ĐỒ ÁN MÔN HỌC

TRUY VẤN THÔNG TIN ĐA PHƯƠNG TIỆN

CS336.L11.KHTN

Đề tài : Truy vấn âm thanh

Thành viên:

- Nguyễn Anh Khoa 18520923
- Trần Thị Phương Thảo 18521422
- Võ Quốc An 18520440

Mục lục:

I. Giới thiệu

1. Lý do chọn đề tài

Bạn là người thường xuyên nghe nhạc và tất nhiên bạn chỉ hay nghe những bài mà bạn thích, hay những bài hát trong thể loại mà bạn thích. Mỗi lần nghe nhạc, bạn lại phải chọn từng bài trong cái đống danh sách bài hát đồ sô. Nó thật phiền phức. Vậy nên chúng tôi hướng tới xây dựng một hệ thống có thể để xuất cho bạn những bài hát tương tự như bạn đã nghe. Hay hệ thống có thể giúp bạn tìm kiếm bài hát dựa trên một đoạn nhạc mà bạn có.

2. Đề tài

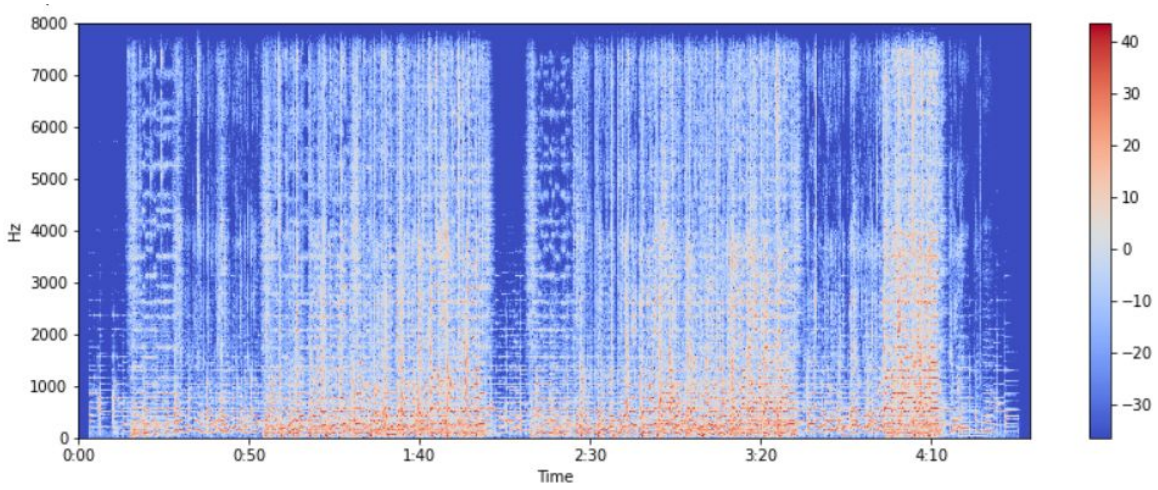
Nội dung đề tài: tìm kiếm bài hát dựa vào âm thanh truyền vào

- Input: Tệp âm thanh định dạng mp3 or wav
- Output: Tên bài hát của tệp âm thanh hoặc những bài hát gần nhất với tệp âm thanh truyền vào

