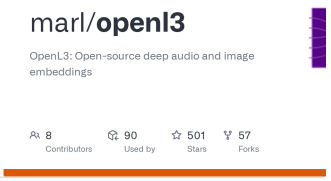
OpenL3 → Vector DB @2025년 3월 19일

GitHub - marl/openl3: OpenL3: Open-source deep audio and image embeddings

OpenL3: Open-source deep audio and image embeddings - marl/openl3

https://github.com/marl/openl3



Chroma

We're currently focused a full public release of Chroma Cloud powered by our open-source distributed and serverless architecture. Chroma Cloud is currently in production in private preview.

https://www.trychroma.com/

1. track이 서버에 생성됨
    a. track.wav를 s3에 저장
    b. RDB에 track 정보가 기록됨
2. fast API로 multipart데이터(track.wav)와 함께

음악 추천 시스템에서 DeepL3 임베딩과 벡터 데이터베이스 활용 방안

1. 딥 오디오 임베딩과 OpenL3 소개

1.1 음악 추천 시스템에서 오디오 임베딩의 필요성

기존의 음악 추천 방식은 주로 협업 필터링(사용자-아이템 상호작용 기반) 또는 메타데이터(장르, 아티스트 등)를 활용한 콘텐츠 기반 접근 방식을 사용했습니다. 이러한 방법들은 음악의 실제 음향적 유사성을 정확하게 포착하는 데 어려움을 겪을 수 있습니다. 딥러닝 기술은 원시 오디오 신호로부터 직접 오디오 임베딩을 추출하여 이러한 한계를 극복할 수 있습니다. 오디오 임베딩은 음악의 의미적, 스타일적 특징을 고차원 벡터 공간에 표현합니다. 서로 다른 곡들의 임베딩을 비교함으로써, 청각적 유사성을 기반으로 음악적으로 유사한 트랙을 식별하고, 이는 종종 사용자의 선호도와 더 잘 일치하는 추천으로 이어집니다**1**. 명시적인 메타데이터나 사용자 상호작용에 의존하는 대신, 오디오 신호 자체로부터 학습하는 방식으로 음악적 유사성을 더 풍부하고 정확하게 표현할 수 있다는 점이 중요합니다. 협업 필터링은 새로운 아이템이나 사용자에 대한 "콜드 스타트" 문제에 직면하며, 메타데이터 기반 접근 방식은 메타데이터의 가용성과 품질에 따라 제한됩니다. 반면, 대규모 음악 데이터셋으로 학습된 오디오 임베딩은 음악의 음향적 본질을 직접 포착하여 이러한 제약을 극복할 수 있습니다. 이는 예상치 못했지만 음향적 속성이 유사한 관련 트랙을 추천하는 데 도움이 될 수 있습니다.

1.2 OpenL3 소개

OpenL3는 딥 오디오 및 이미지 임베딩 계산을 위한 오픈 소스 Python 라이브러리이며, MATLAB에서도 기능을 사용할 수 있습니다**1**. 이 라이브러리는 비디오 데이터에서 오디오와 시각 콘텐츠 간의 대응 관계를 학습하는 자기 지도 학습 방식으로 대규모 AudioSet 데이터셋으로 훈련된 L3-Net 아키텍처를 기반으로 합니다**1**. OpenL3는 다양한 사운드 인식 작업에서 VGGish 및 SoundNet과 같은 기존 오디오 임베딩 기술보다 우수한 성능을 보이는 것으로 나타났으며, 이는 일반적인 오디오 의미 정보를 포착하는 데 효과적임을 시사합니다**1**. Python에서 OpenL3를 설치하는 것은 TensorFlow가 이미 설치되어 있다고 가정할 때 `$ pip install openl3` 명령을 실행하는 것만큼 간단합니다. MATLAB 사용자는 Audio Toolbox™와 Deep Learning Toolbox™가 필요합니다**1**. OpenL3는 오디오 임베딩 생성을 위한 접근성이 높고 성능이 뛰어난 도구를 제공하므로, 고급 음악 추천 시스템 구축에 강력한 후보가 됩니다. AudioSet과 같이 크고 다양한 데이터셋으로 훈련되었다는 점은 다양한 오디오 특성에 대한 폭넓은 이해를 보장합니다. L3-Net의 자기 지도 학습 방식은 방대한 양의 레이블링되지 않은 오디오-시각 데이터를 활용한다는 점에서 특히 강력합니다. 사운드를 해당 시각적 객체와 연결하는 방법을 학습함으로써 네트워크는 오디오 의미 정보에 대한 풍부한 표현을 개발합니다. OpenL3가 다른 인기 있는 방법들보다 성능이 뛰어나다는 사실은 이러한 의미 있는 오디오 특징을 포착하는 능력이 우수함을 나타냅니다.

1.3 보고서 개요

본 보고서는 임베딩 추출, 주요 파라미터, 임베딩의 특징, 벡터 데이터베이스 활용, 실제 고려 사항 및 결론에 대한 섹션을 포함하여 다음과 같은 구조로 구성됩니다.

2. OpenL3 임베딩 추출 이해

2.1 핵심 개념

OpenL3는 오디오 신호를 입력으로 받아 고정 길이의 고차원 벡터인 임베딩을 생성합니다. 이 임베딩은 오디오 콘텐츠의 압축된 표현 역할을 합니다. 기본적인 과정은 오디오 전처리 과정을 거쳐 스펙트로그램(시간에 따른 오디오의 주파수 콘텐츠를 시각적으로 표현한 것)을 생성한 다음, 이 스펙트로그램을 사전 훈련된 심층 신경망에 입력하는 것을 포함합니다**7**. 대규모 데이터셋으로 훈련된 이 네트워크는 다양한 오디오 특성을 임베딩 공간의 특정 영역에 매핑하는 방법을 학습했습니다. 오디오를 스펙트로그램으로 변환한 다음 밀집 벡터 표현으로 변환하는 과정을 통해 신경망은 전통적인 방법으로는 표현하기 어려운 오디오 신호 내의 복잡한 패턴과 관계를 포착할 수 있습니다. 스펙트로그램은 오디오의 시간-주파수 표현을 제공하며, 이는 많은 오디오 분석 작업에서 표준 입력 형식입니다. 심층 신경망은 특징 추출기 역할을 하며, 스펙트로그램으로부터 오디오 콘텐츠의 계층적 표현을 학습합니다. 최종 임베딩 벡터는 비교 및 유사성 분석에 적합한 형식으로 이러한 학습된 특징들을 캡슐화합니다.

2.2 Python 구현

2.2.1 기본 추출

Python에서 오디오 임베딩을 추출하는 가장 간단한 방법은 `openl3.get_audio_embedding(audio, sr)` 함수를 사용하는 것입니다**2**. 이 함수는 오디오 데이터(NumPy 배열 형태)와 해당 샘플링 속도를 입력으로 받습니다. 이 함수는 두 개의 객체를 반환합니다. 첫 번째 객체인 `emb`는 T x D NumPy 배열이며, 여기서 T는 임베딩 프레임의 수(오디오의 서로 다른 시간 세그먼트를 나타냄)이고 D는 임베딩의 차원(6144 또는 512)입니다. 두 번째 객체인 `ts`는 각 임베딩 프레임에 해당하는 타임스탬프를 포함하는 길이 T의 NumPy 배열입니다**3**. 기본적으로 OpenL3는 AudioSet 비디오(주로 음악 공연 포함)로 훈련된 모델을 사용하며, 멜-스펙트로그램(128개 밴드) 표현과 6144의 임베딩 차원을 갖습니다. 출력의 시간적 특성(해당 타임스탬프를 갖는 여러 임베딩 프레임)을 통해 곡 내에서 음악적 특성의 변화를 분석할 수 있으며, 이는 고급 추천 전략에 유용할 수 있습니다. 음악은 시간적인 예술 형태이며, 그 특성은 시간에 따라 크게 변할 수 있습니다. 서로 다른 시간 세그먼트에 대한 임베딩을 제공함으로써 OpenL3는 이러한 시간적 역학 관계를 포착합니다. 이를 통해 곡의 전반적인 유사성뿐만 아니라 구조적 또는 스타일적 발전도 비교할 수 있습니다.

