Roie Schwaber-Cohen(Developer Advocate)

벡터 데이터베이스란 무엇 입니까?

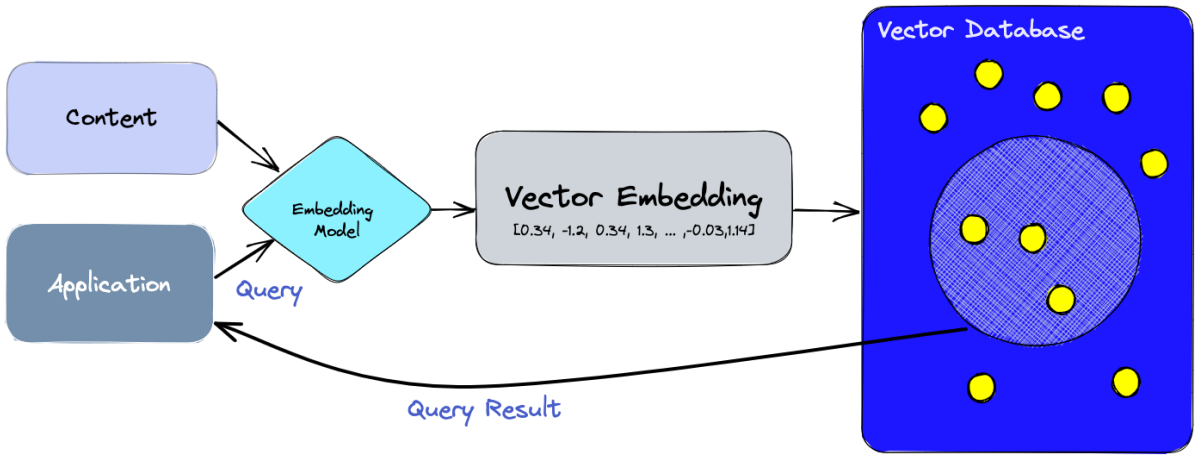
우리는 AI 혁명의 중심에 있습니다. 그것은 영향을 미치는 모든 산업을 뒤엎고 위대한 혁신을 약속하지만 새로운 도전을 제시합니다. 효율적인 데이터 처리는 대규모 언어 모델, 생성 AI 및 시맨틱 검색을 포함하는 애플리케이션에서 그 어느 때보다 중요해졌습니다.
이러한 모든 새로운 애플리케이션은 AI 가 복잡한 작업을 실행할 때 사용할 수 있는 장기 메모리를 이해하고 유지하는 데 중요한 의미론적 정보를 내부에 전달하는 데이터 표현 유형인 벡터 임베딩에 의존합니다.

임베딩은 AI 모델(예: 대규모 언어 모델)에 의해 생성되며 많은 속성 또는 기능이 있어 표현을 관리하기 어렵게 만듭니다. AI 및 기계 학습의 맥락에서 이러한 기능은 패턴, 관계 및 기본 구조를 이해하는 데 필수적인 데이터의 다양한 차원을 나타냅니다.
그렇기 때문에 이러한 유형의 데이터를 처리하도록 특별히 설계된 특수 데이터베이스가 필요합니다. Pinecone 과 같은 **벡터 데이터베이스**는 임베딩을 위한 최적화된 스토리지 및 쿼리 기능을

제공하여 이 요구 사항을 충족합니다. 벡터 데이터베이스는 독립형 벡터 인덱스에는 없는 기존 데이터베이스의 기능과 기존 스칼라 기반 데이터베이스에는 없는 벡터 임베딩 처리 전문화 기능을 가지고 있습니다.

벡터 임베딩 작업의 어려움은 기존의 스칼라 기반 데이터베이스가 이러한 데이터의 복잡성과 규모를 따라갈 수 없어 통찰력을 추출하고 실시간 분석을 수행하기 어렵다는 것입니다. 여기서 벡터 데이터베이스가 사용됩니다. 벡터 데이터베이스는 이러한 유형의 데이터를 처리하고 데이터를 최대한 활용하는 데 필요한 성능, 확장성 및 유연성을 제공하도록 의도적으로 설계되었습니다.

벡터 데이터베이스를 사용하면 의미론적 정보 검색, 장기 기억 등과 같은 고급 기능을 AI 에 추가할 수 있습니다. 아래 다이어그램은 이러한 유형의 애플리케이션에서 벡터 데이터베이스의 역할을 더 잘 이해할 수 있게 해줍니다.



이것을 분석해 보겠습니다.

1. 먼저 **임베딩 모델을** 사용하여 인덱싱하려는 **콘텐츠** 에 대한 **벡터 임베딩을** 생성합니다 .
2. 벡터 **임베딩은** 임베딩이 생성된 원본 **콘텐츠 에 대한 일부 참조와 함께 벡터 데이터베이스** 에 삽입됩니다 .
3. **애플리케이션** 이 쿼리를 발행 하면 동일한 **임베딩 모델을** 사용하여 쿼리에 대한 임베딩을 생성하고 해당 임베딩을 사용하여 **데이터베이스** 에서 *유사한* 벡터 임베딩을 쿼리합니다. 앞서 언급한 바와 같이 유사한 임베딩은 임베딩 을 생성하는 데 사용된 원본 **콘텐츠 와 연결됩니다.**

벡터 인덱스와 벡터 데이터베이스의 차이점은 무엇입니까?

FAISS (Facebook AI Similarity Search) 와 같은 독립형 벡터 인덱스는 벡터 임베딩의 검색 및 검색을 크게 향상시킬 수 있지만 데이터베이스에 존재하는 기능이 부족합니다. 반면에 벡터 데이터베이스는 벡터

임베딩을 *관리하기* 위해 특별히 제작되어 독립형 벡터 인덱스를 사용하는 것보다 몇 가지 이점을 제공합니다.