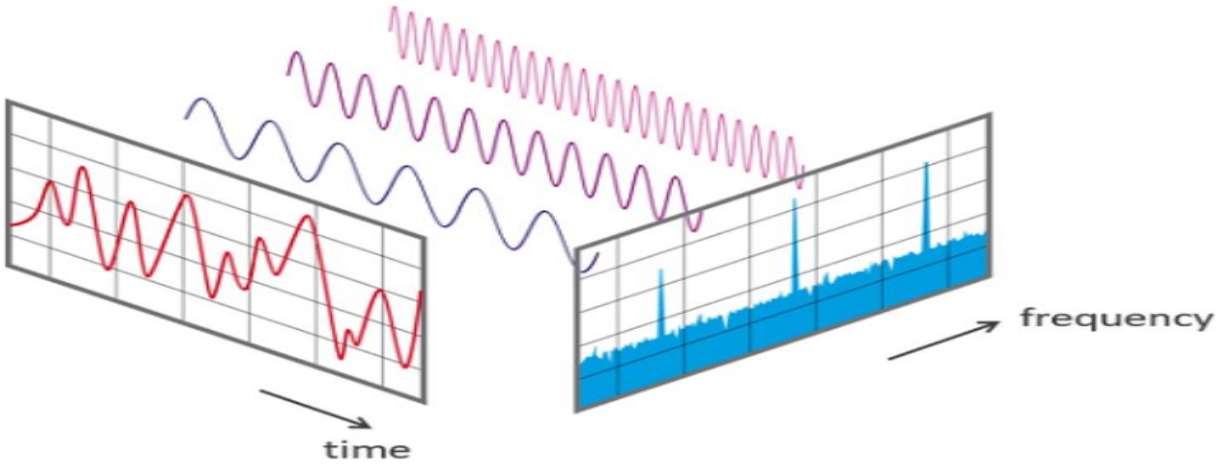
3. Dataset

- Dữ liệu được thu thập từ Zingmp3 theo 3 thể loại:
  - Nhạc trữ tình
  - Nhạc trẻ
  - Nhạc acoustic.
- Công cụ thu thập:
  - Sử dụng công cụ thu thập từ 1 repository trên github
  - https://github.com/hatienl0i261299/Zingmp3
  - Có hướng dẫn sử dụng rõ ràng và khá đơn giản. Nhập vào 3 đường link của 3 thể loại trên Zingmp3 và kết quả trả về là các bài hát thuộc 3 thể loại trên
- Kết quả thu được bộ dataset có 1070 bài hát định dạng mp3.

II. Tổng quan về dữ liệu âm thanh



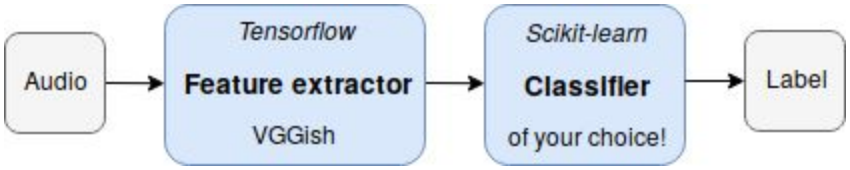
- Âm thanh là một dữ liệu phi cấu trúc
- Dữ liệu dạng âm thanh là một chuỗi các biên độ kèm thời gian tương ứng tức là dữ liệu dạng âm thanh có dạng sóng.
- Các sóng âm thanh được số hóa bằng cách **lấy mẫu** chúng ở những khoảng thời gian riêng biệt được gọi là tốc độ lấy mẫu (**sampling rate).**
- **Sampling rate chuẩn để đảm bảo thông tin được lấy 1 cách toàn vẹn 22000Hz hoặc 44100Hz** .
- Lý do của 2 con số trên là dựa vào ngưỡng nghe của tai người là 22000Hz và dựa vào định lý về lấy mẫu Nyquist Shannon. Phát biểu một cách đơn giản là tần số lấy mẫu ít nhất phải lớn hơn hoặc bằng 2 lần tần số lớn nhất của dãi âm để thu được âm thanh 1 cách trọn vẹn. Trong bài này, nhóm mình sử dụng tần số là 16000Hz, bởi vì nếu sử dụng tần số 22000Hz hoặc 44100Hz thì thu được dữ liệu tốt hơn nhưng cũng đồng nghĩa với việc dữ liệu cho mỗi bài hát cũng tăng lên, do đó hạ thấp tần số 1 ít để có thể xử lý dữ liệu đơn giản hơn nhưng vẫn đảm bảo hiệu quả.
- Mỗi mẫu là biên độ của sóng tại một khoảng thời gian cụ thể



Tóm lại, âm thanh ở dạng tín hiệu liên tục, lấy mẫu với 1 tần số lấy mẫu xác định để chuyển từ dạng liên tục về dạng rời rạc để có thể biểu diễn trên máy tính, quá trình này được thực hiện bằng 1 phép biến đổi gọi là biến đổi Fourier

III. Hướng tiếp cận:

- Để biểu diễn những đặc trưng của mỗi bài hát nhóm mình sử dụng 2 loại đặc trưng:
  - Local feature: đặc trưng cục bộ
  - Deep feature: đặc trưng được lấy từ mạng học sâu