2.2.2 임베딩 파라미터 사용자 정의

`get_audio_embedding` 함수는 임베딩 추출 과정을 사용자 정의하기 위한 여러 파라미터를 제공합니다.

- `content_type`: 모델이 훈련된 오디오 콘텐츠 유형을 지정합니다("env"는 환경 소리, "music"은 음악).

  3

- `input_repr`: 사용된 스펙트로그램 유형을 결정합니다("linear" 또는 "mel128" - 128개의 멜 밴드, 또는 모델 구성에 따라 암시적으로 "mel256").

  3

- `embedding_size`: 출력 임베딩의 차원을 설정합니다(오디오의 경우 512 또는 6144).

  3

- `hop_size`: 연속적인 임베딩 프레임 간의 시간 간격을 제어합니다(기본값은 0.1초).

  3

- `center`: 입력 오디오가 신호 시작 부분에 첫 번째 윈도우가 중앙에 오도록 패딩되는지 여부를 결정하는 부울 파라미터입니다(기본값은 True).

  3

이러한 파라미터를 조정할 수 있는 기능을 통해 사용자는 특정 요구 사항과 음악 라이브러리의 특성에 맞게 임베딩 추출 프로세스를 조정할 수 있습니다. 예를 들어, 라이브러리에 비음악적 오디오가 많은 경우 "env" 콘텐츠 유형을 사용하는 것이 더 적절할 수 있습니다. 다양한 유형의 오디오는 서로 다른 중요한 특징을 가질 수 있습니다. 사용자가 콘텐츠 유형을 선택할 수 있도록 함으로써 OpenL3는 가장 적절한 사전 훈련된 모델이 사용되도록 보장합니다. 마찬가지로, 스펙트로그램 유형 및 임베딩 크기 선택은 임베딩에 포착되는 세부 정보 수준에 영향을 미칠 수 있습니다.

2.2.3 여러 파일 처리

많은 수의 오디오 파일을 처리하기 위해 OpenL3는 `process_audio_file` 함수를 제공합니다**3**. 이 함수는 오디오 파일 경로(또는 파일 목록), 출력 디렉토리 및 임베딩 추출을 사용자 정의하기 위한 선택적 파라미터를 받습니다. 추출된 임베딩과 타임스탬프는 `.npz` 형식으로 디스크에 저장됩니다. 이 함수는 전체 음악 라이브러리에 대한 임베딩 생성을 간소화하여 개별적으로 파일을 처리하는 것보다 훨씬 효율적입니다. 대규모 카탈로그를 위한 음악 추천 시스템을 구축할 때, 일괄 처리를 위한 효율적인 도구를 갖는 것이 중요합니다. `process_audio_file`은 오디오 로딩, 임베딩 추출 및 결과 저장을 처리하여 시스템 초기 설정을 간소화합니다.

2.2.4 사전 훈련된 모델 로드

동일한 구성으로 여러 파일을 처리할 때 불필요한 모델 로딩을 피하기 위해 사전 훈련된 모델을 `openl3.load_audio_model()`을 사용하여 수동으로 로드한 다음 `model` 파라미터를 통해 `get_audio_embedding` 또는 `process_audio_file`에 전달할 수 있습니다**3**. 이 최적화는 대규모 데이터셋을 처리할 때 처리 시간을 크게 줄일 수 있습니다. 심층 학습 모델을 로드하는 것은 시간이 오래 걸리는 작업일 수 있습니다. 모델을 한 번 로드하고 여러 오디오 파일에 재사용함으로써 이러한 오버헤드를 피하고 임베딩 추출 프로세스를 가속화할 수 있습니다.

2.3 MATLAB 구현

2.3.1 `openl3Embeddings` 사용

MATLAB은 Audio Toolbox™에서 OpenL3 특징 임베딩 추출을 위한 `openl3Embeddings` 함수를 제공합니다**5**. 기본 구문은 `embeddings = openl3Embeddings(audioIn, fs)`이며, 여기서 `audioIn`은 오디오 신호이고 `fs`는 샘플링 속도입니다. 이 함수는 시간 경과에 따른 512 또는 6144 요소 특징 벡터의 행렬을 N x L x C 배열(시간 프레임 수, 임베딩 길이, 채널 수)로 반환합니다. MATLAB은 MATLAB 환경의 사용자를 위해 임베딩 추출 프로세스를 단순화하는 고수준 함수를 제공합니다. 출력 형식은 임베딩의 시간적 순서를 명확하게 나타냅니다. MATLAB을 주로 사용하는 엔지니어와 연구자에게 `openl3Embeddings` 함수는 Python으로 전환할 필요 없이 OpenL3를 활용하는 편리한 방법을 제공합니다. 이 함수는 모델 로딩 및 스펙트로그램 계산의 기본 복잡성을 처리합니다.

2.3.2 `openl3Embeddings`를 사용한 파라미터 사용자 정의

`openl3Embeddings` 함수는 추출을 사용자 정의하기 위한 이름-값 쌍 인수를 지원합니다.

- `OverlapPercentage`: 연속적인 스펙트로그램 간의 겹침 비율(기본값 90).

  5

- `SpectrumType`: 스펙트럼 유형('mel128' - 기본값, 'mel256' 또는 'linear').

  5

- `EmbeddingLength`: 출력 임베딩 길이(512 - 기본값 또는 6144).

  5

- `ContentType`: 오디오 콘텐츠 유형('env' - 기본값 또는 'music').

  5

Python 구현과 유사하게 MATLAB에서도 주요 파라미터를 사용자 정의하여 추출된 임베딩의 특성을 제어할 수 있습니다. 이를 통해 Python과 MATLAB 사용자 모두 특정 오디오 데이터 및 애플리케이션 요구 사항에 맞게 OpenL3를 유연하게 조정할 수 있습니다.

2.3.3 `openl3`, `openl3Preprocess` 및 `predict`를 사용한 대체 접근 방식

MATLAB은 사전 훈련된 네트워크를 로드하기 위한 `openl3` 함수, 스펙트로그램을 추출하기 위한 `openl3Preprocess` 함수, 그리고 임베딩을 얻기 위한 `predict` 함수(Deep Learning Toolbox™에서 제공)를 사용하여 더 세분화된 접근 방식을 제공합니다**4**. 이는 먼저 OpenL3용 Audio Toolbox™ 모델을 다운로드하고 압축 해제하는 것을 포함합니다. `openl3` 함수는 사전 훈련된 모델을 나타내는 DAGNetwork 객체를 반환합니다. `openl3Preprocess`는 오디오와 샘플링 속도를 입력으로 받아 스펙트로그램 배열을 반환합니다. 그런 다음 `predict` 함수는 로드된 네트워크를 사용하여 이러한 스펙트로그램으로부터 임베딩을 생성합니다. 이 하위 수준 접근 방식은 임베딩 추출 프로세스의 개별 단계를 더 자세히 제어할 수 있게 하며, 고급 사용자나 디버깅 목적으로 유용할 수 있습니다. 임베딩 추출 프로세스의 개별 구성 요소를 이해하는 것은 모델의 동작을 더 깊이 파고들거나 특정 작업에 맞게 수정하려는 연구자에게 유익할 수 있습니다.

