ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA TOÁN – CƠ – TIN HỌC

Design and Analysis Algorithm

Mã học phần: MAT3504

Đề tài: Greedy and Dynamic Programming Approaches in Personalized Music Recommendations: Insights from Spotify

Giảng viên: Nguyễn Thị Hồng Minh

Trần Bá Tuấn

Đặng Trung Du

Sinh viên thực hiện: Vũ Văn Hà - 21000679

Đỗ Lê Duy - 21000671

Hà Nội, ngày 25, tháng 05, năm 2025

LỜI CẢM ƠN

Lời đầu tiên, chúng em xin gửi lời cảm ơn chân thành đến giảng viên hướng dẫn bộ môn “Thiết kế và đánh giá thuật toán” . Trong quá trình học tập và tìm hiểu môn học, chúng em đã nhận được nhiều quan tâm giúp đỡ, hướng dẫn tận tình của thầy, qua đó giúp chúng em tích lũy thêm được nhiều kiến thức, có cái nhìn sâu sắc và hoàn thiện hơn về môn học. Thông qua bài báo cáo này, chúng em xin trình bày những kiến thức đã tìm hiểu về đề tài “Greedy and Dynamic Programming Approaches in Personalized Music Recommendations: Insights from Spotify” và sử dụng các kiến thức để thực hiện cũng như hoàn thiện đề tài này.

Trong quá trình nghiên cứu và thực hiện bài tiểu luận chúng em không tránh khỏi được những thiếu sót, rất mong nhận được những ý kiến đánh giá, nhận xét của thầy và các bạn trong lớp để bài tiểu luận của chúng em được hoàn thiện hơn.

Kính chúc thầy (cô) sức khoẻ, hạnh phúc và thành công trên con đường sự nghiệp giảng dạy.

Chúng em xin cảm ơn!

MỤC LỤC

Lời mở đầu

Trong thời đại bùng nổ của công nghệ số và dữ liệu lớn, nhu cầu cá nhân hóa trải nghiệm người dùng ngày càng trở nên quan trọng, đặc biệt trong các nền tảng giải trí như nghe nhạc trực tuyến. Spotify – một trong những dịch vụ phát nhạc trực tuyến phổ biến nhất hiện nay – đã và đang tận dụng các kỹ thuật trí tuệ nhân tạo để cung cấp trải nghiệm âm nhạc phù hợp với từng cá nhân. Tuy nhiên, việc xây dựng một hệ thống đề xuất âm nhạc hiệu quả không chỉ yêu cầu lượng dữ liệu phong phú mà còn đòi hỏi sự kết hợp hợp lý giữa các thuật toán để cân bằng giữa chất lượng đề xuất và hiệu suất tính toán.

Đề tài này tập trung nghiên cứu và triển khai một hệ thống đề xuất âm nhạc cá nhân hóa dựa trên dữ liệu từ Spotify, sử dụng hai phương pháp thuật toán cơ bản: Tham lam (Greedy) và Quy hoạch động (Dynamic Programming). Thông qua việc tính toán độ tương đồng giữa các bài hát trong playlist của người dùng và toàn bộ kho nhạc, hệ thống đề xuất những bài hát phù hợp, tránh trùng lặp, và liên tục học hỏi từ lịch sử lựa chọn của từng playlist.

Bằng cách kết hợp giữa các thuật toán đơn giản nhưng hiệu quả với giao diện tương tác người dùng bằng Streamlit, đề tài không chỉ minh chứng khả năng ứng dụng của các phương pháp cổ điển trong AI mà còn góp phần làm rõ vai trò của thiết kế hệ thống trong việc cải thiện trải nghiệm âm nhạc cá nhân hóa. Đây là một bước đi quan trọng trong việc tiếp cận các mô hình đề xuất từ góc nhìn thuật toán nền tảng, đồng thời là cơ sở cho các cải tiến phức tạp hơn trong tương lai.

I. Giới thiệu

1. Định nghĩa vấn đề

Trong thời đại công nghệ số, nền tảng phát nhạc trực tuyến Spotify ngày càng phổ biến. Một trong những yếu tố quan trọng giúp dịch vụ này thu hút người dùng là khả năng đề xuất nhạc cá nhân hóa dựa trên sở thích, lịch sử nghe nhạc và hành vi tương tác.