- Đặc trưng cục bộ: sử dụng đặc trưng dựa trên đường bao phổ tần số mfcc (Mel frequency cepstral coefficients)
- Đặc trưng từ mạng học sâu:(tham khảo) **(1)**
  - Sử dụng mạng VGGish - dựa trên mạng VGG16 nhưng bỏ đi 1 số lớp
  - Bằng kết quả thực nghiệm, quá trình truy vấn sử dụng deep feature tốn thời gian và không hiệu quả bằng đặc trưng cục bộ. Trong bản báo cáo này, nhóm mình chỉ để cập về đặc trưng cục bộ.[1]

[1] Mã nguồn sử dụng deep feature:
https://colab.research.google.com/drive/1YzpxIPxY79lagmjYxN_H09STHsdoMqO_#scrollTo=UsR24bMBI7gx
các file chứa deep feature
https://drive.google.com/drive/folders/1Nj8Ur9py05TXE_56fC-8YjznPwcGl201?usp=sharing

IV. Xử lý dữ liệu âm thanh

1. Đặc trưng âm thanh

- Đặc trưng của âm thanh là gì? Đặc trưng của âm thanh chính là các tham số dùng để để phân biệt, nhận dạng, so khớp các mẫu âm thanh với nhau.
- Có một đặc điểm là kích thước toàn bộ tín hiệu âm thanh rất lớn, tín hiệu âm thanh dễ bị biến đổi trong các điều kiện khác nhau nên không thể sử dụng toàn bộ dữ liệu âm thanh làm vector đặc trưng. Hai điều kiện tiên quyết của vector đặc trưng tín hiệu âm thanh là:

  - Phải phân biệt được các mẫu âm thanh
  - Phải tối thiểu hóa chiều dài vector đặc trưng bằng cách loại bỏ tối đa thông tin dư thừa

2. Trích chọn đặc trưng

- Sử dụng thư viện **librosa** và thư viện **python_speech_features** để trích chọn đặc trưng

2.1 Đặc trưng dựa trên biên độ âm thanh

- Chúng ta có thể biểu diễn thông tin âm thanh theo hai hình thức. Ở hình thức thử nhất chúng ta sẽ định nghĩa một tốc độ lấy mẫu và giá trị về biên độ âm thanh làm đặc trưng cho dữ liệu.

```
song = 'path_to_audio'
y, sr = librosa.load(song, sr=16000)
```

- Trong đó y là giá trị của biên độ và sr là **sampling rate** - tốc dộ lấy mẫu. Tuy nhiên việc lấy tham số này làm đặc trưng của âm thanh sẽ dẫn đến nhiều các sai số nhất là đối với bài toán **Tìm kiếm âm thanh** .

2.2 Đặc trưng dựa trên đường bao phổ tần số MFCC

- Tai của con người nhận biết được những âm thanh có tần số thấp (**<1kHz**) tốt hơn những âm thanh có tần số cao. Vì vậy điều quan trọng là cần làm nổi bật lên những âm thanh có tần số thấp hơn là tần số cao. Dải tần số của tín hiệu tiếng nói là khoảng 10kHz. Tần số tiếng nói là dưới 3kHz, cao hơn các thành phần tần số chính liên quan đến người nói, âm nhạc, dụng cụ âm thanh hoặc hiệu ứng. Formants cũng là thông tin quan trọng.
- Cepstral là một phương pháp để trích chọn đặc trưng âm thanh. Trích chọn tham số đặc trưng âm thanh dựa trên hai cơ chế:
    - Mô phỏng lại quá trình cảm nhận âm thanh của tai người.
    - Mô phỏng lại quá trình tạo âm của cơ quan phát âm.
- **MFCC - Mel Frequency Cepstral Coefficient** đây là một trong những phương pháp lấy đặc trưng âm thanh dựa trên phổ tần số phổ biến trong các hệ thống nhận dạng giọng nói, tổng hợp tiếng nói. Chúng tôi sử dụng thư viện **python_speech_features** để xử lý MFCC, đinh nghĩa hàm extract_features:

```
def extract_features(y, sr=16000, nfilt=10, winsteps=0.02):
    try:
        feat = mfcc(y, sr, nfilt=nfilt, winstep=winsteps)
        return feat
    except:
        raise Exception("Extraction feature error")
```