2.3.4 `openl3Features`에 대한 참고 사항

`openl3Features` 함수는 `openl3Embeddings`와 유사하지만 향후 릴리스에서 제거될 수 있습니다. 대신 `openl3Embeddings`를 사용하는 것이 좋습니다**6**. 이 지침은 사용자가 MATLAB에서 최신 및 지원되는 기능을 사용하도록 보장합니다. 더 이상 사용되지 않는 함수를 사용하면 향후 소프트웨어 버전에서 호환성 문제가 발생할 수 있습니다. 사용자에게 권장되는 함수를 안내하면 개발 프로세스가 더 원활해집니다.

3. 임베딩에 영향을 미치는 주요 파라미터

3.1 샘플링 속도

OpenL3 모델은 일반적으로 특정 샘플링 속도(종종 48kHz)의 오디오로 훈련됩니다. 라이브러리가 내부적으로 리샘플링을 처리할 수 있지만, 최적의 성능을 위해서는 예상되는 샘플링 속도로 오디오를 제공하는 것이 일반적으로 권장됩니다. MATLAB에서 Simulink의 `OpenL3 Embeddings` 블록은 샘플링 속도가 48kHz인지 여부에 따라 입력 프레임 길이에 대한 특정 요구 사항이 있습니다. 다른 경우 입력 프레임 길이는 블록이 수행하는 리샘플링 작업의 데시메이션 팩터의 배수여야 합니다. 입력 오디오의 샘플링 속도를 모델 훈련에 사용된 속도와 일치시키는 것이 더 정확하고 대표적인 임베딩을 얻는 데 도움이 될 수 있습니다. 신경망은 특정 샘플링 속도로 추출된 오디오 특징의 패턴을 인식하도록 훈련됩니다. 다른 샘플링 속도로 오디오를 제공하면 이러한 특징이 변경되어 사전 훈련된 모델의 성능이 저하될 수 있습니다.

3.2 중첩 비율

이 파라미터는 Python(암시적으로 `hop_size`를 통해)과 MATLAB(`OverlapPercentage`를 통해) 모두에서 사용할 수 있으며, 스펙트로그램을 생성할 때 연속적인 시간 창 간의 중첩 정도를 제어합니다. 더 높은 중첩률(예: 90%)은 단위 시간당 더 많은 임베딩 프레임을 생성하여 임베딩의 시간적 해상도를 높입니다. 기본 중첩률은 종종 90%이며, 이는 높은 시간적 해상도를 의미합니다. 중첩 비율을 선택하는 것은 추출된 특징의 시간적 세분성에 영향을 미칩니다. 더 높은 중첩률은 음악의 빠른 변화를 포착할 수 있지만, 처리하고 저장해야 하는 임베딩의 수도 증가시킵니다. 악기나 다이내믹의 빠른 변화가 있는 음악의

경우, 더 높은 시간적 해상도가 유리할 수 있습니다. 그러나 변화가 느린 음악의 경우, 더 낮은 중첩률로도 충분하며 계산 효율성이 더 높을 수 있습니다.

3.3 스펙트럼 유형

OpenL3는 신경망의 입력으로 멜(128 또는 256개 주파수 대역) 및 선형의 다양한 유형의 스펙트로그램을 지원합니다. 멜 스펙트로그램은 인간 청각의 비선형 주파수 인식을 모방하도록 설계되었으며, 인간 청각과 관련된 오디오 작업에 자주 사용됩니다. 스펙트럼 유형을 선택하는 것은 네트워크가 학습하는 특징에 영향을 미칠 수 있습니다. 멜 스펙트로그램(특히 128개 대역, 종종 기본값)을 사용하는 것은 인간의 청각 인식과 일치하므로 음악 관련 작업에 일반적으로 좋은 시작점이 됩니다. 인간의 청각 시스템은 주파수를 선형적으로 인식하지 않습니다. 멜 스펙트로그램은 인간이 소리를 인식하는 방식과 더 유사한 표현을 제공하므로 음높이 및 음색과 같은 음악적으로 관련된 특징을 포착하는 데 잠재적으로 더 효과적입니다.

3.4 콘텐츠 유형

`content_type` 파라미터를 사용하면 환경 소리 또는 음악 소리 중 하나로 훈련된 사전 훈련된 모델 중에서 선택할 수 있습니다. 적절한 콘텐츠 유형을 선택하는 것은 의미 있는 임베딩을 얻는 데 매우 중요합니다. 음악 추천의 경우 일반적으로 "musical sounds" 모델을 사용해야 합니다. 적절한 유형의 오디오로 훈련된 모델을 사용하면 네트워크의 학습된 특징이 입력 오디오의 특성과 관련되도록 보장합니다. 환경 소리에 대해 훈련된 신경망은 자동차 경적, 개 짖는 소리 및 음성과 같은 소리를 구별하는 방법을 학습합니다. 이러한 특징은 다양한 장르 또는 스타일의 음악을 구별하는 데는 관련성이 떨어질 수 있습니다. 음악 데이터에 대해 특별히 훈련된 모델을 사용하면 음악적 특성을 더 잘 포착하는 임베딩이 생성됩니다.

3.5 임베딩 길이

`embedding_size` (Python) 또는 `EmbeddingLength` (MATLAB) 파라미터는 출력 임베딩 벡터의 차원을 결정합니다. OpenL3는 일반적으로 오디오 임베딩에 대해 512 또는 6144 차원의 옵션을 제공합니다. 더 높은 차원은 오디오에 대한 더 미세한 세부 정보를 잠재적으로 포착할 수 있지만, 저장 요구 사항과 유사성 비교를 위한 계산 비용도 증가시킵니다. 낮은 차원은 더 넓은 음악적 유사성을 포착하는 데 충분할 수 있으며, 저장 및 계산 측면에서 더 효율적입니다. 임베딩에 포착되는 세부 정보 수준과 필요한 계산 리소스 간에는 균형이 있습니다. 특정 애플리케이션에 대한 최적의 균형을 찾기 위해서는 기본값(종종 Python에서는 6144, MATLAB에서는 함수에 따라 512)으로 시작하여 실험해 보는 것이 필요할 수 있습니다. 더 높은 차원의 임베딩은 더 복잡한 특징 조합을 나타낼 수 있으므로 잠재적으로 다양한 음악 스타일 간의 더 미세한 구별이 가능합니다. 그러나 저장하는 데 더 많은 메모리가 필요하고 비교하는 데 더 많은 계산 능력이 필요합니다. 대규모 음악 라이브러리의 경우, 가장 중요한 음악적 특성을 여전히 포착하면서 효율성을 위해 낮은 차원의 임베딩이 선호될 수 있습니다.

4. 음악을 위한 OpenL3 임베딩의 특징

4.1 의미적 표현

OpenL3 임베딩은 오디오의 고수준 의미적 콘텐츠를 포착하도록 설계되었습니다. 음악의 경우 이는 유사한 음악적 특징(예: 장르, 악기 구성, 분위기, 스타일)을 가진 곡들이 고차원 임베딩 공간에서 서로 가까운 임베딩을 갖는 경향이 있음을 의미합니다. 이러한 속성은 유사성 기반 음악 추천에 적합하게 만듭니다. 벡터 공간에서 임베딩의 근접성은 OpenL3 모델이 인식하는 음악적 유사성의 척도로 작용합니다. AudioSet에 대한 자기 지도 학습을 통해 모델은 의미적으로 유사한 오디오 예제가 임베딩 공간에서 가까운 점으로 매핑되는 표현을 학습할 수 있습니다. 이 학습된 표현은 특정 선호도와 음향적으로 유사한 음악을 찾는 데 활용될 수 있습니다.

