**1. En Yakın Komşu Arama Algoritmaları**

**a. k-En Yakın Komşu (k-NN) Algoritması**

**Genel Bakış**: k-NN algoritması, bir sorgu vektörüne en yakın **k** komşu vektörü bulmayı amaçlar.

* **Çalışma Prensibi**:
  + Sorgu vektörü ile indekslenmiş tüm vektörler arasındaki mesafe hesaplanır.
  + Mesafelere göre sıralama yapılarak en yakın **k** vektör seçilir.
* **Elasticsearch'te Kullanımı**:
  + Büyük veri kümelerinde doğrudan uygulanması performans açısından maliyetli olabilir.
  + Elasticsearch, bu nedenle **yaklaşık en yakın komşu (ANN)** algoritmalarını tercih eder.

**b. Yaklaşık En Yakın Komşu (ANN) Algoritmaları**

Büyük veri kümelerinde performansı artırmak için kullanılan ANN algoritmaları, küçük bir doğruluk kaybıyla aramaları hızlandırır.

**i. Hiyerarşik Gezilebilir Küçük Dünya (HNSW) Graf Algoritması**

* **Genel Bakış**:
  + HNSW, yüksek boyutlu vektör uzaylarında hızlı ve etkili en yakın komşu araması yapmak için kullanılan graf tabanlı bir algoritmadır.
* **Çalışma Prensibi**:
  + **Graf Oluşturma**:
    - Vektörler, çok katmanlı bir graf yapısında organize edilir.
    - Üst katmanlar, daha az düğüm ve bağlantı içerir; arama başlangıç noktası olarak hizmet eder.
  + **Arama Süreci**:
    - Arama, en üst katmandan başlar ve sorgu vektörüne daha yakın olan komşu düğümlere doğru ilerler.
    - Alt katmanlara indikçe arama daha hassas hale gelir ve en yakın komşular bulunur.
* **Avantajları**:
  + Büyük veri kümelerinde yüksek performans ve hız.
  + Doğruluk ve performans arasında iyi bir denge sağlar.
* **Elasticsearch'te Kullanımı**:
  + Elasticsearch'ün **k-NN Plugin** eklentisi ile HNSW desteklenir.
  + İndeks oluştururken HNSW parametreleri (m, ef\_construction) ayarlanabilir.

**2. Mesafe Metrikleri**

Vektörler arasındaki benzerliği veya mesafeyi ölçmek için kullanılan matematiksel fonksiyonlardır.

**a. Kosinüs Benzerliği (Cosine Similarity)**

* **Tanım**:
  + İki vektör arasındaki açının kosinüsünü hesaplar.
  + Vektörlerin yönlerine odaklanır, büyüklüklerini göz ardı eder.
* **Formül**:

Kosinu¨s Benzerlig˘i=A⋅B∥A∥∥B∥\text{Kosinüs Benzerliği} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}Kosinu¨s Benzerlig˘​i=∥A∥∥B∥A⋅B​

* **Özellikler**:
  + Değer aralığı: -1 ile 1 arasında.
  + 1 değeri, tam benzerliği (aynı yön) gösterir.
* **Kullanım Alanları**:
  + Metin verilerinde anlam benzerliğini ölçmek için idealdir.
* **Elasticsearch'te Kullanımı**:
  + dense\_vector alanlarında "similarity": "cosine" ayarı yapılır.
  + Sorgularda cosineSimilarity fonksiyonu kullanılır.

**b. Öklidyen Mesafe (Euclidean Distance)**

* **Tanım**:
  + İki vektör arasındaki düz çizgi mesafesini ölçer.
* **Formül**:

O¨klidyen Mesafe=∑i=1n(Ai−Bi)2\text{Öklidyen Mesafe} = \sqrt{\sum\_{i=1}^{n} (A\_i - B\_i)^2}O¨klidyen Mesafe=i=1∑n​(Ai​−Bi​)2​

* **Özellikler**:
  + Değerler her zaman pozitiftir.
  + 0 mesafesi, vektörlerin aynı olduğunu gösterir.
* **Kullanım Alanları**:
  + Vektör büyüklüklerinin önemli olduğu durumlarda kullanılır.
* **Elasticsearch'te Kullanımı**:
  + dense\_vector alanlarında "similarity": "l2\_norm" ayarı yapılır.
  + Sorgularda l2norm fonksiyonu kullanılır.

**c. Skaler Çarpım (Dot Product)**

* **Tanım**:
  + İki vektörün bileşenlerinin çarpımlarının toplamını hesaplar.
* **Formül**:

Skaler C¸arpım=∑i=1nAiBi\text{Skaler Çarpım} = \sum\_{i=1}^{n} A\_i B\_iSkaler C¸​arpım=i=1∑n​Ai​Bi​

* **Özellikler**:
  + Vektörlerin hem büyüklüklerini hem de yönlerini dikkate alır.
* **Kullanım Alanları**:
  + Öneri sistemleri ve puanlama modellerinde kullanılır.
* **Elasticsearch'te Kullanımı**:
  + dense\_vector alanlarında "similarity": "dot\_product" ayarı yapılır.
  + Sorgularda dotProduct fonksiyonu kullanılır.