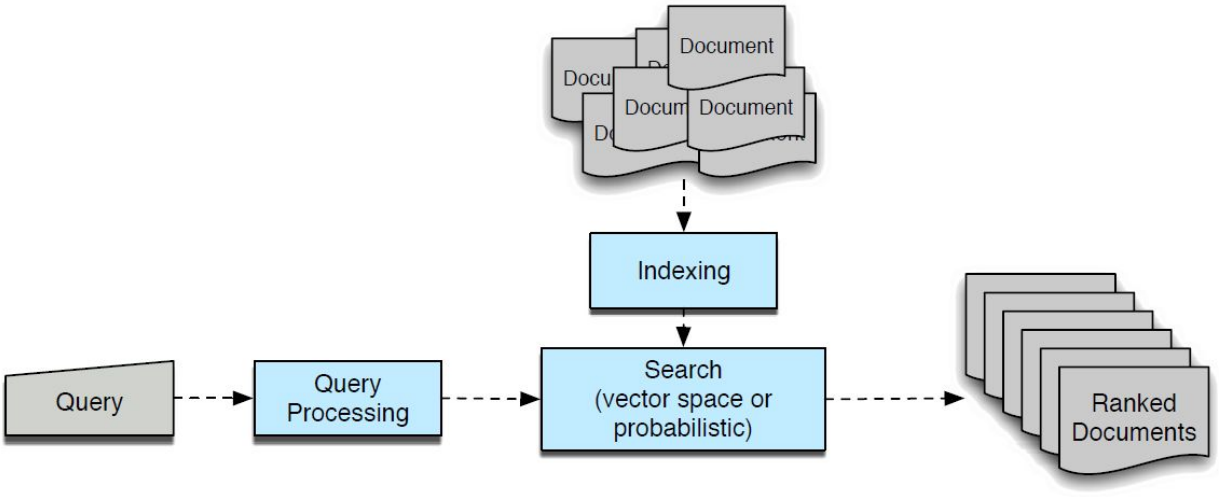
- Trong đó tham số **nfilt** tương ứng với việc độ dài của mỗi thuộc tính MFCC là 10 và **winsteps** tương ứng với việc chúng ta sẽ lấy mẫu theo 0.02 giây cho một thuộc tính MFCC. Như vậy mà nói thì 1 giây chúng ta sẽ sinh ra được 50 MFCC đặc trưng cho âm thanh. Tuy nhiên việc lấy 10 MFCC và thời gian lấy mẫu là 0.02 giây dường như là quá nhỏ và chưa đủ đặc trưng. Chính vì thế chúng tôi sẽ gộp 10 MFCC này lại thành một vector 100 chiều. Đại diện cho mỗi đoạn âm thanh 200ms.

```
def crop_feature(feat, i = 0, nb_step=10, maxlen=100):
    crop_feat = np.array(feat[i : i + nb_step]).flatten()
    print(crop_feat.shape)
    crop_feat = np.pad(crop_feat, (0, maxlen - len(crop_feat)), mode='constant')
    return crop_feat
```

- Mỗi bài hát hay đoạn nhạc sẽ phụ thuộc vào thời gian của nó mà có số lượng vector khác nhau. Chạy trên trục thời gian của bài hát hay đoạn nhạc được truy vấn để lấy ra số lượng vector phù hợp.

```
for song in tqdm(os.listdir(data_dir)):
   song = os.path.join(data_dir, song)
    y, sr = librosa.load(song, sr=16000)
    feat = extract_features(y)
    for i in range(0, feat.shape[0] - 10, 5):
        features.append(crop_feature(feat, i, nb_step=10))
        songs.append(song)
```

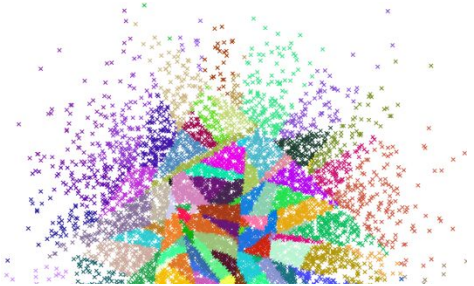
V. Mô hình truy vấn



Hình 1: Sơ đồ của 1 mô hình truy vấn

Truy vấn bằng **Approximate Nearest Neighbors**

- Vì không gian của bài toán rất lớn nên chúng tôi sử dụng giải thuật Approximate Nearest



Neighbors Oh Yeah (Annoy) để giúp việc tìm kiếm có tốc độ nhanh.

