và tính thời gian bằng công thức:

$$t_i = \frac{i \times \text{hop\_length}}{\text{sample\_rate}} \tag{3.1}$$

3.4 Phân tích kết quả

3.4.1 Thống kê hợp âm dự đoán

Sau khi merge tất cả file dự đoán, notebook thực hiện phân tích đơn giản:

- Đếm tần suất xuất hiện của mỗi hợp âm
- Hiển thị top 10 hợp âm phổ biến nhất
- Tổng số frame được phân loại

Nhận xét chung:

- Các hợp âm phổ biến (C:maj, G:maj, F:maj, A:min, D:min) thường xuất hiện nhiều hơn
- Trạng thái "N"(no-chord) có thể chiếm tỷ lệ đáng kể nếu có nhiều đoạn im lặng hoặc intro/outro
- Trạng thái "Complex"xuất hiện ít hơn, phản ánh việc đơn giản hoá không gian trạng thái

3.4.2 Ưu điểm của phương pháp

- **Tận dụng thông tin chuỗi**: HMM mô hình hoá tiến trình hợp âm tự nhiên (ví dụ: I-IV-V-I), giúp dự đoán mượt mà hơn so với phân loại độc lập từng frame.
- **Xử lý nhiễu tốt**: GMM với nhiều thành phần có thể bắt được sự đa dạng của cùng một hợp âm trên nhiều nhạc cụ, phong cách khác nhau.

- **Hiệu quả tính toán**: Viterbi với $O(T \times N^2)$ khả thi cho real-time hoặc near real-time với $N = 26$ và T vài nghìn frame.
- **Supervised learning đơn giản**: Không cần thuật toán phức tạp như Baum-Welch, chỉ cần đếm tần suất từ dữ liệu có nhãn.

3.4.3 Hạn chế và hướng cải thiện

- **Không gian trạng thái rút gọn**: 26 trạng thái không thể biểu diễn các hợp âm phức tạp (7th, 9th, sus4, dim, aug). Có thể mở rộng thêm trạng thái nhưng cần dữ liệu lớn hơn.
- **Giả định Markov bậc 1**: Không bắt được cấu trúc dài hạn (ví dụ: verse-chorus pattern). Có thể sử dụng higher-order HMM hoặc kết hợp với mô hình cấu trúc âm nhạc.
- **Phụ thuộc vào chất lượng annotation**: Nếu file .lab có lỗi hoặc không đồng nhất, tham số HMM sẽ bị ảnh hưởng.
- **Tuning GMM components**: Số thành phần 40 là một lựa chọn ban đầu. Có thể tối ưu hoá bằng cross-validation hoặc grid search với BIC/AIC (code có phần comment về điều này).
- **Đánh giá định lượng**: Hiện tại chưa có metric như accuracy, precision, recall trên test set. Nên tách train/test và tính các chỉ số đánh giá.

3.5 Đề xuất cải tiến

1. **Mở rộng tập trạng thái**: Thêm các hợp âm 7th, suspended, diminished phổ biến (ví dụ: C:maj7, G:7, D:sus4).
2. **Duration modeling**: HMM chuẩn không mô hình hoá độ dài trạng thái (duration). Có thể dùng Hidden Semi-Markov Model (HSMM) để mô hình hoá rõ ràng thời gian tồn tại của mỗi hợp âm.
3. **Context features**: Thêm các đặc trưng ngoài chroma (ví dụ: MFCC, spectral contrast) để cải thiện khả năng phân biệt.
4. **Deep learning hybrid**: Kết hợp với mạng neural (ví dụ: CNN hoặc RNN) để học biểu diễn tốt hơn, sau đó dùng HMM để làm mịn chuỗi.
5. **Ensemble methods**: Kết hợp nhiều mô hình (HMM với tham số khác nhau, hoặc HMM + classification) để tăng độ tin cậy.