**3. Elasticsearch'te Vektör Aramalarının Uygulanması**

**a. Dense Vector Alanları**

* **Tanım**:
  + Elasticsearch, yüksek boyutlu sayısal vektörleri saklamak için dense\_vector alan tipini destekler.
* **Kullanımı**:
  + Metin embedding vektörlerini veya diğer veri türlerini saklamak için kullanılır.
* **Yapılandırma**:
  + Vektör boyutu (dims) belirtilir.
  + Kullanılacak benzerlik metriği (similarity) tanımlanır.
  + İndeksleme için "index": true ayarı yapılabilir.
* **Örnek**:

json

Kodu kopyala

{

"mappings": {

"properties": {

"chunk\_text": { "type": "text" },

"chunk\_embedding\_vector": {

"type": "dense\_vector",

"dims": 768,

"index": true,

"similarity": "cosine",

"index\_options": {

"type": "hnsw",

"m": 16,

"ef\_construction": 100

}

}

}

}

}

**b. k-NN Plugin ve HNSW**

* **Amaç**:
  + Elasticsearch'ün büyük veri kümelerinde etkili k-NN aramaları yapabilmesini sağlar.
* **Özellikler**:
  + HNSW algoritması ile yaklaşık en yakın komşu araması yapar.
  + İndeksleme ve arama performansını artırır.
* **Parametreler**:
  + **İndeksleme Parametreleri**:
    - m: Her yeni eleman için oluşturulan çift yönlü bağlantı sayısı.
    - ef\_construction: İndeksleme sırasında doğruluk ve bellek tüketimini kontrol eder.
  + **Arama Parametreleri**:
    - ef\_search: Arama sırasında doğruluk ve performans dengesini ayarlar.
* **Kullanımı**:
  + Elasticsearch'e k-NN Plugin yüklenir.
  + İndeks oluştururken HNSW ayarları yapılır.
  + knn sorgusu kullanılarak arama yapılır.
* **Örnek Sorgu**:

json

Kodu kopyala

{

"size": 10,

"query": {

"knn": {

"field": "chunk\_embedding\_vector",

"query\_vector": [0.1, 0.2, ..., 0.768],

"k": 10,

"num\_candidates": 1000

}

}

}

**4. Hibrit Arama (Hybrid Search)**

**a. Tanım**

Hibrit arama, geleneksel anahtar kelime tabanlı arama yöntemlerini vektör benzerlik aramalarıyla birleştirerek hem tam eşleşmeleri hem de semantik benzerlikleri dikkate alır.

**b. Bileşenler**

* **BM25 Algoritması**:
  + Elasticsearch'ün tam metin arama için kullandığı sıralama fonksiyonudur.
  + Terim frekansı, belge uzunluğu ve ters belge frekansı gibi faktörleri kullanır.
* **Vektör Benzerlik Araması**:
  + Embedding vektörler ve benzerlik metrikleri kullanılarak yapılır.

**c. Nasıl Çalışır**

* **Sorgu Oluşturma**:
  + Hem anahtar kelime eşleşmesi hem de vektör benzerlik sorguları oluşturulur.
  + bool sorgusu ile bu sorgular birleştirilir.
* **Skorlama Mekanizması**:
  + Her iki sorgunun skorları birleştirilerek sonuçlar sıralanır.
  + Ağırlıklandırma yaparak hangi sorgunun daha etkili olacağı ayarlanabilir.
* **Örnek Sorgu**:

json

Kodu kopyala

{

"size": 10,

"query": {

"bool": {

"should": [

{

"match": {

"chunk\_text": "aranan anahtar kelime"

}

},

{

"script\_score": {

"query": { "match\_all": {} },

"script": {

"source": "cosineSimilarity(params.query\_vector, 'chunk\_embedding\_vector') + 1.0",

"params": {

"query\_vector": [0.1, 0.2, ..., 0.768]

}

}

}

}

]

}

}

}

**d. Avantajları**

* **Kapsamlı Sonuçlar**:
  + Hem tam anahtar kelime eşleşmelerini hem de semantik olarak benzer dökümanları getirir.
* **Gelişmiş İlgililik**:
  + Kullanıcının aradığı içeriği daha iyi yakalar.

**e. Dikkat Edilmesi Gerekenler**

* **Skor Dengesini Ayarlama**:
  + Hangi arama türünün daha fazla ağırlık taşıyacağı belirlenmelidir.
* **Performans**:
  + Vektör aramaları daha fazla hesaplama gücü gerektirebilir.
  + HNSW gibi indeksleme teknikleri ile performans artırılabilir.