1. **데이터 관리:** 벡터 데이터베이스는 데이터 삽입, 삭제 및 업데이트와 같은 데이터 저장을 위해 잘 알려져 있고 사용하기 쉬운 기능을 제공합니다. 따라서 스토리지 솔루션과 통합하기 위해 추가 작업이 필요한 FAISS 와 같은 독립형 벡터 *인덱스를* 사용하는 것보다 벡터 데이터를 쉽게 관리하고 유지할 수 있습니다 .
2. **메타데이터 저장 및 필터링:** 벡터 데이터베이스는 각 벡터 항목과 관련된 메타데이터를 저장할 수 있습니다. 그런 다음 사용자는 보다 세분화된 쿼리를 위해 추가 메타데이터 필터를 사용하여 데이터베이스를 쿼리할 수 있습니다.
3. **확장성:** 벡터 데이터베이스는 증가하는 데이터 볼륨과 사용자 요구에 따라 확장되도록 설계되어 분산 및 병렬 처리에 대한 더 나은 지원을 제공합니다. 독립형 벡터 인덱스는 유사한 수준의 확장성을 달성하기 위해 사용자 지정 솔루션이 필요할 수 있습니다(예: Kubernetes 클러스터 또는 기타 유사한 시스템에서 배포 및 관리).
4. **실시간 업데이트:** 벡터 데이터베이스는 종종 실시간 데이터 업데이트를 지원하여 데이터를 동적으로 변경할 수 있는 반면, 독립형 벡터 인덱스는 새로운 데이터를 통합하기 위해 전체 재인덱싱 프로세스가 필요할 수 있으므로 시간이 많이 걸리고 계산 비용이 많이 들 수 있습니다.
5. **백업 및 수집:** 벡터 데이터베이스는 데이터베이스에 저장된 모든 데이터를 백업하는 일상적인 작업을 처리합니다. 또한 Pinecone 을 사용하면 나중에 사용할 수 있도록 해당 인덱스에 데이터를 저장하는 "컬렉션" 형태로 백업할 수 있는 특정 인덱스를 선택적으로 선택할 수 있습니다.
6. **에코시스템 통합:** 벡터 데이터베이스는 ETL 파이프라인(예: Spark), 분석 도구(예: Tableau 및 Segment) 및 시각화 플랫폼(예: Grafana)과 같은 데이터 처리 에코시스템의 다른 구성 요소와 보다 쉽게 통합되어 데이터 관리 워크플로를 간소화할 수 있습니다. 또한 LangChain , LlamaIndex 및 ChatGPT 의 플러그인 과 같은 다른 AI 관련 도구와 쉽게 통합할 수 있습니다 .
7. **데이터 보안 및 액세스 제어:** 벡터 데이터베이스는 일반적으로 내장된 데이터 보안 기능과 액세스 제어 메커니즘을 제공하여 독립 실행형 벡터 인덱스 솔루션에서는 사용할 수 없는 중요한 정보를 보호합니다.

즉, 벡터 데이터베이스는 확장성 문제, 번거로운 통합 프로세스, 실시간 업데이트 및 내장 보안 조치의 부재와 같은 독립형 벡터 인덱스의 한계를 해결하여 벡터 임베딩을 처리하기 위한 우수한 솔루션을 제공합니다. 효과적이고 능률적인 데이터 관리 경험.

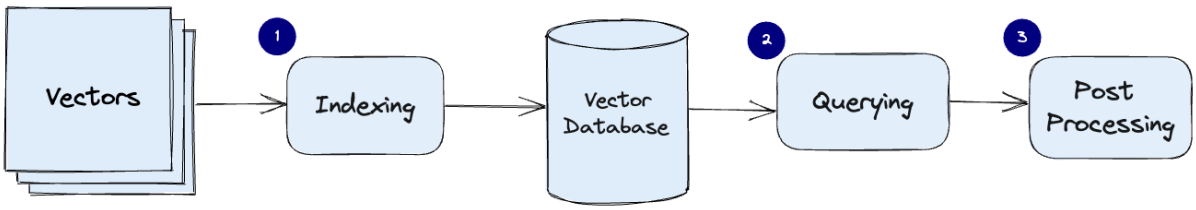
벡터 데이터베이스는 어떻게 작동합니까?

우리는 모두 전통적인 데이터베이스가 어떻게 작동하는지(다소) 알고 있습니다. 데이터베이스는 문자열, 숫자 및 기타 유형의 스칼라 데이터를 행과 열에 저장합니다. 반면에 벡터 데이터베이스는 벡터에서 작동하므로 최적화 및 쿼리 방식이 상당히 다릅니다.

기존 데이터베이스에서는 일반적으로 값이 일반적으로 쿼리와 정확히 일치하는 데이터베이스의 행을 쿼리합니다. 벡터 데이터베이스에서는 유사성 메트릭을 적용하여 쿼리와 **가장 유사한** 벡터를 찾습니다 .

벡터 데이터베이스는 ANN(Approximate Nearest Neighbor) 검색에 모두 참여하는 서로 다른 알고리즘의 조합을 사용합니다. 이러한 알고리즘은 해싱, 양자화 또는 그래프 기반 검색을 통해 검색을 최적화합니다. 이러한 알고리즘은 쿼리된 벡터의 이웃을 빠르고 정확하게 검색하는 파이프라인으로 어셈블됩니다. 벡터 데이터베이스는 **대략적인** 결과를 제공하므로 우리가 고려하는 주요 트레이드 오프는 정확도와 속도 사이입니다. 결과가 정확할수록 쿼리 속도가 느려집니다. 그러나 좋은 시스템은 거의 완벽에 가까운 정확도로 초고속 검색을 제공할 수 있습니다.

다음은 벡터 데이터베이스에 대한 일반적인 파이프라인입니다.



1. **인덱싱**: 벡터 데이터베이스는 PQ, LSH 또는 HNSW(자세한 내용은 아래 참조)와 같은 알고리즘을 사용하여 벡터를 인덱싱합니다. 이 단계는 더 빠른 검색을 가능하게 하는 데이터 구조에 벡터를 매핑합니다.
2. **쿼리**: 벡터 데이터베이스는 가장 가까운 이웃을 찾기 위해 인덱스 쿼리 벡터를 데이터 세트의 인덱스 벡터와 비교합니다(해당 인덱스에서 사용하는 유사성 메트릭 적용).
3. **사후 처리**: 경우에 따라 벡터 데이터베이스는 데이터 세트에서 가장 가까운 최종 이웃을 검색하고 사후 처리하여 최종 결과를 반환합니다. 이 단계에는 다른 유사성 척도를 사용하여 가장 가까운 이웃의 순위를 재지정하는 작업이 포함될 수 있습니다.

다음 섹션에서는 이러한 각 알고리즘에 대해 자세히 설명하고 이들이 벡터 데이터베이스의 전체 성능에 어떻게 기여하는지 설명합니다.