4 Kết luận

4.1 Tóm tắt công việc

Báo cáo này trình bày một hệ thống hoàn chỉnh cho bài toán nhận diện hợp âm từ tín hiệu âm thanh sử dụng Hidden Markov Model và thuật toán Viterbi. Các đóng góp chính bao gồm:

1. **Pipeline xử lý tín hiệu số**: Từ tín hiệu âm thanh thô đến đặc trưng chroma được chuẩn hoá, với lựa chọn tham số DSP hợp lý (sample rate 22050 Hz, hop length 512).
2. **Mô hình hoá HMM supervised**: Ước lượng tham số HMM từ dữ liệu có nhãn, sử dụng GMM với 40 thành phần để mô hình hoá emission probabilities.
3. **Giải mã Viterbi hiệu quả**: Implementation thuật toán Viterbi trong miền log-probability để tránh underflow và tìm chuỗi hợp âm có xác suất cao nhất.
4. **Thiết kế module tái sử dụng**: Lớp `AudioProcessor` đóng gói các chức năng trích xuất và căn chỉnh, dễ dàng áp dụng cho file mới.

4.2 Ý nghĩa và ứng dụng

Hệ thống này có thể được ứng dụng trong:

- **Phân tích âm nhạc tự động**: Hỗ trợ nhạc sĩ, nhà nghiên cứu âm nhạc học phân tích cấu trúc hợp âm của bản nhạc.
- **Hệ thống học nhạc**: Tự động tạo chord chart cho người học guitar, piano.
- **Tạo nhạc tự động (music generation)**: Cung cấp thông tin hợp âm làm điều kiện cho mô hình sinh nhạc.
- **Music Information Retrieval (MIR)**: Làm nền tảng cho các tác vụ như tìm kiếm bài hát theo progression, phân loại thể loại dựa trên harmony.

4.3 Bài học kinh nghiệm

- **Tầm quan trọng của tham số DSP**: Lựa chọn sample rate và hop length ảnh hưởng trực tiếp đến chất lượng đặc trưng và hiệu suất tính toán. Cần cân bằng giữa độ phân giải và chi phí.

- **Trade-off giữa độ phức tạp mô hình và dữ liệu**: Không gian trạng thái lớn hơn (nhiều loại hợp âm hơn) yêu cầu dữ liệu huấn luyện nhiều hơn để tránh overfitting.
- **Smoothing quan trọng**: Thêm giá trị smoothing nhỏ (10^{-5}) vào ma trận chuyển trạng thái giúp tránh xác suất bằng 0, cải thiện độ ổn định của mô hình.
- **GMM hiệu quả cho emission modeling**: So với single Gaussian, GMM linh hoạt hơn trong việc mô hình hoá phân phối phức tạp của chroma features.

4.4 Hướng phát triển tương lai

Các hướng nghiên cứu tiếp theo có thể bao gồm:

1. **Kết hợp deep learning**: Sử dụng Convolutional Neural Network (CNN) hoặc Recurrent Neural Network (RNN) để học biểu diễn từ spectrogram, sau đó dùng HMM hoặc CRF (Conditional Random Field) để làm mịn chuỗi.
2. **Transfer learning**: Pre-train mô hình trên dataset lớn (ví dụ: McGill Billboard, Isophonics), sau đó fine-tune cho domain cụ thể.
3. **Multi-resolution analysis**: Kết hợp thông tin từ nhiều hop length khác nhau để bắt cả biến đổi nhanh và xu hướng dài hạn.
4. **Đánh giá định lượng chặt chẽ**: Thiết lập protocol đánh giá với train/validation/test split, các metric chuẩn (frame-level accuracy, segment-level precision/recall), so sánh với baseline.
5. **Real-time implementation**: Tối ưu hoá code (ví dụ: dùng Cython, numba) để xử lý streaming audio với độ trễ thấp.

4.5 Lời kết

Báo cáo này đã trình bày chi tiết một cách tiếp cận cổ điển nhưng hiệu quả cho bài toán nhận diện hợp âm, từ lý thuyết xử lý tín hiệu số, mô hình Markov ẩn, đến implementation cụ thể với Python và các thư viện mã nguồn mở. Mặc dù có những hạn chế, phương pháp HMM/GMM vẫn là nền tảng quan trọng, giúp hiểu sâu về bản chất của bài toán và làm tiền đề cho các phương pháp hiện đại hơn. Hy vọng tài liệu này hữu ích cho người đọc quan tâm đến lĩnh vực Music Information Retrieval và xử lý tín hiệu âm thanh.