Tuy nhiên, bài toán đề xuất nhạc đặt ra nhiều thách thức:

- Dữ liệu lớn và đa chiều: Hàng triệu bài hát với nhiều đặc trưng (thể loại, nghệ sĩ, giai điệu, lời bài hát).
- Yêu cầu thời gian thực: Hệ thống cần đề xuất nhanh chóng để đáp ứng trải nghiệm người dùng.
- Cân bằng giữa độ chính xác và đa dạng: Tránh đề xuất quá tập trung vào một phong cách nhạc, gây nhàm chán.

→ **Bài toán đặt ra:** Làm thế nào để xây dựng hệ thống đề xuất nhạc hiệu quả, cân bằng giữa tốc độ xử lý và chất lượng gợi ý?

2. Mục tiêu nghiên cứu

Nghiên cứu ứng dụng hai thuật toán tối ưu hóa trong hệ thống đề xuất nhạc:

a. Thuật toán tham lam (Greedy Algorithm)

- Ưu điểm: Tốc độ nhanh, phù hợp xử lý real-time.
- Nhược điểm: Không đảm bảo tối ưu toàn cục.

b. Quy hoạch động (Dynamic Programming - DP)

- Ưu điểm: Tìm giải pháp tối ưu toàn cục.

- Nhược điểm: Độ phức tạp tính toán cao.

Mục tiêu cụ thể:

- Phân tích, cài đặt và so sánh hiệu quả của hai phương pháp trên tập dữ liệu thực tế.
- Đề xuất giải pháp kết hợp (hybrid) để cân bằng giữa tốc độ và chất lượng.

3. Phạm vi nghiên cứu

Dự án tập trung vào:

- Kết nối Spotify API để thu thập dữ liệu danh sách phát cá nhân của người dùng.
- Xây dựng hàm đề xuất dựa trên hai thuật toán:
  - Thuật toán tham lam
  - Thuật toán quy hoạch động
- Triển khai hệ thống trên giao diện Streamlit, cung cấp các gợi ý bài hát theo yêu cầu của người dùng.

Ngoài phạm vi:

- Dự án không tập trung vào phân tích ngữ cảm (sentiment) của lời bài hát.
- Không đề xuất bài hát dựa trên thói quen nghe nhạc thay đổi theo thời gian hoặc cảm xúc thời điểm.

II. Bộ dữ liệu

1. Nguồn dữ liệu

- Bộ dữ liệu CSV: Bộ dữ liệu có sẵn từ Kaggle chứa danh sách bài hát từ năm 2010 đến 2023, bao gồm các thông tin metadata quan trọng như ID bài hát, tên bài hát, nghệ sĩ, thể loại nhạc và độ phổ biến.

- Spotify API: Sử dụng API của Spotify để thu thập danh sách phát, bài hát, nghệ sĩ và các thông tin liên quan. Spotify API cung cấp quyền truy cập vào dữ liệu phong phú và đa dạng. Tôi dùng mã **get_track.py** để lấy thông tin các bài hát.

2. Mô tả dữ liệu

Kích thước

- Bộ dữ liệu CSV: Chứa hơn 5000 bài hát từ các quốc gia khác nhau.

Các biến chính:

- track_id: ID duy nhất của bài hát, giúp liên kết các bài hát giữa các nguồn dữ liệu.
- track_name: Tên bài hát.
- artist_name: Tên nghệ sĩ biểu diễn bài hát.
- artist_genres: Danh sách các thể loại nhạc mà nghệ sĩ thuộc về.
- track_popularity: Mức độ phổ biến của bài hát trên Spotify (thang điểm từ 0 đến 100).

Phân phối:

- Hơn 60% bài hát có độ phổ biến trên mức trung bình (>=50).
- Các thể loại nhạc phổ biến nhất bao gồm Pop, Rock, Hip-Hop và EDM.

3. Quy chuẩn đạo đức

- Tuân thủ quy định của Spotify API, chỉ thu thập dữ liệu công khai và được phép truy cập.