알고리즘

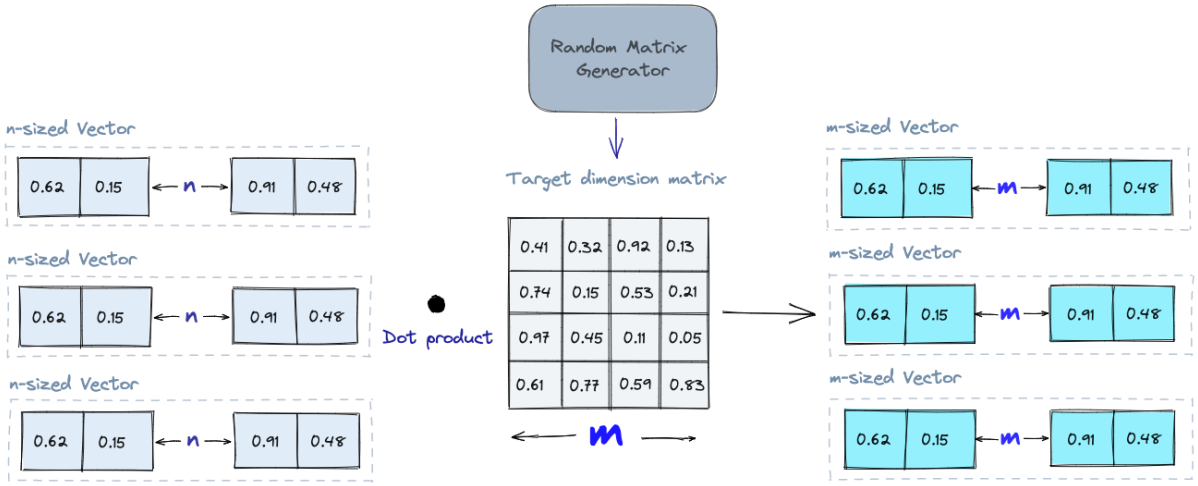
여러 알고리즘을 사용하여 벡터 인덱스 생성을 용이하게 할 수 있습니다. 이들의 공통 목표는 빠르게 순회할 수 있는 데이터 구조를 생성하여 빠른 쿼리를 가능하게 하는 것입니다. 일반적으로 원래 벡터의 표현을 압축된 형식으로 변환하여 쿼리 프로세스를 최적화합니다.

그러나 Pinecone 사용자는 이러한 다양한 알고리즘의 복잡함과 선택에 대해 걱정할 필요가 없습니다. Pinecone 은 백그라운드에서 모든 복잡성과 알고리즘 결정을 처리하도록 설계되어 번거로움 없이 최상의 성능과 결과를 얻을 수 있습니다. Pinecone 의 전문 지식을 활용하면 귀중한 통찰력을 추출하고 강력한 AI 솔루션을 제공하는 등 진정으로 중요한 일에 집중할 수 있습니다.

다음 섹션에서는 벡터 임베딩을 처리하기 위한 몇 가지 알고리즘과 고유한 접근 방식을 살펴봅니다. 이 지식을 통해 정보에 입각한 결정을 내리고 애플리케이션의 잠재력을 최대한 발휘할 때 Pinecone 이 제공하는 원활한 성능을 감상할 수 있습니다.

무작위 투영

무작위 투영의 기본 아이디어는 무작위 투영 행렬을 사용하여 고차원 벡터를 저차원 공간에 투영하는 것입니다. 난수 행렬을 만듭니다. 행렬의 크기는 우리가 원하는 목표 저차원 값이 될 것입니다. 그런 다음 입력 벡터와 행렬의 내적을 계산하여 원래 벡터보다 차원이 적지만 유사성을 유지하는 **투영 행렬을 생성합니다.**



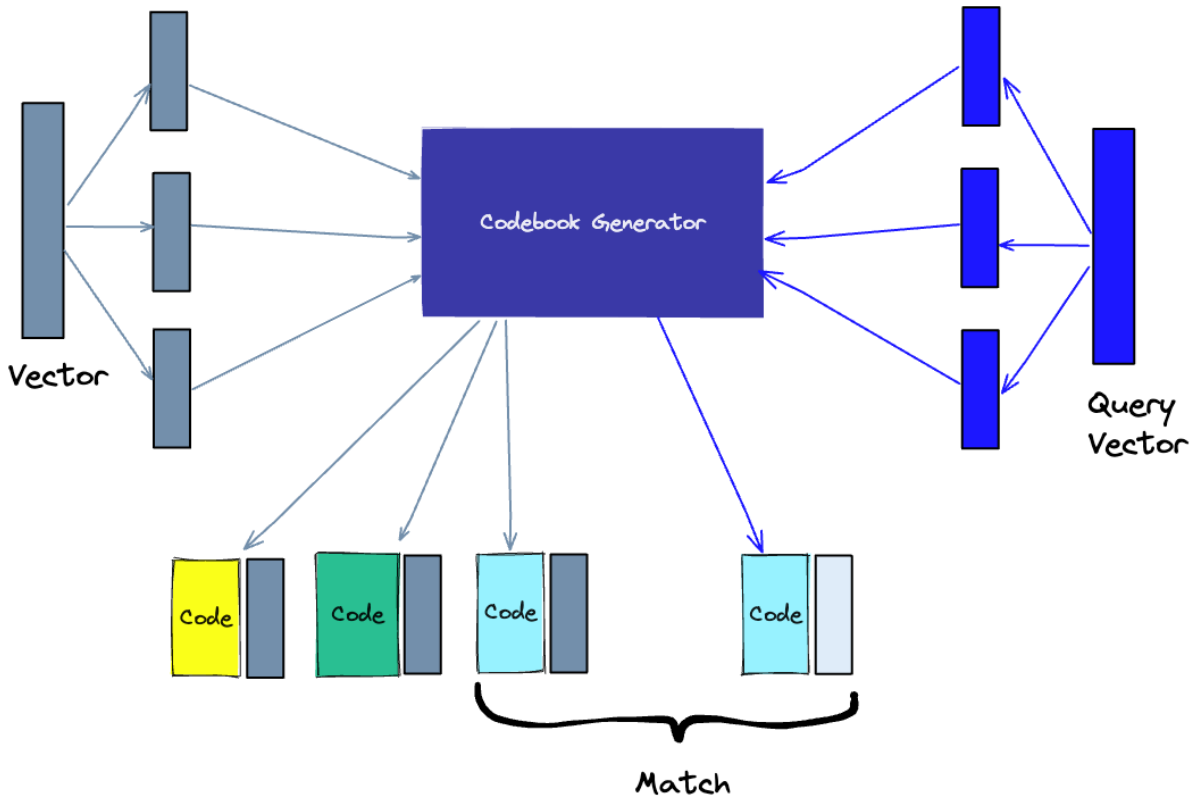
쿼리할 때 동일한 투영 행렬을 사용하여 쿼리 벡터를 저차원 공간에 투영합니다. 그런 다음 투영된 쿼리 벡터를 데이터베이스의 투영된 벡터와 비교하여 가장 가까운 이웃을 찾습니다. 데이터의 차원이 줄어들기 때문에 전체 고차원 공간을 검색하는 것보다 검색 프로세스가 훨씬 빠릅니다.