4.2 시간 정보

앞서 언급했듯이 임베딩 추출 프로세스는 일반적으로 오디오의 짧은 시간 프레임 각각에 해당하는 임베딩 벡터 시퀀스를 생성합니다. 이 시간적 시퀀스는 곡 내에서 음악적 특징의 변화를 포착합니다. 단일 집계된 임베딩(예: 시간적 임베딩의 평균)을 사용하여 전체 곡을 나타낼 수 있지만, 시간적 시퀀스 자체는 곡 간의 구조적 유사성을 식별하거나 유사한 음악적 발전을 가진 곡을 추천하는 것과 같은 고급 분석에 유용할 수 있습니다. 임베딩의 시간적 측면은 전반적인 음향적 특성을 넘어 음악의 역동적인 특성을 포착할 수 있게 해줍니다. 음악은 정적이지 않고 시간에 따라 변화합니다. 임베딩에 시간 정보를 유지함으로써 이러한 변화를 포착하고, 전반적인 사운드는 약간 다를 수 있지만 유사한 음악적 구조나 진행을 가진 곡을 잠재적으로 식별할 수 있습니다.

4.3 차원

512차원과 6144차원 임베딩 간의 선택은 절충점을 제공합니다. 더 높은 차원의 임베딩은 오디오에 대한 더 미묘하고 상세한 정보를 잠재적으로 포착하여 복잡한 음악 스타일에 대해 잠재적으로 더 정확한 유사성 비교를 가능하게 합니다. 그러나 벡터 데이터베이스에 더 많은 저장 공간이 필요하고 쿼리에 더 많은 계산 리소스가 필요합니다. 낮은 차원의 임베딩은 더 넓은 음악적 유사성을 포착하는 데 충분할 수 있으며, 저장 및 계산 측면에서 더 효율적입니다. 최적의 임베딩 차원은 음악 라이브러리의 특정 요구 사항과 추천의 원하는 세분성 수준을 포함하여 음악 추천 시스템의 특정 요구 사항에 따라 달라집니다. 매우 큰 음악 라이브러리의 경우, 쿼리 성능을 유지하기 위해 낮은 차원의 임베딩을 사용해야 할 수 있습니다. 그러나 목표가 미묘한 스타일 차이에 기반한 매우 구체적인 추천을 제공하는 것이라면, 충분한 계산 리소스가 있다고 가정할 때 더 높은 차원의 임베딩이 선호될 수 있습니다.

4.4 학습 데이터 영향

OpenL3가 훈련된 AudioSet 데이터셋의 특성은 모델이 효과적으로 표현하는 음악적 특징 유형에 영향을 미칩니다. AudioSet은 크고 다양한 데이터셋이지만, 그 구성에 따른 잠재적인 편향이나 제한 사항을 인식하는 것이 중요합니다. 예를 들어, 모델은 AudioSet에 잘 представленный 장르에서 더 나은 성능을 보일 수 있습니다. 학습 데이터를 이해하는 것은 임베딩을 해석하고 다양한 유형의 음악에 대한 모델의 성능을 예측하는 데 도움이 될 수 있습니다. 대상 음악 라이브러리에 AudioSet에 덜 일반적인 장르의 트랙이 상당수 포함되어 있는 경우, OpenL3 임베딩은 더 일반적인 장르의 특정 특성을 효과적으로 포착하지 못할 수 있습니다. 이러한 경우, 더 구체적인 데이터셋으로 모델을 미세 조정하는 것을 고려할 수 있습니다.

5. 음악 추천을 위한 벡터 데이터베이스 활용

5.1 벡터 데이터베이스 소개

벡터 데이터베이스는 고차원 벡터 임베딩의 효율적인 저장 및 검색을 위해 설계된 특수 데이터베이스입니다. 구조화된 데이터에 뛰어난 기존 데이터베이스와 달리 벡터 데이터베이스는 고차원 공간에서 거리 메트릭(예: 코사인 유사도, 유클리드 거리)을 기반으로 한 유사성 검색에 최적화되어 있습니다. 이는 음악 추천을 위해 OpenL3 임베딩으로 작업하는 데 이상적입니다. 벡터 데이터베이스는 OpenL3 임베딩을 기반으로 음악적으로 유사한 곡을 효율적으로 검색하는 데 필요한 인프라를 제공합니다. 음악 라이브러리의 트랙이 OpenL3 임베딩으로 표현되면, 벡터 데이터베이스를 사용하여 선호하는 곡의 임베딩과 가장 유사한 임베딩을 가진 트랙을 빠르게 찾을 수 있습니다. 임베딩 공간에서의 이러한 유사성은 오디오의 지각적 유사성에 해당합니다.

5.2 OpenL3 임베딩 저장

음악 추천을 위해 벡터 데이터베이스를 사용하려면 음악 라이브러리의 OpenL3 임베딩을 데이터베이스에 저장해야 합니다. 각 곡에 대해 일반적으로 임베딩(전체 곡을 나타내는 단일 집계 벡터 또는 시간

적 임베딩 시퀀스)을 추출하고 곡에 대한 메타데이터(예: 제목, 아티스트, 앨범, 트랙 ID)와 함께 저장합니다. 시간적 임베딩을 사용하는 경우 각 프레임의 임베딩을 해당 타임스탬프 또는 프레임 인덱스와 함께 저장할 수 있습니다. 임베딩을 저장하는 방식(집계 대 시간적)은 수행할 수 있는 유사성 쿼리 유형에 영향을 미칩니다. 단일 집계 임베딩을 저장하면 전반적으로 유사한 곡을 찾을 수 있습니다. 시간적 임베딩을 저장하면 유사한 음악적 진행이나 다른 곡의 특정 섹션과 유사한 특정 섹션을 가진 곡을 찾는 것과 같은 더 복잡한 쿼리가 가능합니다.

5.3 유사한 음악 검색

사용자가 특정 곡에 관심을 표명하면(예: 좋아요를 누르거나 선택), 해당 곡의 OpenL3 임베딩을 추출하여 벡터 데이터베이스에 대한 쿼리로 사용할 수 있습니다. 그러면 데이터베이스는 선택된 거리 메트릭을 기반으로 쿼리 임베딩과 가장 유사한 임베딩을 반환합니다. 그런 다음 이러한 유사한 임베딩과 관련된 메타데이터를 검색하여 사용자에게 추천 곡을 제시할 수 있습니다. 거리 메트릭 선택(예: 코사인 유사도, 유클리드 거리)은 검색 결과에서 우선시되는 유사성 유형에 영향을 미칠 수 있습니다. 코사인 유사도는 벡터 간의 각도를 측정하여 크기에 관계없이 의미적 유사성을 포착하므로 고차원 임베딩에 자주 사용됩니다. 코사인 유사도는 오디오 임베딩에 특히 유용합니다. 이는 고차원 공간에서 벡터의 방향에 초점을 맞추는데, 이는 종종 벡터의 크기로 표현되는 전체적인 "크기" 또는 강도보다는 오디오의 의미적 콘텐츠에 해당합니다.