- Annoy (Approximate Nearest Neighbors Oh Yeah) là một thư viện C ++ với các liên kết Python để tìm kiếm các điểm trong không gian gần với một điểm truy vấn nhất định. Nó cũng tạo ra các cấu trúc dữ liệu dựa trên tệp chỉ đọc lớn được ánh xạ vào bộ nhớ để nhiều quá trình có thể chia sẻ cùng một dữ liệu.
- Sử dụng thư viện **annoy index** cho việc tìm kiếm vector.

- Add dữ liệu vào trong annoy indexing
  -
```
from annoy import AnnoyIndex

f = 100
t = AnnoyIndex(f)

for i in range(len(features)):
    v = features[i]
    t.add_item(i, v)
```
- Tạo cây index

```
t.build(100) # 100 trees
t.save('music.ann')
```

- Load lại cây indexing trong annoy

```
u = AnnoyIndex(f)

u.load('music.ann')
```

- Tìm kiếm bài hát tương tự
  - Rút trích đặc trưng cho bài hát hay đoạn nhạc cần truy vấn

```
song = os.path.join(data_dir, 'audio.mp3')
y, sr = read_song_frequency(song)
feat = extract_features(y)
```

- Lưu lại tất cả các kết quả có xuất hiện trong top 5 của truy vấn vào trong mảng **results**

```
results = []
for i in range(0, feat.shape[0], 10):
    crop_feat = crop_feature(feat, i, nb_step=10)
    result = u.get_nns_by_vector(crop_feat, n=5)
    result_songs = [songs[k] for k in result]
    results.append(result_songs)

results = np.array(results).flatten()
```

- Và cuối cùng đó là đưa ra top 5 bài hát phù hợp nhất với đoạn âm thanh các bạn vừa nhập vào .

```
from collections import Counter

most_song = Counter(results)
most_song.most_common()[:5]
```

VI. Mã nguồn:

Mã nguồn gồm có:

- File feature: chứa các vector đặc trưng của mỗi bài hát, gồm 2722733 vector 100 chiều
- File song: chứa tên bài hát tương ứng của mỗi vector trong file features
- File Music.ann và Music_.ann chứa 2 cây truy vấn
- File Final_Truyvan.ipynb chứa mã nguồn chính

(nguồn:https://drive.google.com/drive/u/0/folders/1J7RU03uGhHxNy7Mtmr1gmJdfqCeqbaYb)

Hướng dẫn sử dụng mã nguồn:
https://github.com/anvq38/CS336.L11.KHTN?fbclid=IwAR0_fxYIMGrBtHTdvsO5KkRoewmrCJdq3EdSjX1eHGRMWaJMMuY7yXjqpXQ

VII. Tài liệu tham khảo

1. https://github.com/spotify/annoy
2. models/research/audioset/vggish at master · tensorflow/models · GitHub
3. https://arxiv.org/pdf/1709.04396v2.pdf A Tutorial on Deep Learning for Music Information Retrieval
4. Audio Transfer Learning with Scikit-learn and Tensorflow | Jordi Pons

---

Xem thêm:

1. Giải đáp câu hỏi:

Câu hỏi: tại sao cần chuyển âm thanh biểu diễn dưới dạng các giá trị biên độ về dạng tần số?

Trả lời: Âm thanh biểu diễn dưới dạng biên độ chỉ biểu diễn độ lớn của âm thanh, chuyển về miền tần số là các sóng để phân tích được thành phần các sóng của âm thanh từ đó lấy ra được đặc trưng của mỗi bài hát

2. Một số lưu ý:

- Thời gian để load và xử lý tệp âm thanh khá lâu (trung bình để load 1 bản nhạc là 5s) nên chia ra từng phần để xử lý và dễ kiểm soát
- Đặc trưng thu được cho 1070 bài hát là 2722733 vector 100 chiều, bởi vì cứ 200ms sẽ được 1 vector 100 chiều, do đó với từng bài hát sẽ có số lượng vector khác nhau.
- Dùng giải thuật annoy với hơn 2 tr vector trên colab là quá lớn dẫn tới tự động restart process, do đó chia làm phần gồm hơn 1tr vector hoạt động khá tốt