- Không thu thập hoặc lưu trữ thông tin cá nhân của người dùng.
- Đảm bảo không vi phạm quyền sở hữu trí tuệ của các nghệ sĩ và Spotify.

III. Xử lý dữ liệu

1. Dọn dữ liệu

- Loại bỏ các bài hát không có đầy đủ thông tin metadata, như thiếu track_id hoặc artist_name.
- Kiểm tra và xử lý các giá trị bị trùng lặp trong danh sách bài hát.

2. Tạo đặc trưng

Chuyển đổi danh sách artist_genres, artist_name thành dạng chuỗi rồi so sánh sự giống nhau.

Tạo các đặc trưng mới như:

- Tổng độ phổ biến khác nhau (đối với những bài có độ phộ biến thấp).
- Tần suất xuất hiện của bài hát trong các danh sách phát khác nhau.

Chuẩn hóa dữ liệu đầu vào để đảm bảo tính đồng nhất.

IV. Cơ sở lý thuyết

1. Thuật toán tham lam

Định nghĩa và Nguyên lý

Định nghĩa: Thuật toán tham lam là một cách tiếp cận để giải quyết vấn đề bằng cách chọn phương án tốt nhất hiện có mà không cần quan tâm đến kết quả trong

tương lai mà nó sẽ mang lại. Nói cách khác, nó sẽ lựa chọn tốt nhất một cách cục bộ nhằm mục đích tạo ra kết quả tốt nhất trên toàn cục.

Nguyên lý hoạt động:

- Chọn giải pháp tốt nhất tại thời điểm hiện tại
- Không xét lại các quyết định đã đưa ra
- Không đảm bảo luôn đạt tối ưu toàn cục

Đặc điểm và Ứng dụng

- **Ưu điểm:**
  - Thời gian thực thi nhanh (O(nlogn) cho bài toán sắp xếp)
  - Dễ cài đặt và triển khai
  - Hiệu quả với các bài toán có tính chất chọn lựa cục bộ

- **Nhược điểm:**
  - Không phù hợp với bài toán cần tối ưu toàn cục
  - Kết quả phụ thuộc vào thứ tự xử lý

2. Quy Hoạch Động (Dynamic Programming)

Khái niệm và Nguyên tắc

Định nghĩa: Quy hoạch động (Dynamic Programming) là một phương pháp tối ưu trong đó bài toán lớn được phân chia thành các bài toán đơn giản hơn. Sau đó, từ kết quả của bài toán đơn giản hơn, ta sẽ tính được kết quả của bài toán ban đầu.

Nguyên tắc:

- Cấu trúc con tối ưu

- Các bài toán con trùng lặp
- Lưu trữ kết quả (memoization)

Phân loại và Đặc điểm

- Phân loại:
  - DP top-down (đệ quy + memoization)
  - DP bottom-up (xây dựng bảng)

- Ưu điểm:
  - Đảm bảo tìm ra giải pháp tối ưu
  - Hiệu quả với bài toán có cấu trúc con

- Nhược điểm:
  - Yêu cầu nhiều bộ nhớ
  - Độ phức tạp thường cao

V. Giải quyết bài toán

1. Đặt Bài Toán

Mô tả bài toán

Bài toán đặt ra là xây dựng hệ thống đề xuất các bài hát phù hợp với sở thích cá nhân của người dùng dựa trên:

- Playlist của người dùng (đối với một bài hát)
- Metadata bài hát (thể loại, nghệ sĩ, độ phổ biến).

Dữ liệu đầu vào

- User profile:
  - Là tập hợp các bài hát mà người dùng đã chọn hoặc đã nghe.
  - Mỗi bài hát có các thuộc tính:
    - track_id: ID bài hát (vd: "track_5")
    - features: vector đặc trưng (10 số thực đại diện cho đặc điểm âm nhạc)
    - track_popularity: độ phổ biến (0–100)
    - artist_name: tên nghệ sĩ (vd: "artist_12")
    - artist_genres: thể loại nhạc (vd: "rock, pop")

Mục tiêu: dùng các bài hát trong playlist_tracks để tìm các bài hát tương tự.