무작위 프로젝션은 대략적인 방법이며 프로젝션 품질은 프로젝션 매트릭스의 속성에 따라 달라집니다. 일반적으로 프로젝션 매트릭스가 더 무작위적일수록 프로젝션의 품질이 더 좋아집니다. 그러나

진정한 무작위 프로젝션 매트릭스를 생성하는 것은 특히 대규모 데이터 세트의 경우 계산 비용이 많이 들 수 있습니다. 무작위 프로젝션에 대해 자세히 알아보세요.

제품 양자화

인덱스를 구축하는 또 다른 방법은 벡터 임베딩과 같은 고차원 벡터에 대한 *손실 압축 기술인 제품 양자화(PQ)입니다.* 원본 벡터를 가져와서 더 작은 청크로 나누고 각 청크에 대한 대표 "코드"를 생성하여 각 청크의 표현을 단순화한 다음 유사성 작업에 중요한 정보를 잃지 않고 모든 청크를 다시 결합합니다. PQ 프로세스는 분할, 교육, 인코딩 및 쿼리의 네 단계로 나눌 수 있습니다.



1. **분할** - 벡터가 세그먼트로 나뉩니다.
2. **교육** - 각 세그먼트에 대한 "코드북"을 작성합니다. 간단히 말해서 알고리즘은 벡터에 할당할 수 있는 잠재적인 "코드" 풀을 생성합니다. 실제로 - 이 "코드북"은 벡터의 각 세그먼트에서 k-평균 클러스터링을 수행하여 생성된 클러스터의 중심점으로 구성됩니다. 세그먼트 코드북에는 k-평균 클러스터링에 사용하는 값과 동일한 수의 값이 있습니다.
3. **인코딩** - 알고리즘은 각 세그먼트에 특정 코드를 할당합니다. 실제로 훈련이 완료된 후 코드북에서 각 벡터 세그먼트에 가장 가까운 값을 찾습니다. 세그먼트에 대한 PQ 코드는

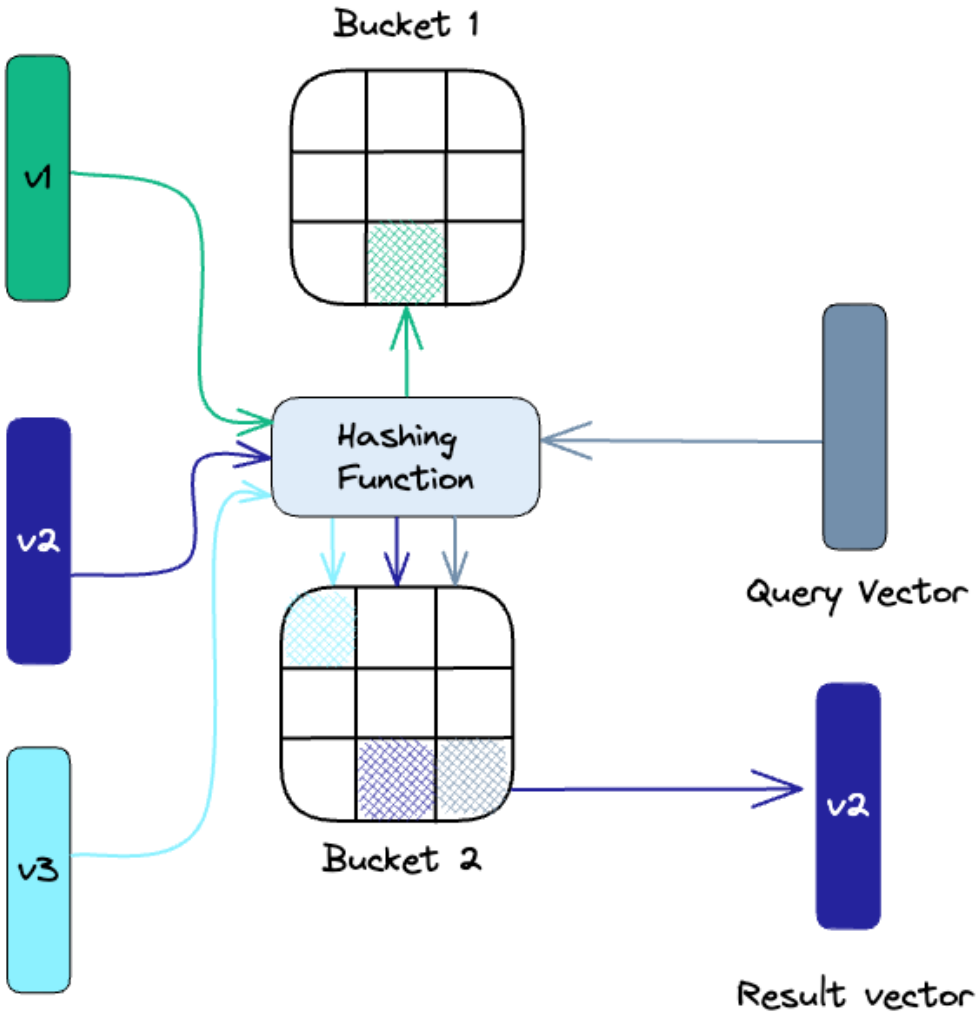
코드북의 해당 값에 대한 식별자가 됩니다. 원하는 만큼 많은 PQ 코드를 사용할 수 있습니다. 즉, 코드북에서 여러 값을 선택하여 각 세그먼트를 나타낼 수 있습니다.

4. **쿼리** - 쿼리할 때 알고리즘은 벡터를 하위 벡터로 분해하고 동일한 코드북을 사용하여 양자화합니다. 그런 다음 인덱싱된 코드를 사용하여 쿼리 벡터에 가장 가까운 벡터를 찾습니다.

코드북의 대표 벡터 수는 표현의 정확도와 코드북 검색의 계산 비용 간의 균형입니다. 코드북의 대표 벡터가 많을수록 하위 공간의 벡터 표현이 더 정확해지지만 코드북을 검색하는 계산 비용이 높아집니다. 대조적으로, 코드북의 대표 벡터가 적을수록 표현의 정확도는 떨어지지만 계산 비용은 낮아집니다. PQ 에 대해 자세히 알아보세요 .