5.4 효율성을 위한 인덱싱 전략

대규모 음악 라이브러리의 경우 가장 유사한 임베딩에 대한 무차별 검색을 수행하는 것은 계산 비용이 많이 들 수 있습니다. 벡터 데이터베이스는 종종 근사 최근접 이웃(ANN) 알고리즘과 같은 인덱싱 기술을 사용하여 검색 프로세스를 가속화합니다. 이러한 기술은 임베딩에 대한 인덱스를 구축하여 정확성과 속도 간의 절충점을 가지면서도 근사 최근접 이웃을 효율적으로 검색할 수 있도록 합니다. 효율적인 인덱싱은 많은 수의 곡을 처리하고 적시에 추천을 제공할 수 있는 확장 가능한 음악 추천 시스템을 구축하는 데 매우 중요합니다. 음악 라이브러리 크기가 증가함에 따라 유사한 임베딩을 검색하는 데 걸리는 시간이 선형적으로 증가합니다. 인덱싱 기술을 통해 벡터 데이터베이스는 검색 공간을 크게 좁힐 수 있으므로 훨씬 빠른 쿼리 시간을 얻을 수 있으며, 이는 반응성이 뛰어난 사용자 경험에 필수적입니다.

5.5 예시 워크플로

1. **임베딩 추출:** 음악 라이브러리의 각 곡에 대해 Python 또는 MATLAB 인터페이스를 사용하여 적절한 파라미터(예: `content_type='music'`, 적절한 `embedding_size`)를 선택하여 OpenL3 임베딩을 추출합니다. 집계된 임베딩 또는 시간적 시퀀스를 사용할지 결정합니다.
2. **벡터 데이터베이스 선택:** 필요에 맞는 벡터 데이터베이스(예: Milvus, Pinecone, Weaviate, ChromaDB)를 선택합니다. 확장성, 비용, 사용 편의성 및 사용 가능한 기능과 같은 요소를 고려합니다.
3. **인덱스 생성:** 선택한 벡터 데이터베이스에 인덱스를 생성합니다. 여기에는 임베딩의 차원과 유사성 검색에 사용할 거리 메트릭을 지정하는 것이 포함됩니다.
4. **데이터베이스 채우기:** 추출된 OpenL3 임베딩을 각 곡에 대한 해당 메타데이터(예: 곡 ID, 제목, 아티스트)와 함께 벡터 데이터베이스에 삽입합니다.
5. **추천을 위한 쿼리:** 사용자가 곡과 상호 작용할 때 해당 임베딩을 벡터 데이터베이스에서 검색합니다. 이 임베딩을 쿼리로 사용하여 데이터베이스에서 유사성 검색을 수행합니다. 검색할 추천 수(상위 k개)를 지정합니다.
6. **메타데이터 검색:** 벡터 데이터베이스에서 반환된 상위 k개 가장 유사한 곡의 메타데이터를 얻습니다.

7. **추천 제시:** 사용자에게 추천된 곡을 표시합니다.

특징	Milvus	Pinecone	Weaviate	ChromaDB
호스팅	자체 호스팅, 클라우드	클라우드 호스팅	자체 호스팅, 클라우드	인메모리 임베딩 데이터베이스
확장성	높음	높음	높음	중간
비용	배포에 따라 다름	사용량 기반	배포에 따라 다름	오픈 소스, 선택적 클라우드 호스팅
사용 편의성	중간	높음	중간	높음
주요 기능	오픈 소스, GPU 가속, 다양한 인덱스 유형	관리형 서비스, 다양한 인덱스 유형, 필터링	GraphQL API, 시맨틱 검색, 지식 그래프	개발자 친화적, LangChain과 쉬운 통합
유사성 메트릭	코사인, L2, IP	코사인, L2, IP	코사인, L2, L1, 내적	코사인, L2

6. 실제 고려 사항 및 워크플로

6.1 데이터 준비

음악 라이브러리가 OpenL3에서 지원하는 형식(예: WAV, OGG, FLAC)의 오디오 파일로 구성되어 있는지 확인하십시오**3**. 손상되거나 잘못된 형식의 오디오 파일과 관련된 잠재적인 문제를 처리합니다. 오디오를 일관된 음량 수준으로 정규화하는 것과 같은 전처리 단계를 고려하십시오. 이는 추출된 임베딩의 일관성을 향상시킬 수 있습니다. 또한 다양한 길이의 곡을 처리하는 방법을 고려해야 합니다. 전체 재생 시간에 대한 임베딩을 추출하거나 특정 세그먼트에 대해서만 추출하도록 선택할 수 있습니다. 깨끗하고 일관된 오디오 데이터는 신뢰할 수 있고 의미 있는 임베딩을 얻는 데 매우 중요합니다. 일관성 없는 오디오 품질 또는 형식은 임베딩 추출 중 오류를 발생시키거나 음악 콘텐츠를 정확하게 나타내지 않는 임베딩으로 이어질 수 있습니다. 정규화는 음량 변화가 유사성 비교에 부당한 영향을 미치지 않도록 하는 데 도움이 될 수 있습니다.

6.2 임베딩 파라미터 선택

`content_type` ('music'으로 시작), `embedding_size` (512와 6144 모두 시도), `hop_size` (시간적 해상도 조정) 및 `SpectrumType`과 같은 파라미터에 대해 다양한 값을 실험하여 특정 음악 라이브러리 및 원하는 추천 결과에 가장 적합한 구성을 확인하십시오. 추천을 질적으로 또는 정량적 메트릭을 사용하여 평가해야 할 수도 있습니다. 최적의 임베딩 파라미터는 음악 데이터의 특성과 추천 시스템의 특정 목표에 따라 달라질 수 있습니다. 예를 들어, 복잡한 화성 구조를 가진 음악은 더 높은 차원의 임베딩이나 특정 스펙트럼 유형의 이점을 누릴 수 있습니다.

6.3 시간적 임베딩 집계 (선택 사항)

시간적 임베딩 시퀀스를 추출하기로 선택한 경우, 유사성 비교를 위해 전체 곡을 나타내는 전략이 필요합니다. 일반적인 접근 방식으로는 곡에 대한 모든 임베딩 벡터를 평균화하거나, 시간 경과에 따른 가장 중요한 특징을 포착하기 위해 최대 풀링을 사용하거나, 시간 차원에 걸쳐 통계(예: 평균, 표준 편차)를 계산하는 것과 같은 더 복잡한 방법을 사용하는 것이 있습니다. 또는 일부 벡터 데이터베이스는 벡터 시퀀스를 기반으로 한 쿼리를 지원하여 더 정교한 시간적 유사성 비교를 허용할 수 있습니다. 집계 방법(사용된 경우)은 곡의 유사성을 측정하는 방법에 큰 영향을 미칠 수 있습니다. 평균화는 시간에 따른 곡의 특성에 대한 일반적인 표현을 제공합니다. 최대 풀링은 가장 눈에 띄는 특징에 초점을 맞춥니다. 가장 적합한 접근 방식은 포착하려는 음악적 유사성의 유형에 따라 달라질 수 있습니다.

6.4 추천 성능 평가

음악 추천의 품질을 평가하는 방법이 있어야 합니다. 여기에는 정밀도(추천된 항목 중 관련성이 있는 항목의 비율), 재현율(관련 항목 중 추천된 항목의 비율), 참신성(추천이 사용자가 이미 상호 작용한 항목과 얼마나 다른지) 및 다양성(추천된 항목이 얼마나 다양한지)과 같은 메트릭을 사용하는 것이 포함될 수 있습니다. 추천 품질에 대한 주관적인 피드백을 얻기 위해 사용자 연구 또는 A/B 테스트를 고려할 수도 있습니다. 객관적이고 주관적인 평가는 음악 추천 시스템을 반복적으로 개선하는 데 매우 중요합니다. 적절한 평가 없이는 시스템 변경(예: 다른 임베딩 파라미터, 벡터 데이터베이스 설정)이 실제로 사용자에게 더 나은 추천으로 이어지는지 알기 어렵습니다.