**5. Uygulama: Anahtar Kelimenin İlgili Chunk'larda Embedding Vektörlerle Aranması**

**a. Uygulama Adımları**

1. **Embedding Vektörlerinin Oluşturulması**:
   * Metin chunk'ları ve anahtar kelimeler için embedding vektörleri oluşturun.
   * **Transformer Modelleri**: BERT, RoBERTa, Sentence Transformers gibi modeller kullanılabilir.
2. **Verilerin Elasticsearch'e İndekslenmesi**:
   * Chunk'ları ve onların embedding vektörlerini dense\_vector alanları ile indeksleyin.
   * HNSW ayarlarını yaparak indeksinizi optimize edin.
3. **Mesafe Metriklerinin Seçilmesi**:
   * **Kosinüs Benzerliği** genellikle metin embedding'leri için en uygun metriktir.
4. **Arama Sorgusunun Oluşturulması**:
   * Hibrit arama yaklaşımını kullanarak sorgu oluşturun.
   * Hem match hem de script\_score sorgularını birleştirin.
5. **Arama Parametrelerinin Optimizasyonu**:
   * HNSW parametrelerini (ef\_search) performans ve doğruluk dengesine göre ayarlayın.
   * Sorgularda ağırlıklandırma yaparak sonuçların sıralamasını etkileyin.
6. **Sonuçların İşlenmesi**:
   * Elasticsearch'ten gelen sonuçları işleyin ve kullanıcıya sunun.

**b. Dikkat Edilmesi Gerekenler**

* **Model Kalitesi**:
  + Kullandığınız embedding modellerinin kalitesi, arama sonuçlarını doğrudan etkiler.
* **Vektör Normalizasyonu**:
  + Kosinüs benzerliği kullanırken vektörlerin normalize edilmesi önemlidir.
* **İndeks Boyutu ve Performans**:
  + Büyük veri kümelerinde HNSW kullanımı performansı artırır.
* **Skalabilite**:
  + Elasticsearch kümenizi ölçeklendirerek artan veri ve sorgu yükünü yönetebilirsiniz.

**6. Sonuç ve Özet**

* **Algoritmalar**:
  + Elasticsearch, vektör aramaları için hem kesin hem de yaklaşık en yakın komşu algoritmalarını destekler.
  + HNSW, büyük ve yüksek boyutlu veri kümelerinde en etkili algoritmadır.
* **Mesafe Metrikleri**:
  + Kosinüs benzerliği, Öklidyen mesafe ve skaler çarpım temel metriklerdir.
  + Seçim, verinin doğasına ve uygulamanın gereksinimlerine bağlıdır.
* **Hibrit Arama**:
  + Geleneksel anahtar kelime araması ile vektör benzerlik aramasını birleştirir.
  + Daha kapsamlı ve ilgili arama sonuçları sağlar.
* **Uygulama Önerileri**:
  + Doğru embedding modellerini kullanın.
  + İndeks ve sorgu ayarlarını ihtiyaçlarınıza göre optimize edin.
  + Sonuçları düzenli olarak değerlendirip ayarlamalar yapın.

**1. Yaklaşık En Yakın Komşu (ANN) Algoritmalarına Genel Bakış**

**a. Neden ANN Algoritmaları?**

* **Büyük Veri Kümeleri**: Yüksek boyutlu ve büyük veri kümelerinde, geleneksel k-NN aramaları hesaplama açısından maliyetli ve yavaştır.
* **Yüksek Boyutluluk**: Veri boyutları arttıkça, arama uzayı da katlanarak büyür (Curse of Dimensionality).
* **Performans İhtiyacı**: Gerçek zamanlı uygulamalarda hızlı arama sonuçlarına ihtiyaç duyulur.

**Çözüm**: ANN algoritmaları, tam doğruluk yerine kabul edilebilir bir doğruluk seviyesinde hızlı arama sonuçları sunar. Bu, özellikle kullanıcı deneyiminin önemli olduğu uygulamalarda kritik öneme sahiptir.

**2. HNSW (Hierarchical Navigable Small World) Algoritması**

**a. Genel Bakış**

* **HNSW**, yüksek boyutlu vektör uzaylarında hızlı ve etkili yakın komşu araması yapmak için kullanılan bir **graf tabanlı ANN algoritmasıdır**.
* **Elasticsearch'te Varsayılan ANN Algoritmasıdır**: HNSW, Elasticsearch'ün **k-NN Plugin** eklentisi ile birlikte gelir ve büyük ölçekli veri kümelerinde vektör aramalarında kullanılır.