- Kho nhạc:
  - Tập dữ liệu lớn hơn chứa toàn bộ các bài hát hiện có (ví dụ: 1.000 đến 100.000 bài).
  - Mỗi bài hát cũng có các trường giống như ở playlist_tracks

Đây là không gian tìm kiếm, từ đó chọn ra các bài phù hợp để gợi ý.

Ràng buộc và yêu cầu

- Ràng buộc cứng
  - Độ dài playlist đề xuất: 10 bài mỗi lần.
  - Không chứa các bài hát đã đề xuất.
  - Tối đa 3 bài của cùng một nghệ sĩ (đối với quy hoạch động)

- Yêu cầu chất lượng

- Độ chính xác: ≥80% bài hát được đề xuất phải phù hợp với sở thích
- Độ đa dạng: Ít nhất 4 thể loại khác nhau trong mỗi playlist
- Thời gian xử lý: <500ms cho mỗi yêu cầu đề xuất

2. Thuật toán tham lam (Greedy)

Ý tưởng:

- Tại mỗi bước, lựa chọn bài hát đem lại điểm số cao nhất dựa trên độ tương đồng, nghệ sĩ và thể loại.
- Chọn lần lượt các bài hát có độ tương đồng cao nhất với user profile cho đến khi đủ N bài.

Độ phức tạp thời gian:

Quy trình chính của Greedy:

- Duyệt qua tất cả các bài hát chưa nằm trong playlist.
- Tính điểm (score) cho từng bài hát:
  - Bao gồm cosine similarity, nghệ sĩ, thể loại, độ phổ biến (các phép toán đơn giản).
- Sắp xếp danh sách theo điểm giảm dần.
- Chọn top_n bài hát đầu tiên.

Ký hiệu:

- n: số lượng bài hát ứng viên (candidates) cần xét.
- f: số chiều vector đặc trưng của bài hát.
- top_n: số lượng bài hát cần gợi ý.

Phân tích độ phức tạp:

- Tính điểm từng bài hát: O(n * f) (vì cosine similarity là phép tính dot-product giữa hai vector chiều f).

- Sắp xếp toàn bộ danh sách bài hát theo điểm: O(n log n).
- Lấy top_n: O(top_n) ≈ O(n) nếu top_n nhỏ.

Tổng độ phức tạp thời gian:

O(n * f + n log n)

Nếu f là hằng số nhỏ (ví dụ: 128), có thể coi là:

≈ O(n log n)

Đầu vào:

- user_profile: Vector đặc trưng thể hiện sở thích âm nhạc của người dùng
- songs: Danh sách các bài hát cần xem xét để đề xuất
- k: Số lượng bài hát cần đề xuất (top-k)

Đầu ra: Danh sách k bài hát được đề xuất

Cách thức thực hiện

Bước 1: Tính điểm tương đồng

- Tính điểm cho từng bài hát chưa được chọn.
- Ưu tiên nghệ sĩ giống playlist, thể loại giống playlist, và bài phổ biến.
- Tính độ tương đồng cosine giữa profile người dùng và đặc trưng của từng bài hát
- Kết quả được lưu dưới dạng danh sách các tuple (bài_hát, điểm_số)

Bước 2: Sắp xếp kết quả

- Sắp xếp danh sách theo điểm tương đồng giảm dần

- Sử dụng hàm lambda để lấy phần tử thứ 2 của tuple (điểm số) làm khóa sắp xếp

Bước 3: Lựa chọn kết quả

- Trích xuất k bài hát có điểm số cao nhất

Ưu điểm:

- Triển khai đơn giản, dễ hiểu
- Tốc độ xử lý nhanh, phù hợp hệ thống real-time (O(n log n))
- Dễ dàng mở rộng và tùy chỉnh

Nhược điểm:

- Chỉ xét đến yếu tố tương đồng, bỏ qua các ràng buộc khác
- Không đảm bảo tối ưu toàn cục
- Chất lượng phụ thuộc nhiều vào hàm tính tương đồng