지역에 민감한 해싱

LSH(Locality-Sensitive Hashing)는 근사 최근접 이웃 검색 컨텍스트에서 인덱싱하는 기술입니다. 속도에 최적화되어 있으면서도 대략적이고 포괄적이지 않은 결과를 제공합니다. LSH 는 아래와 같이 일련의 해싱 함수를 사용하여 유사한 벡터를 "버킷"으로 매핑합니다.

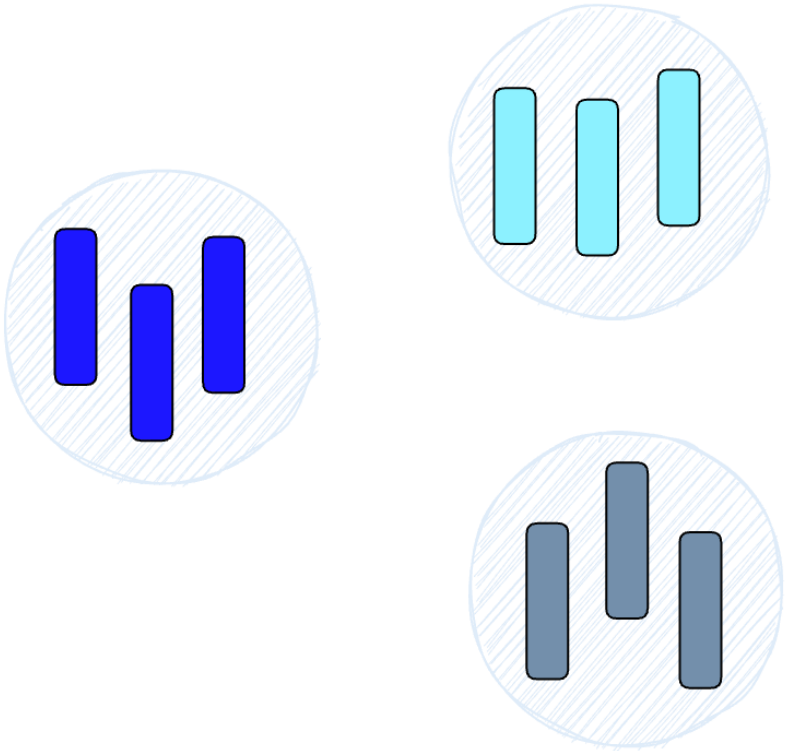


주어진 쿼리 벡터에 대한 가장 가까운 이웃을 찾기 위해 유사한 벡터를 해시 테이블로 "버킷"하는 데 사용되는 것과 동일한 해싱 함수를 사용합니다. 쿼리 벡터는 특정 테이블로 해시된 다음 동일한 테이블의 다른 벡터와 비교되어 가장 일치하는 항목을 찾습니다. 이 방법은 전체 공간보다 각 해시 테이블의 벡터가 훨씬 적기 때문에 전체 데이터 세트를 검색하는 것보다 훨씬 빠릅니다.

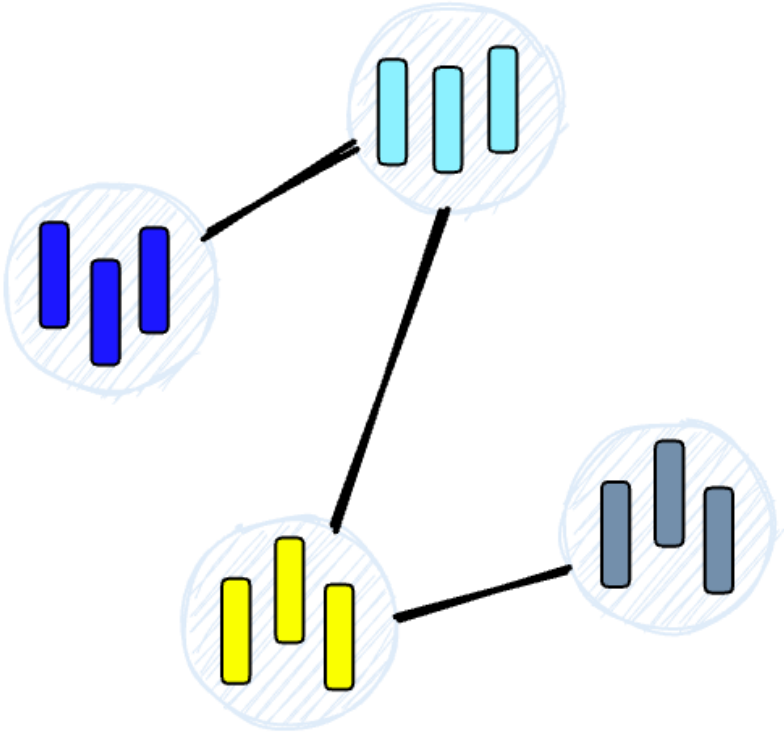
LSH 는 근사 방법이며 근사의 품질은 해시 함수의 속성에 따라 다르다는 점을 기억하는 것이 중요합니다. 일반적으로 더 많은 해시 함수를 사용할수록 근사 품질이 더 좋아집니다. 그러나 많은 수의 해시 함수를 사용하면 계산 비용이 많이 들 수 있으며 대규모 데이터 세트에는 적합하지 않을 수 있습니다. LSH 에 대해 자세히 알아보세요.

HSNW(계층적 탐색 가능 소형 세계)

HSNW 는 트리의 각 노드가 벡터 세트를 나타내는 계층적 트리와 같은 구조를 생성합니다. 노드 사이의 모서리는 벡터 간의 **유사성을 나타냅니다.** 알고리즘은 각각 적은 수의 벡터가 있는 노드 집합을 생성하는 것으로 시작합니다. 이것은 임의로 또는 각 클러스터가 노드가 되는 k-평균과 같은 알고리즘으로 벡터를 클러스터링하여 수행할 수 있습니다.



그런 다음 알고리즘은 각 노드의 벡터를 검사하고 해당 노드와 해당 노드와 가장 유사한 벡터를 가진 노드 사이에 에지를 그립니다.



HSNW 인덱스를 쿼리할 때 이 그래프를 사용하여 트리를 탐색하고 쿼리 벡터에 가장 가까운 벡터를 포함할 가능성이 가장 높은 노드를 방문합니다. HSNW 에 대해 자세히 알아보세요 .