7. 결론 및 잠재적 응용 분야

7.1 주요 내용 요약

OpenL3를 사용하여 오디오 임베딩을 추출하는 프로세스와 이러한 임베딩을 벡터 데이터베이스와 함께 활용하여 지각적 유사성을 기반으로 음악 추천 시스템을 구축하는 방법을 간략하게 요약합니다. 임베딩의 파라미터와 특징을 이해하는 것의 중요성과 이러한 시스템 구축을 위한 실제 고려 사항을 다시 강조합니다.

7.2 음악 추천 외 잠재적 응용 분야

의미 있는 오디오 임베딩을 추출하는 OpenL3의 능력은 음악 추천을 넘어 확장됩니다. 이러한 임베딩은 음악 장르 분류, 유사성을 기반으로 한 자동 재생 목록 생성, 오디오 핑거프린팅 및 식별, 그리고 일반 오디오에서의 사운드 이벤트 감지와 같은 작업에 사용될 수 있습니다. OpenL3는 다양한 오디오 분석 작업을 위한 다재다능한 도구를 제공합니다. OpenL3 임베딩이 포착하는 의미 정보는 다양한 유형의 사운드를 분류하거나 유사한 오디오 세그먼트를 찾는 등 오디오 콘텐츠 이해가 필요한 모든 애플리케이션에서 유용할 수 있습니다.

7.3 향후 연구 방향

향후 연구에서는 음악적 특징의 시간적 변화를 모델링하기 위해 순환 신경망을 사용하는 것과 같이 음악 추천을 위해 OpenL3 임베딩의 시간 정보를 활용하는 더 고급 기술을 탐구할 수 있습니다. 특정 음악 데이터셋에 대해 OpenL3 모델을 미세 조정하거나 하이브리드 추천 시스템에서 OpenL3 임베딩을 다른 유형의 특징(예: 협업 필터링 데이터, 가사)과 결합하는 것의 영향도 유망한 연구 분야입니다. 딥 오디오 임베딩을 사용하여 음악 추천 시스템을 더욱 향상시킬 수 있는 많은 기회가 있습니다. 음악 추천 분야는 끊임없이 발전하고 있습니다. 음악적 유사성을 모델링하고 다양한 정보원을 통합하는 더 정교한 방법을 탐구하면 사용자에게 더욱 개인화되고 관련성 높은 추천을 제공할 수 있습니다.

🤗 음악 추천 시스템에서 DeepL3(OpenL3)로 추출한 임베딩을 벡터 DB와 함께 활용하는 방식을 자세히 설명해 드릴게요. 이를 통해 유사한 음악을 추천하는 이유와 방법을 이해할 수 있습니다. 조금만 기다려 주세요.

OpenL3 임베딩 개념

OpenL3는 **L3-Net**이라고 불리는 딥러닝 모델을 기반으로 한 오디오 임베딩 기법입니다. 이 모델은 소리와 시각 정보를 함께 학습하여, 인간이 인식하는 음악의 특징을 벡터 형태로 추출하도록 설계되었습니다 (News - Justin Salamon). 구체적으로, 동영상 속 **오디오-비디오 대응관계**(예: 특정 장면에 등장하는 소리)를 self-supervised 방식으로 학습하여, 비슷한 소리는 벡터 공간에서 가깝게 매핑되도록 합니다 (News - Justin Salamon). OpenL3는 오디오 신호(예: raw 파형이나 스펙트로그램)를 **합성곱 신경망**(CNN)에 통과시켜 **국소적인 음향 특징**들을 뽑아내고, 심층 계층에서 이를 요약하여 **압축된 벡터 표현**을 만들어냅니다 (Top 10 Audio Embedding Models for AI: A Complete Guide - Zilliz Learn). 이러한 임베딩 벡터는 음악의 **음색, 피치, 리듬** 등 **본질적 특성**들을 함축하고 있어, 원본 오디오의 중요한 특징들을 담은 숫자 벡터라고 볼 수 있습니다 (Top 10 Audio Embedding Models for AI: A Complete Guide - Zilliz Learn).

OpenL3 임베딩 벡터의 차원은 모델 설정에 따라 다르지만, 기본값으로 **6144차원**을 사용합니다 (OpenL3 tutorial — OpenL3 0.4.2 documentation). 이는 OpenL3의 기본 모델이 **AudioSet**이라는 거대한 데이터셋(음악이 포함된 동영상 모음)으로 학습되면서, 오디오를 1초 길이의 여러 윈도우로 나누어 분석한 결과 각 윈도우마다 6144차원의 벡터를 산출하도록 설계되었기 때문입니다 (OpenL3 tutorial — OpenL3 0.4.2 documentation). 경우에 따라 이 벡터의 크기를 **512차원**으로 줄이는 것도 가능한데, 이는 임베딩의 압축 수준을 조절하는 옵션입니다 (OpenL3 tutorial — OpenL3 0.4.2 documentation). 예를 들어 환경음 등 다른 콘텐츠 유형에 대해서도 학습된 모델을 사용할 수 있으며, 입력 표현(input representation)으로 **멜-스펙트로그램**(기본 128 밴드) 등을 활용합니다 (OpenL3 tutorial — OpenL3 0.4.2 documentation). 요약하면, OpenL3는 **음악의 고차원적 특징 공간**에서 유사한 소리일수록 가까운 벡터로 매핑해 주는 임베딩 기법으로, 이를 통해 한 곡의 **사운드 특징**을 숫자 벡터로 나타낼 수 있습니다 (Top 10 Audio Embedding Models for AI: A Complete Guide - Zilliz Learn). 이러한 벡터화 덕분에 사람의 장르 구분이나 분위기 감지와 비슷한 역할을 기계가 수행할 수 있게 됩니다. 또한 OpenL3는 **VGGish**나 **SoundNet** 등의 이전 모델보다도 여러 음향 인식 태스크에서 뛰어난 성능을 보이는 것으로 보고되어 있습니다 (News - Justin Salamon).

벡터 DB의 역할

음악 임베딩 벡터를 활용하려면, 많은 곡들의 벡터들을 효율적으로 저장하고 검색할 수 있는 데이터베이스가 필요합니다. 일반적인 **관계형 DB**(Relational Database)는 표 형태의 스키마에 따라 데이터를 저장하고 SQL로 질의하는 방식으로, **정확한 조건 일치**에는 강하지만 **유사한 항목 찾기**에는 한계가 있습니다 (ELI5: What is the difference between vector database and relational database? : r/explainlikeimfive). 즉, "장르가 같은 곡"이나 "아티스트명이 동일한 곡"은 SQL로 쉽게 찾을 수 있지만, **"소리가 비슷한 곡"**을 찾기 위한 추상적인 유사도 질의는 전통적인 DB로 처리하기 어렵습니다 (ELI5: What is the difference between vector database and relational database? : r/explainlikeimfive). 반면 **벡터 데이터베이스(Vector DB)**는 이러한 임베딩 **벡터 형태의 비정형 데이터**를 저장하고 다룰 수 있도록 특화된 데이터베이스입니다 (Choosing Between Relational and Vector Databases - Zilliz blog). 관계형 DB가 행과 열로 데이터를 관리하는 데 비해, 벡터 DB는 데이터를 고차원 공간의 **벡터로 저장**하며, 이미지나 오디오, 텍스트 등의 데이터를 벡터 형태로 받아들여 관리할 수 있습니다 (Choosing Between Relational and Vector Databases - Zilliz blog).