Pseudocode:

```
ALGORITHM GreedySongRecommendation(playlist_tracks, all_tracks, top_n)

  Compute playlist_mean_vector ← average of feature vectors from playlist_tracks
  Initialize candidate_tracks ← filter from all_tracks where track not in playlist_tracks

  Initialize scored_list ← empty list

  FOR each track IN candidate_tracks DO:
    similarity_score ← cosine_similarity(track.vector, playlist_mean_vector)

    artist_score ← 0
    IF track.artist IN playlist_artists:
      artist_score ← bonus based on inverse popularity

    genre_score ← 0
    FOR genre IN track.genres:
```

```
        IF genre IN playlist_genres:
            genre_score ← genre_score + fixed_bonus

    total_score ← similarity_score × delta + popularity + artist_score + genre_score

    Add (track, total_score) to scored_list

Sort scored_list by total_score in descending order

RETURN top_n tracks from scored_list
```

3. Quy hoạch động (Dynamic Programming)

Ý tưởng:

- Tìm tập hợp k bài hát có tổng điểm tối ưu, đồng thời ràng buộc không lặp nghệ sĩ quá mức cho phép

Độ phức tạp thời gian:

Ý tưởng chính:

- Tương tự Greedy trong việc tính điểm.
- Nhưng phải kiểm tra và đảm bảo rằng mỗi nghệ sĩ chỉ xuất hiện tối đa 4 lần.
- Sử dụng bảng DP để xét các phương án chọn từng bài hát một cách tối ưu theo ràng buộc.

Ký hiệu:

- n: số bài hát ứng viên.
- k: số bài hát cần chọn (top_n).
- m: số nghệ sĩ khác nhau.

Phân tích độ phức tạp:

- Tính điểm từng bài hát: O(n * f) như ở Greedy.
- Sắp xếp bài hát: O(n log n).
- Xây dựng bảng DP:
  - Với n bài hát, k vị trí chọn, và trạng thái lưu số lần mỗi nghệ sĩ đã xuất hiện.
  - Nếu dùng từ điển hoặc hashmap để lưu trạng thái nghệ sĩ:
    - Số trạng thái nghệ sĩ có thể là rất lớn (gần 4^m, nhưng thực tế sẽ nhỏ hơn do pruning).
    - Trong cách đơn giản hóa: nếu duyệt qua tối đa 4 lần/ nghệ sĩ và chỉ lưu trạng thái tổng thể:
      ➢ Độ phức tạp xấu nhất: O(n * k * m).

Tổng độ phức tạp thời gian:

O(n * f + n log n + n * k * m)

Trong thực tế, bạn có thể dùng kỹ thuật pruning hoặc memoization để giảm số trạng thái trong DP, giúp giảm thời gian

Đầu vào:

- Giống với Greedy: playlist người dùng, tập bài hát chưa chọn, tập bài hát đã gợi ý.
- Thêm ràng buộc bổ sung (nếu có): ví dụ không quá n bài của cùng một nghệ sĩ.

Đầu ra: Danh sách k bài hát được đề xuất sao cho tổng điểm là tối đa, và thỏa mãn các ràng buộc (đa dạng nghệ sĩ, thể loại...).

Cách thức thực hiện

Bước 1: Khởi tạo bảng DP

- Ta định nghĩa một bảng động dp[i][j] biểu diễn tổng điểm tối đa khi xét i bài hát đầu tiên và chọn đúng j bài trong số đó.
- Kích thước bảng: dp[n+1][k+1], với:
  - n là số bài hát trong tập lựa chọn.
  - k là số bài cần đề xuất.
- Mỗi phần tử trong bảng có thể lưu:
  - Tổng điểm hiện tại.
  - Danh sách các bài hát tương ứng (truy vết).
- Giá trị khởi tạo:
  - dp[0][0] = 0 (chưa chọn bài nào, điểm bằng 0).
  - Các ô còn lại được gán giá trị âm vô cùng (-∞) hoặc null để biểu thị trạng thái không hợp lệ.