유사성 측정

이전에 논의한 알고리즘을 기반으로 벡터 데이터베이스에서 유사성 측정의 역할을 이해해야 합니다. 이러한 측정은 벡터 데이터베이스가 주어진 쿼리에 대해 가장 관련성이 높은 결과를 비교하고 식별하는 방법의 기초입니다.

유사성 측정은 두 벡터가 벡터 공간에서 얼마나 유사한지를 결정하기 위한 수학적 방법입니다. 유사성 측정은 데이터베이스에 저장된 벡터를 비교하고 주어진 쿼리 벡터와 가장 유사한 벡터를 찾기 위해 벡터 데이터베이스에서 사용됩니다.

다음을 포함하여 여러 유사성 측정을 사용할 수 있습니다.

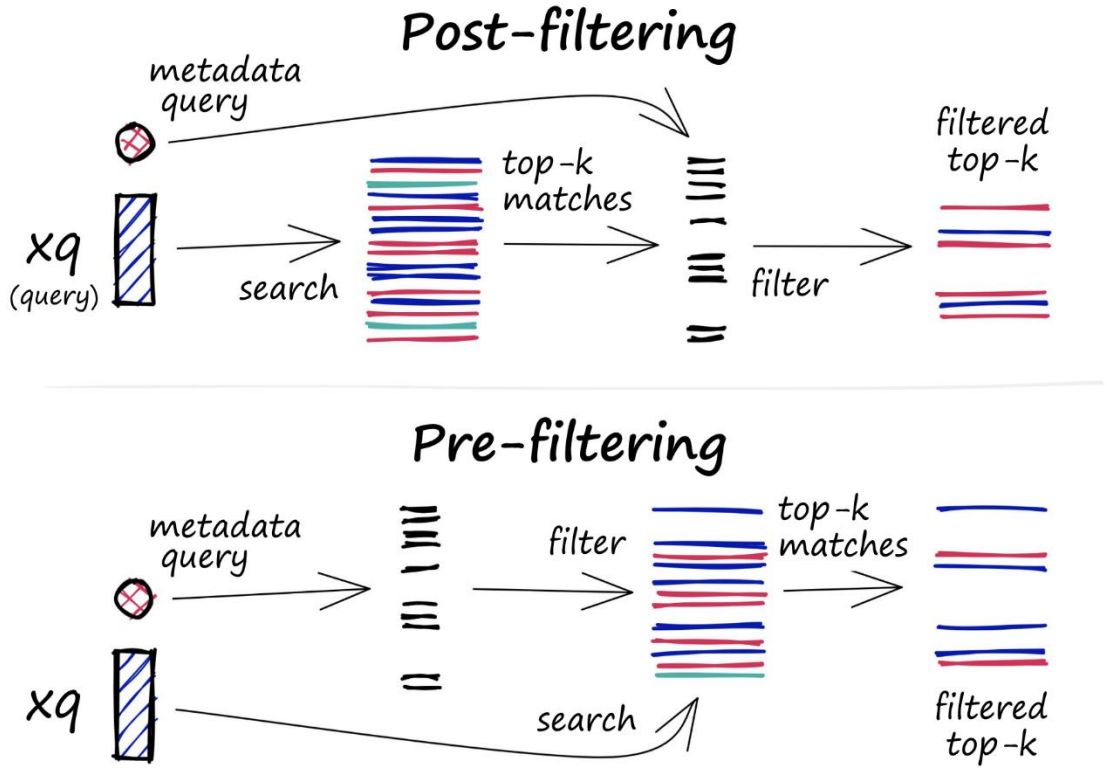
- **코사인 유사성:** 벡터 공간에서 두 벡터 간의 각도의 코사인을 측정합니다. 범위는 -1 에서 1 까지이며, 여기서 1 은 동일한 벡터를 나타내고, 0 은 직교 벡터를 나타내고, -1 은 정반대의 벡터를 나타냅니다.
- **유클리드 거리:** 벡터 공간에서 두 벡터 사이의 직선 거리를 측정합니다. 범위는 0 에서 무한대까지이며, 여기서 0 은 동일한 벡터를 나타내고 값이 클수록 점점 더 다른 벡터를 나타냅니다.

- **내적:** 두 벡터 크기의 곱과 두 벡터 사이 각도의 코사인 값을 측정합니다. 범위는 -∞에서 ∞까지이며 양수 값은 같은 방향을 가리키는 벡터를 나타내고 0 은 직교 벡터를 나타내고 음수 값은 반대 방향을 가리키는 벡터를 나타냅니다.

유사성 측정의 선택은 벡터 데이터베이스에서 얻은 결과에 영향을 미칩니다. 각 유사성 측정에는 고유한 장점과 단점이 있으며 사용 사례와 요구 사항에 따라 올바른 것을 선택하는 것이 중요합니다. 유사성 측정에 대해 자세히 알아보세요.

필터링

데이터베이스에 저장된 모든 벡터에는 메타데이터도 포함됩니다. 유사한 벡터를 쿼리하는 기능 외에도 벡터 데이터베이스는 메타데이터 쿼리를 기반으로 결과를 필터링할 수도 있습니다. 이를 위해 벡터 데이터베이스는 일반적으로 벡터 인덱스와 메타데이터 인덱스라는 두 개의 인덱스를 유지합니다. 그런 다음 벡터 검색 자체 전이나 후에 메타데이터 필터링을 수행하지만 두 경우 모두 쿼리 프로세스가 느려지는 어려움이 있습니다.



필터링 프로세스는 벡터 검색 자체 전이나 후에 수행할 수 있지만 각 접근 방식에는 쿼리 성능에 영향을 줄 수 있는 고유한 문제가 있습니다.