벡터 DB를 사용하면 얻을 수 있는 주요 이점은 **대규모 벡터 유사도 검색의 효율성**입니다. 전문적인 벡터 DB 시스템(예: Milvus, FAISS, Chroma 등)은 수백만에서 수억 개에 이르는 고차원 벡터들도 **근접한 이웃**을 빠르게 찾을 수 있게 해줍니다 (Choosing Between Relational and Vector Databases - Zilliz blog). 이는 벡터 DB가 내부적으로 **고급 인덱싱 기법**(예: HNSW 그래프 등)을 사용하여, 벡터들 간의 거리를 빠르게 비교할 수 있는 구조를 갖추고 있기 때문입니다 (How to Leverage Vector Databases for Audio Information Retrieval). 이러한 **근사 최근접 이웃(ANN)** 알고리즘들은 약간의 정확도 손실을 감수하는 대신 대용량 데이터에서 **실시간에 가까운 검색 성능**을 제공합니다 (How to Leverage Vector Databases for Audio Information Retrieval). 예를 들어 Spotify에서는 수백만 곡의 임베딩 벡터 중 유사한 것을 찾아야 하는데, 이때 자체 개발한 Annoy와 같은 벡터 검색 라이브러리를 사용하여 메모리 효율을 높이면서도 신속한 검색을 구현했습니다 (How would you go about building a content based recommendation system with reinforcement? : r/datascience). 요약하면, 벡터 DB는 **임베딩된 데이터에 대한 유사도 기반 질의**를 지원함으로써, 음악 추천 시스템에서 필수적인 **"비슷한 노래 찾기"** 작업을 효과적으로 수행하게 해주는 역할을 합니다. 또한 벡터와 함께 메타데이터(예: 장르, 아티스트 등)를 저장해 **하이브리드 질의**(유사도 + 속성 필터)도 가능하므로, 유사한 노래 중 사용자가 선호하는 아티스트의 곡만 추려내는 등의 응용도 가능합니다.

유사도 검색 알고리즘

벡터 DB에서는 두 임베딩 벡터 간의 **유사도**를 계산하여 가장 가까운 항목들을 찾습니다. 대표적인 유사도 측정 방법으로 **코사인 유사도**(Cosine similarity)와 **유클리드 거리**(Euclidean distance)가 많이 활용됩니다 (Embeddings and Vector Databases With ChromaDB – Real Python). **코사인 유사도**는 두 벡터 사이의 **각도**에 따른 유사도를 나타내는 지표로, 두 벡터의 내적을 각 벡터 크기의 곱으로 나눈 값으로 정의됩니다. 값의 범위는 -1부터 1까지이며, **1에 가까울수록 방향이 동일(매우 유사)**하고 0이면 관계가 없고 -1이면 정반대임을 의미합니다 (Embeddings and Vector Databases With ChromaDB – Real Python). 코사인 유사도의 큰 특징은 벡터의 **크기(에너지)**를 무시하고 **방향**만 고려한다는 점인데, 이를 통해 음량이나 파일 길이 등이 다르더라도 **패턴이 비슷한** 두 오디오 임베딩을 가려낼 수 있습니다 (Embeddings and Vector Databases With ChromaDB – Real Python). 예를 들어 코사인 유사도가 1에 가깝다는 것은 특정 곡의 임베딩과 방향성이 거의 같은 벡터를 가지는 다른 곡을 의미하며, 이는 두 곡이 매우 흡사한 분위기나 사운드 특성을 지님을 시사합니다. 한편 **유클리드 거리**는 고차원 공간에서 두 벡터 간의 **직선 거리**를 재는 방식으로, 값이 작을수록 두 벡터가 가깝다는 뜻입니다. 만약 모든 벡터를 정규화(단위 벡터화)했다면 유클리드 거리 순위는 코사인 유사도 순위와 거의 동일해지므로, 둘 중 편한 지표를 선택하면 됩니다. 실제 벡터 DB들은 **메트릭 옵션**으로 코사인, 내적(dot product), L2(유클리드) 거리 등을 제공하며, 응용에 맞게 선택하여 최근접 이웃을 검색할 수 있습니다 (Embeddings and Vector Databases With ChromaDB – Real Python). 음악 추천에서는 보통 코사인 유사도를 많이 사용하며, 임베딩 벡터끼리 이루는 각도가 작을수록(유사도가 높을수록) 더 유사한 곡으로 판단합니다. 이처럼 수치적으로 계산된 **벡터 유사도**를 토대로 가장 가까운 곡들을 찾아내는 것이 유사도 기반 추천의 핵심입니다.

추천 시스템 설계 과정

이제 OpenL3 임베딩과 벡터 DB를 활용하여 **음악 추천 시스템**을 설계하는 전체 흐름을 단계별로 살펴보겠습니다:

1. **음악 임베딩 사전 생성**: 추천에 활용할 **모든 곡들에 대해 임베딩 벡터를 미리 계산**합니다. 각 오디오 파일을 OpenL3 모델에 입력하여 벡터를 얻으며, 곡이 길다면 1초 단위 등의 프레임마다 여러 개의 벡터가 나옵니다. 이 경우 **여러 벡터의 평균값**을 사용하거나 서로 이어 붙이는 등의 방법으로 곡당 하나의 **대표 임베딩**을 만듭니다 (How to Leverage Vector Databases for Audio Information Retrieval). 이렇게 하면 한 곡을 하나의 고차원 벡터로 요약할 수 있습니다.

2. **벡터 DB에 저장**: 계산된 곡 임베딩들을 벡터 DB에 삽입하여 **벡터 색인**을 구축합니다. 이때 각 벡터에는 해당 곡을 식별할 수 있는 **ID나 메타데이터**(예: 곡 제목, 아티스트, 장르 등)를 함께 저장합니다. 벡터 DB는 이 벡터들을 효율적으로 검색하기 위한 인덱스를 생성해 두므로, 이후 유사한 벡터 질의가 빠르게 이루어질 수 있습니다. 저장 단계에서는 주로 `collection.add()`와 같은 함수를 통해 한번에 다수의 벡터를 추가합니다.
3. **사용자 입력 임베딩 변환**: 사용자가 한 곡을 입력하거나 현재 듣고 있는 곡에 기반해 추천을 요청하면, **그 곡의 임베딩 벡터**를 얻습니다. 만약 해당 곡도 이미 DB에 포함되어 있다면 미리 계산된 벡터를 불러올 수 있고, 새로운 입력(예를 들어 사용자가 업로드한 녹음 파일 등)이라면 OpenL3를 사용해 실시간으로 임베딩을 계산합니다. 이 단계에서도 OpenL3 `get_audio_embedding` 함수를 이용하며, 기존에 평균 처리 등을 했다면 동일하게 적용합니다.
4. **벡터 DB 유사도 검색**: 앞서 얻은 **사용자 입력 곡의 벡터**를 쿼리로 하여, 벡터 DB에서 **가장 유사한 곡들의 벡터**를 검색합니다. 벡터 DB의 API (`collection.query()` 등)를 사용하여 쿼리 벡터와 **코사인 유사도** 등이 가장 높은 상위 *k*개의 벡터를 찾습니다 (Top 10 Audio Embedding Models for AI: A Complete Guide - Zilliz Learn). 내부적으로는 벡터 DB가 저장된 모든 곡 임베딩과 쿼리 벡터 간 거리를 계산하고, 거리(또는 유사도) 순으로 결과를 정렬합니다. 이때 수만~수백만 개의 후보 곡 중에서 근사 최근접 이웃 알고리즘으로 빠르게 추려내므로, 사용자 요청에 실시간에 가까운 응답을 제공할 수 있습니다.
5. **추천 결과 반환**: 검색된 상위 유사 곡들의 ID를 통해 실제 곡 **메타데이터**(제목, 아티스트 등)를 조회합니다. 그리고 이 **추천 곡 리스트**를 사용자에게 제공합니다. 예를 들어 "이 곡과 비슷한 노래" 기능을 구현했다면, 해당 결과에 나온 곡들을 사용자에게 플레이리스트 형태로 보여주거나 바로 재생할 수 있도록 합니다. 필요에 따라 유사도 점수를 함께 제공하거나, 너무 이미 사용자가 잘 아는 곡은 제외하는 등의 후처리를 거칠 수도 있습니다. 전체적으로 이 파이프라인을 통해, **한 곡의 임베딩 → 가까운 임베딩들의 곡 목록**이라는 흐름으로 유사한 음악 추천이 이루어집니다.