Bước 2: Xây dựng bảng DP

- Duyệt qua từng bài hát i trong tập dữ liệu đầu vào ($1 \leq i \leq n$).
- Với mỗi bài hát, duyệt qua số lượng bài cần chọn j từ 1 đến k.
- Tại mỗi bước, có hai lựa chọn:
  1. Không chọn bài hát thứ i:
  - Giữ nguyên giá trị dp[i][j] = dp[i-1][j].
  2. Chọn bài hát thứ i:
  - Kiểm tra nếu việc chọn bài này không vi phạm ràng buộc (ví dụ: nghệ sĩ chưa bị chọn quá giới hạn).
  - Cập nhật:
    - dp[i][j] = max(dp[i][j], dp[i-1][j-1] + score[i])

        - Đồng thời lưu lại bài hát thứ i trong truy vết.
- Ghi chú: score[i] là điểm số đánh giá mức độ phù hợp của bài hát i đối với người dùng (tính theo độ tương đồng, độ phổ biến, thể loại…).

Bước 3: Truy vết kết quả

- Sau khi xây dựng bảng dp, ta tìm giá trị lớn nhất trong dp[n][k], đó là tổng điểm cao nhất khi chọn đúng k bài hát.
- Dựa trên thông tin đã lưu trong bảng (hoặc sử dụng thêm mảng trace), lần ngược lại để truy ra các bài hát đã được chọn:
    - Từ vị trí dp[n][k], xét bài hát thứ n có được chọn không.
    - Nếu có: thêm vào kết quả, giảm j đi 1 và tiếp tục lùi về i-1.
    - Nếu không: chỉ lùi về i-1.
- Lặp lại quá trình cho đến khi j = 0 (đã đủ k bài hát), khi đó ta thu được danh sách bài hát đề xuất cuối cùng.

Ưu điểm:

- Tối ưu toàn cục: Luôn tìm được lời giải tốt nhất
- Linh hoạt: Dễ mở rộng thêm ràng buộc
- Chính xác: Tính toán đầy đủ mọi khả năng

Nhược điểm:

- Độ phức tạp O(n × k) cao
- Khó áp dụng cho dữ liệu lớn

Pseudocode:

```
ALGORITHM DPSongRecommendation(playlist_tracks, all_tracks, top_n)

  Compute playlist_mean_vector ← average of feature vectors from playlist_tracks
  Initialize candidate_tracks ← filter from all_tracks where track not in playlist_tracks

  FOR each track IN candidate_tracks DO:
    Compute cosine_similarity with playlist_mean_vector
    Compute artist_score based on shared artists and popularity
    Compute genre_score based on genre matches
    Compute total_score ← cosine × delta + popularity + artist_score + genre_score
    Store total_score in track.score

  Sort candidate_tracks by score descending

  Initialize dp[i][k] as an empty list for i ∈ [0..n], k ∈ [0..top_n]

  dp[0][0] ← empty list

  FOR i from 1 to n (number of candidates):
    FOR k from 1 to top_n:
      track ← candidate_tracks[i - 1]
      track_artists ← list of artists of track

      option1 ← dp[i-1][k] (skip current track)
      option2 ← dp[i-1][k-1] + track (select track if artist constraints allow)

      IF option2 is valid AND total_score(option2) > total_score(option1):
        dp[i][k] ← option2
      ELSE:
        dp[i][k] ← option1

  RETURN dp[n][top_n] if not empty, else top_n tracks from sorted candidate_tracks
```