- **사전 필터링:** 이 접근 방식에서는 벡터 검색 전에 메타데이터 필터링이 수행됩니다. 이렇게 하면 검색 공간을 줄이는 데 도움이 되지만 시스템에서 메타데이터 필터 기준과 일치하지 않는 관련

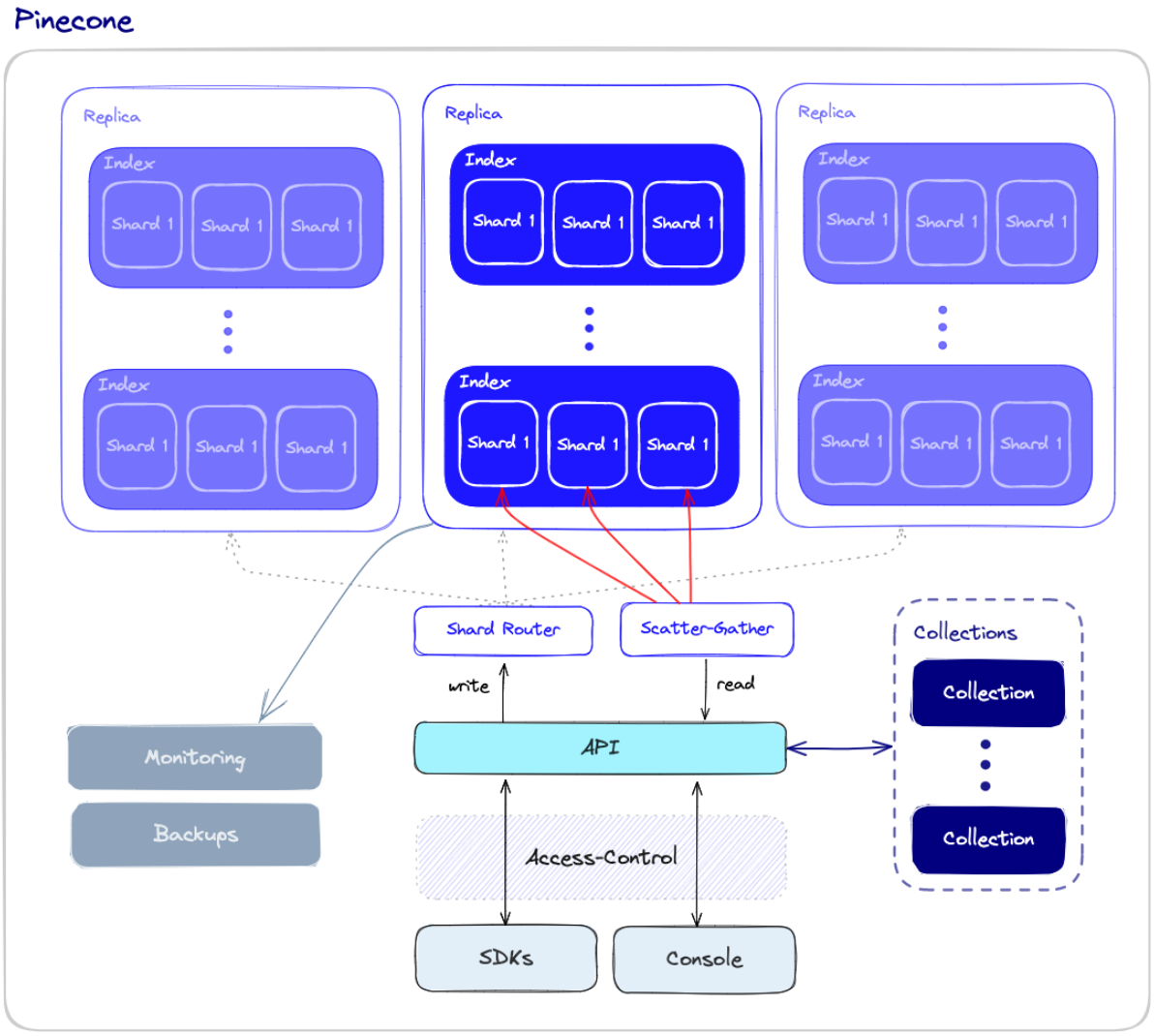
결과를 간과할 수도 있습니다. 또한 광범위한 메타데이터 필터링으로 인해 계산 오버헤드가 추가되어 쿼리 프로세스가 느려질 수 있습니다.

- **사후 필터링:** 이 접근 방식에서는 벡터 검색 후에 메타데이터 필터링이 수행됩니다. 이렇게 하면 모든 관련 결과를 고려하는 데 도움이 될 수 있지만 검색이 완료된 후 관련 없는 결과를 필터링해야 하므로 추가 오버헤드가 발생하고 쿼리 프로세스 속도가 느려질 수도 있습니다.

필터링 프로세스를 최적화하기 위해 벡터 데이터베이스는 메타데이터에 대한 고급 인덱싱 방법을 활용하거나 병렬 처리를 사용하여 필터링 작업 속도를 높이는 등 다양한 기술을 사용합니다. 벡터 데이터베이스에서 효율적이고 관련성 높은 쿼리 결과를 제공하려면 검색 성능과 필터링 정확도 간의 균형을 맞추는 것이 필수적입니다. 벡터 검색 필터링에 대해 자세히 알아보십시오.

데이터베이스 작업

벡터 인덱스와 달리 벡터 데이터베이스는 대규모 생산 설정에서 사용하기에 더 적합하도록 만드는 일련의 기능을 갖추고 있습니다. 데이터베이스 운영과 관련된 구성 요소에 대한 전반적인 개요를 살펴보겠습니다.



성능 및 내결함성

성능과 내결함성은 밀접한 관련이 있습니다. 데이터가 많을수록 더 많은 노드가 필요하고 오류 및 실패 가능성이 커집니다. 다른 유형의 데이터베이스의 경우와 마찬가지로 기본 노드 중 일부가 실패하더라도 가능한 한 빨리 쿼리가 실행되도록 해야 합니다. 이는 하드웨어 장애, 네트워크 장애 또는 기타 유형의 기술적 버그 때문일 수 있습니다. 이러한 종류의 실패는 다운타임 또는 잘못된 쿼리 결과를 초래할 수 있습니다.

고성능과 내결함성을 모두 보장하기 위해 벡터 데이터베이스는 샤딩을 사용하고 복제는 다음을 적용합니다.

1. **샤딩** - 여러 노드에 데이터를 분할합니다. 데이터를 분할하는 방법에는 여러 가지가 있습니다. 예를 들어 데이터의 서로 다른 클러스터의 유사성에 따라 분할하여 유사한 벡터가 동일한 분할에 저장되도록 할 수 있습니다. 쿼리가 생성되면 모든 샤드에 쿼리가 전송되고 결과가 검색되어 결합됩니다. 이를 "분산-수집" 패턴이라고 합니다.