이러한 콘텐츠 기반 추천 과정은 신규 곡이나 인기 기록이 부족한 곡도 **오디오 자체의 특징**만으로 추천 가능하다는 장점이 있습니다. 또한 사용자별 선호도와 결합한 **하이브리드 추천**에 활용되어, 사용자가 좋아하지만 아직 들어보지 않은 새로운 아티스트의 노래를 제안하는 등 폭넓은 응용이 가능합니다.

Python 구현 예시 (OpenL3 + ChromaDB)

이제 실제 Python 코드로 OpenL3 임베딩을 벡터 DB에 적용하는 간단한 예를 살펴보겠습니다. 아래 예시는 `openl3` 라이브러리를 사용해 오디오 파일에서 임베딩을 추출하고 (OpenL3 tutorial — OpenL3 0.4.2 documentation), 해당 벡터를 **ChromaDB**에 저장한 뒤, 유사한 곡을 질의하는 과정을 보여줍니다:

```
import openl3
import soundfile as sf
from chromadb import Client
from chromadb.config import Settings

# 1. OpenL3를 사용하여 오디오 파일을 임베딩 벡터로 변환
audio, sr = sf.read("song1.wav")
emb, _ = openl3.get_audio_embedding(audio, sr)      # 다수의 프레임별 임베딩 반환
song1_vec = emb.mean(axis=0)                        # 여러 임베딩의 평균으로 대표 벡터 산출

# 2. ChromaDB 클라이언트 초기화 및 컬렉션 생성
client = Client(Settings())
```

```
collection = client.create_collection(name="music", metadata={"hnsw:space": "cosi
ne"})

# 3. 벡터 DB에 임베딩 추가 (곡 메타데이터와 함께)
collection.add(
    embeddings=[song1_vec.tolist()],
    metadatas=[{"title": "Song 1", "artist": "Artist A"}],
    ids=["song1"]
)

# (다른 곡들도 동일한 방식으로 임베딩을 추출하여 추가)

# 4. 새로운 곡에 대한 임베딩 계산 (사용자 질의 곡)
audio2, sr2 = sf.read("query_song.wav")
emb2, _ = openl3.get_audio_embedding(audio2, sr2)
query_vec = emb2.mean(axis=0)

# 5. 벡터 DB에서 유사한 곡 검색 (코사인 유사도 기반 상위 5개)
results = collection.query(
    query_embeddings=[query_vec.tolist()],
    n_results=5,
    include=["metadatas", "distances"]
)
for meta, dist in zip(results["metadatas"][0], results["distances"][0]):
    print(f"추천 곡: {meta['title']} - {meta['artist']}, 유사도 거리: {dist:.3f}")
```

위 코드에서는 `song1.wav`라는 오디오 파일을 불러와 **OpenL3 임베딩**을 계산한 뒤, `song1_vec`이라는 1차원 벡터로 얻고 있습니다. 그런 다음 ChromaDB에 `"music"` 컬렉션을 만들어 해당 벡터를 **추가(add)** 하고, 가상의 `query_song.wav`에 대해서도 임베딩을 뽑아 **질의(query)** 하고 있습니다. `metadata={"hnsw:space": "cosine"}`로 설정하여 코사인 유사도 기반의 검색을 사용하고 있으며, `n_results=5`로 가장 가까운 5개의 결과를 요청합니다. 최종적으로 `results`에서 유사한 곡들의 메타데이터와 거리를 출력하면, 입력 곡과 **소리가 가장 비슷한 상위 5곡**을 확인할 수 있습니다. 실제 응용에서는 이 과정이 백엔드에서 수행되고, 결과로 얻은 추천 곡들을 사용자 인터페이스에 보여주게 됩니다. (예를 들어, "이 노래와 비슷한 곡" 리스트를 생성하여 앱에 표시할 수 있습니다.) 이처럼 OpenL3로 임베딩을 만들고 ChromaDB와 같은 벡터 DB에 저장/질의하는 구현을 통해 **콘텐츠 기반 음악 추천 시스템**을 구축할 수 있습니다.

활용 사례 및 실제 적용

현재 많은 **음악 스트리밍 서비스**들이 OpenL3와 유사한 오디오 임베딩 기법과 벡터 DB를 활용하여 음악 추천 품질을 높이고 있습니다. 예를 들어 **Spotify**는 청취 이력을 분석할 뿐만 아니라, 곡 자체의 **오디오 특성 벡터**를 활용한 콘텐츠 기반 필터링을 적용합니다 (How to Leverage Vector Databases for Audio Information Retrieval). 앞서 언급한 Annoy 라이브러리는 Spotify 내부에서 **유사한 곡 또는 유사한 사용자를 찾기 위해 개발된 근사 이웃 검색용 도구**로서, 수많은 곡을 고차원 임베딩 공간에 배치하고 빠르게 최근접 항목을 찾아내는 데 사용되었습니다 (How would you go about building a content based recommendation system with reinforcement? : r/datascience). 이를 통해 사용자가 좋아할 만한 새로운 곡을 음향적 유사성에 근거하여 찾아내고, 기존 인기곡에 치우치지 않는 다양한 추천을 제공하고 있습니다.

YouTube Music을 비롯한 다른 서비스들도 콘텐츠 기반 임베딩을 적극 활용하고 있습니다. Google의 경우 방대한 YouTube 동영상을 기반으로 한 **AudioSet** 데이터셋을 공개하고, 거기서 추출한 **VGGish**와 같은 오디오 임베딩 모델을 활용해왔는데, 이러한 기술은 YouTube Music의 음악 분류와 추천에 응용되고 있습니다. 예를 들어 YouTube Music은 사용자에게 새로운 음악을 추천할 때, 사용자가 감상한 곡들의 **오디오 임베딩**을 분석하여 **음향적으로 유사한 노래**들을 찾아 재생 목록을 만들곤 합니다. 이처럼 **음향 신호 그 자체의 유사성**을 기반으로 추천하면, 가령 특정 분위기(예: 잔잔한 어쿠스틱 기타 사운드)의 곡을 좋아하는 사용자에게 다른 비슷한 분위기의 신곡을 발견해주는 식의 개인화된 추천이 가능합니다. 결과적으로 OpenL3와 벡터 DB 조합은 **Spotify, YouTube Music** 등에서 **콘텐츠 기반 음악 추천**을 실현하는 핵심 요소로 활용되고 있으며 (How to Leverage Vector Databases for Audio Information Retrieval), 새로운 음악 발견(discovery)부터 자동 플레이리스트 생성, 라디오 기능 등에 이르기까지 폭넓게 기여하고 있습니다. 실제로 **유사도 검색** 기술은 음악 도메인 외에도 동영상 추천, 이미지 검색 등 다양한 미디어 서비스에서 공통적으로 사용되고 있어, 벡터 DB와 임베딩을 활용한 추천 시스템은 현대 AI 응용에서 표준적인 구조가 되어가고 있습니다.

Sources: 音