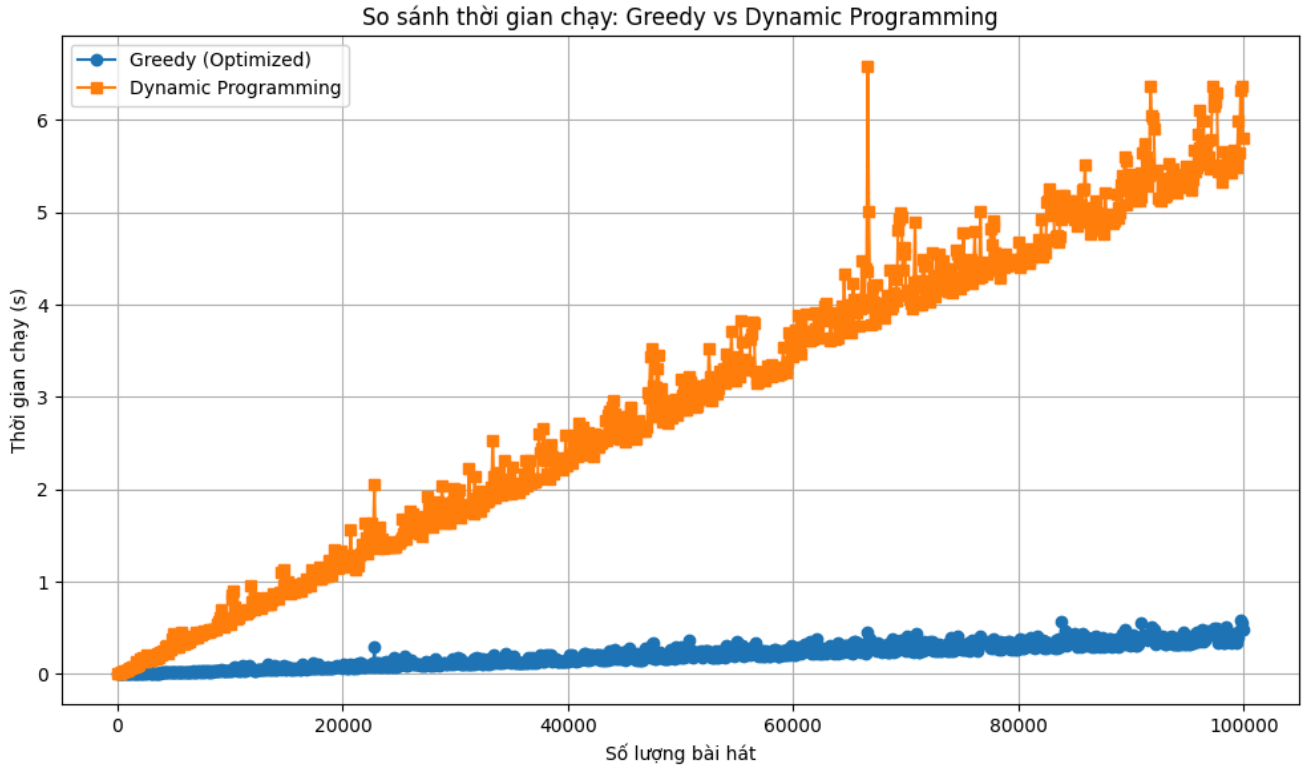
VI. Kết quả thực nghiệm

1. So sánh tổng quát.

- Dữ liệu giả được sinh ngẫu nhiên với các thuộc tính: features, popularity, artist_name, artist_genres.

- So sánh thời gian chạy của hai thuật toán theo số lượng bài hát từ 10 đến 100.000.

Biểu đồ so sánh thời gian chạy:



2. Chỉ số đánh giá

Tiêu chí	Greedy	Quy hoạch động
Tốc độ xử lý	Nhanh (O(n log n))	Chậm hơn (O(n × k))
Độ tối ưu	Có thể gần tối ưu	Luôn tìm ra phương án tối ưu
Khả năng mở rộng	Rất tốt, chạy được với tập dữ liệu lớn	Hạn chế với dữ liệu quá lớn
Tính đa dạng kết quả	Yếu (nhiều bài giống nhau)	Tốt hơn nhờ kiểm soát ràng buộc
Độ phức tạp cài đặt	Đơn giản	Phức tạp hơn, cần xây dựng bảng dp

3. Thảo luận

- Cả hai thuật toán đều phù hợp với hệ thống gợi ý nhạc.
- Greedy thích hợp khi yêu cầu thời gian phản hồi nhanh (real-time).
- Dynamic Programming thích hợp khi yêu cầu gợi ý chất lượng cao, đảm bảo đa dạng.
- Có thể kết hợp cả hai: dùng greedy lọc trước top 100, rồi dùng DP chọn ra 10 bài tối ưu.