2. **복제** - 서로 다른 노드에서 데이터의 여러 복사본을 생성합니다. 이렇게 하면 특정 노드가 실패하더라도 다른 노드가 이를 대체할 수 있습니다. *최종* 일관성과 *강력한* 일관성 이라는 두 가지 주요 일관성 모델이 있습니다. 최종 일관성은 데이터의 서로 다른 복사본 간에 일시적인 불일치를 허용하여 가용성을 개선하고 대기 시간을 줄이지 만 충돌 및 심지어 데이터 손실을 초래할 수 있습니다. 반면 강력한 일관성에서는 쓰기 작업이 완료된 것으로 간주되기 전에 모든 데이터 복사본이 업데이트되어야 합니다. 이 접근 방식은 더 강력한 일관성을 제공하지만 대기 시간이 길어질 수 있습니다.

모니터링

벡터 데이터베이스를 효과적으로 관리하고 유지하려면 데이터베이스의 성능, 상태 및 전반적인 상태의 중요한 측면을 추적하는 강력한 모니터링 시스템이 필요합니다. 모니터링은 잠재적인 문제를 감지하고 성능을 최적화하며 원활한 생산 운영을 보장하는 데 중요합니다. 벡터 데이터베이스 모니터링의 몇 가지 측면은 다음과 같습니다.

1. **자원 사용량** - CPU, 메모리, 디스크 공간, 네트워크 활동과 같은 자원 사용량을 모니터링하면 데이터베이스 성능에 영향을 미칠 수 있는 잠재적인 문제나 자원 제약 조건을 식별할 수 있습니다.
2. **쿼리 성능** - 쿼리 대기 시간, 처리량 및 오류율은 해결해야 할 잠재적인 시스템 문제를 나타낼 수 있습니다.
3. **시스템 상태** - 전체 시스템 상태 모니터링에는 개별 노드의 상태, 복제 프로세스 및 기타 중요 구성 요소가 포함됩니다.

액세스 제어

액세스 제어는 데이터 및 리소스에 대한 사용자 액세스를 관리하고 규제하는 프로세스입니다. 이는 데이터 보안의 중요한 구성 요소로서 승인된 사용자만 벡터 데이터베이스에 저장된 민감한 데이터를 보고, 수정하거나 상호 작용할 수 있도록 합니다.

액세스 제어는 여러 가지 이유로 중요합니다.

1. **데이터 보호:** AI 애플리케이션은 민감한 기밀 정보를 다루는 경우가 많기 때문에 엄격한 액세스 제어 메커니즘을 구현하면 무단 액세스 및 잠재적 위반으로부터 데이터를 보호하는 데 도움이 됩니다.
2. **규정 준수:** 의료 및 금융과 같은 많은 산업에서 엄격한 데이터 개인 정보 보호 규정이 적용됩니다. 적절한 액세스 제어를 구현하면 조직이 이러한 규정을 준수하고 법적 및 재정적 영향으로부터 보호할 수 있습니다.

3. **책임 및 감사:** 액세스 제어 메커니즘을 통해 조직은 벡터 데이터베이스 내에서 사용자 활동 기록을 유지할 수 있습니다. 이 정보는 감사 목적에 매우 중요하며 보안 위반이 발생하면 무단 액세스 또는 수정을 추적하는 데 도움이 됩니다.
4. **확장성 및 유연성:** 조직이 성장하고 발전함에 따라 액세스 제어 요구 사항이 변경될 수 있습니다. 강력한 액세스 제어 시스템을 통해 사용자 권한을 원활하게 수정하고 확장할 수 있으므로 조직이 성장하는 동안 데이터 보안이 그대로 유지됩니다.

백업 및 컬렉션

다른 모든 것이 실패할 경우 벡터 데이터베이스는 정기적으로 생성된 백업에 의존할 수 있는 기능을 제공합니다. 이러한 백업은 외부 스토리지 시스템 또는 클라우드 기반 스토리지 서비스에 저장하여 데이터의 안전과 복구 가능성을 보장할 수 있습니다. 데이터가 손실되거나 손상된 경우 이러한 백업을 사용하여 데이터베이스를 이전 상태로 복원하여 다운타임을 최소화하고 전체 시스템에 미치는 영향을 최소화할 수 있습니다. Pinecone 을 사용하면 사용자는 특정 색인을 백업하고 나중에 새 색인을 채우는 데 사용할 수 있는 "컬렉션"으로 저장할 수도 있습니다.

API 및 SDK

이것은 고무가 길을 만나는 곳입니다. 데이터베이스와 상호 작용하는 개발자는 친숙하고 편안한 도구 세트를 사용하여 사용하기 쉬운 API 로 데이터베이스와 상호 작용하기를 원합니다. 사용자 친화적인 인터페이스를 제공함으로써 벡터 데이터베이스 API 계층은 고성능 벡터 검색 애플리케이션의 개발을 단순화합니다.

API 외에도 벡터 데이터베이스는 종종 API 를 래핑하는 프로그래밍 언어별 SDK 를 제공합니다. SDK 를 사용하면 개발자가 애플리케이션에서 데이터베이스와 더 쉽게 상호 작용할 수 있습니다. 이를 통해 개발자는 기본 인프라 복잡성에 대해 걱정할 필요 없이 시맨틱 텍스트 검색, 생성적 질문 응답, 하이브리드 검색, 이미지 유사성 검색 또는 제품 추천과 같은 특정 사용 사례에 집중할 수 있습니다.

요약

NLP, 컴퓨터 비전 및 기타 AI 애플리케이션과 같은 분야에서 벡터 임베딩의 기하급수적인 성장으로 인해 벡터 데이터베이스가 애플리케이션의 벡터 임베딩과 효과적으로 상호 작용할 수 있게 해주는 계산 엔진으로 등장했습니다.

벡터 데이터베이스는 프로덕션 시나리오에서 벡터 임베딩을 관리할 때 발생하는 문제를 해결하기 위해 특별히 구축된 데이터베이스입니다. 이러한 이유로 기존의 스칼라 기반 데이터베이스 및 독립형 벡터 인덱스에 비해 상당한 이점을 제공합니다.

이 포스트에서 우리는 벡터 데이터베이스의 작동 방식, 사용하는 알고리즘, 프로덕션 시나리오를 위해 작동 준비가 된 추가 기능을 포함하여 벡터 데이터베이스의 주요 측면을 검토했습니다. 이것이 벡터 데이터베이스의 내부 동작을 이해하는 데 도움이 되기를 바랍니다. 운 좋게도 이것은 Pinecone 을 사용하기 위해 알아야 할 것이 아닙니다. Pinecone 은 이러한 모든 고려 사항(및 일부)을 처리하고 애플리케이션의 나머지 부분에 집중할 수 있도록 합니다.

출처: Pinecone.io site의 로이 슈바버-코헨(개발자) 자료를 인용함.