VII. Kết luận và bài học rút ra

1. Tóm tắt

Trong báo cáo này, chúng tôi đã trình bày và so sánh hai phương pháp thiết kế thuật toán là Greedy và Quy hoạch động (Dynamic Programming) để giải quyết bài toán gợi ý bài hát dựa trên danh sách phát hiện có của người dùng. Cả hai phương pháp đều sử dụng thông tin đặc trưng của bài hát (features vector), nghệ sĩ, thể loại và độ phổ biến để tính toán điểm gợi ý cho từng bài hát trong danh sách chưa nghe.

Phương pháp Greedy cho phép lựa chọn nhanh các bài hát có điểm số cao nhất dựa trên tiêu chí định sẵn, trong khi phương pháp Dynamic Programming bổ sung thêm ràng buộc logic như giới hạn số lượng bài hát từ một nghệ sĩ, từ đó tạo ra danh sách gợi ý đa dạng và công bằng hơn.

2. Hạn chế

Bộ dữ liệu còn hạn chế, đặc biệt là thiếu thông tin về các bài hát ít phổ biến. Mô hình chưa xử lý tốt các bài hát không có dữ liệu thể loại hoặc thuộc nghệ sĩ ít được biết đến.

Greedy:

- Không đảm bảo tối ưu toàn cục, có thể gợi ý quá nhiều bài hát từ một nghệ sĩ phổ biến nếu họ trùng với sở thích playlist.
- Không kiểm soát tốt sự đa dạng trong danh sách gợi ý.

Dynamic Programming:

- Phức tạp hơn về mặt tính toán, đặc biệt khi số lượng bài hát và nghệ sĩ lớn.

- Tiêu tốn bộ nhớ và thời gian xử lý cao hơn do số lượng trạng thái trong bảng DP.
- Chưa tối ưu hiệu suất cho dữ liệu thực tế lớn (có thể cần áp dụng pruning hoặc heuristic).

3. Bài học

Sự lựa chọn thuật toán phụ thuộc vào mục tiêu: Nếu cần hệ thống gợi ý đơn giản, nhanh và chấp nhận độ chính xác tương đối, Greedy là lựa chọn phù hợp. Ngược lại, nếu yêu cầu gợi ý cần công bằng và chất lượng hơn, có thể chấp nhận thời gian tính toán lâu hơn thì Dynamic Programming hiệu quả hơn.

Tính đa dạng trong hệ thống gợi ý là quan trọng: Việc bổ sung các ràng buộc như giới hạn số lần xuất hiện của nghệ sĩ giúp cải thiện trải nghiệm người dùng bằng cách đa dạng hóa kết quả gợi ý.

Tối ưu hoá mô hình cần cân nhắc giữa hiệu suất và chất lượng đầu ra: Trong các ứng dụng thực tế như Spotify, việc gợi ý tức thời là quan trọng, nên có thể cần kết hợp giữa Greedy + bộ lọc hoặc heuristic nhẹ thay vì DP thuần túy.

4. Hướng phát triển

Kết hợp hai phương pháp: Tích hợp phương pháp Greedy làm bước lọc sơ bộ, sau đó áp dụng quy hoạch động để tinh chỉnh kết quả cuối cùng nhằm cân bằng hiệu suất và chất lượng.

Áp dụng học máy (machine learning): Thay vì dùng điểm số thủ công, có thể huấn luyện mô hình học sâu để gợi ý bài hát dựa trên đặc trưng cá nhân hóa cho từng người dùng.

Nâng cao độ chính xác vector đặc trưng: Sử dụng embedding học từ dữ liệu nghe thực tế (ví dụ: word2vec cho nhạc) sẽ cải thiện độ chính xác khi tính cosine similarity.

Tối ưu hoá thuật toán DP: Áp dụng kỹ thuật giảm không gian trạng thái, memoization hoặc beam search để giảm chi phí xử lý trong các hệ thống thực tế.

VIII. Tài liệu tham khảo

[1] Chat GPT

[2] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.

[3] Scikit-learn Documentation. “Cosine Similarity.”

[4] Ricci, F., Rokach, L., & Shapira, B. (2011). Introduction to Recommender Systems Handbook. Springer

[5] Spotify Developer Documentation. “Web API Reference.”

[6] NumPy Documentation. “numpy.mean — NumPy v1.26 Manual.”

[7] Introduction to the Design & Analysis of Algorithms