**b. HNSW'nin Çalışma Prensibi**

**i. Küçük Dünya Graf Teorisi**

* **Küçük Dünya Özelliği**: Sosyal ağlar gibi bazı graf yapılarında, herhangi iki düğüm arasındaki ortalama yol uzunluğu düşüktür.
* **Graf Yapısı**: HNSW, bu küçük dünya özelliklerine sahip bir graf oluşturur, böylece hızlı gezinme ve arama mümkün hale gelir.

**ii. Hiyerarşik Yapı**

* **Çok Katmanlı Grafikler**: HNSW, vektörleri farklı çözünürlüklerde temsil eden birden fazla katmandan oluşur.
  + **Üst Katmanlar**: Daha az düğüm ve bağlantı içerir; arama başlangıç noktasıdır.
  + **Alt Katmanlar**: Daha fazla düğüm ve bağlantı içerir; arama hassasiyeti artar.

**iii. İndeksleme Süreci**

* **Vektörlerin Eklemesi**:
  + Yeni bir vektör eklendiğinde, grafın belirli katmanlarına yerleştirilir.
  + Vektör, rastgele bir maksimum katman seviyesine atanır.
* **Bağlantıların Oluşturulması**:
  + Her katmanda, belirli sayıda yakın komşu (bağlantı) oluşturulur.
  + Bağlantı sayısı ve seçimi, parametrelerle kontrol edilir.

**iv. Arama Süreci**

1. **Başlangıç Noktası Seçimi**:
   * Arama, en üst katmandaki rastgele bir düğümden başlar.
2. **Gezinme**:
   * Sorgu vektörüne en yakın düğüm bulunana kadar komşular arasında gezinilir.
3. **Alt Katmanlara İniş**:
   * En yakın düğüm bulunduğunda bir alt katmana inilir.
   * Bu süreç en alt katmana kadar tekrarlanır.
4. **Sonuçların Toplanması**:
   * En alt katmanda, sorgu vektörüne en yakın **k** komşu bulunur.

**c. HNSW Parametreleri ve Etkileri**

**i. İndeksleme Parametreleri**

* **m (Bağlantı Sayısı)**:
  + Her düğümün sahip olacağı maksimum bağlantı sayısını belirler.
  + **Etki**:
    - **Artırılması**: Doğruluk artar, indeks boyutu ve indeksleme süresi uzar.
    - **Azaltılması**: Doğruluk düşer, indeks boyutu ve indeksleme süresi azalır.
* **ef\_construction (İnşaat Faktörü)**:
  + İndeksleme sırasında kullanılan önbellek boyutunu belirler.
  + **Etki**:
    - **Artırılması**: Daha iyi bağlantılar oluşturulur, doğruluk artar, indeksleme süresi uzar.
    - **Azaltılması**: Daha hızlı indeksleme, ancak doğruluk düşebilir.

**ii. Arama Parametresi**

* **ef\_search (Arama Faktörü)**:
  + Arama sırasında değerlendirilecek aday düğüm sayısını belirler.
  + **Etki**:
    - **Artırılması**: Doğruluk artar, arama süresi uzar.
    - **Azaltılması**: Arama hızı artar, doğruluk düşebilir.

**d. Avantajları ve Dezavantajları**

**Avantajları**

* **Hızlı Arama**: Büyük veri kümelerinde bile milisaniye seviyesinde sonuçlar.
* **Yüksek Doğruluk**: Doğruluk ve performans arasında iyi bir denge sağlar.
* **Ölçeklenebilirlik**: Milyonlarca vektör üzerinde etkili çalışır.

**Dezavantajları**

* **Hafıza Kullanımı**: İndeks boyutu artabilir, daha fazla RAM gerekebilir.
* **İndeksleme Süresi**: Yüksek doğruluk için indeksleme süresi uzayabilir.

**e. Elasticsearch'te HNSW'nin Kullanımı**

**i. İndeks Yapılandırması**

* **Dense Vector Alanı Tanımlama**:
  + Vektör boyutu (dims) belirtilir.
  + HNSW ayarları yapılır.
* **Örnek**:

json

Kodu kopyala

{

"mappings": {

"properties": {

"embedding": {

"type": "dense\_vector",

"dims": 768,

"index": true,

"similarity": "cosine",

"index\_options": {

"type": "hnsw",

"m": 16,

"ef\_construction": 100

}

}

}

}

}

**ii. Sorgu Oluşturma**

* **k-NN Sorgusu**:
  + knn sorgusu ile HNSW üzerinden arama yapılır.
* **Örnek Sorgu**:

json

Kodu kopyala

{

"size": 5,

"query": {

"knn": {

"field": "embedding",

"query\_vector": [0.1, 0.2, ..., 0.768],

"k": 5,

"num\_candidates": 100

}

}

}

* **Parametreler**:
  + **k**: Döndürülecek en yakın komşu sayısı.
  + **num\_candidates**: Arama sırasında değerlendirilecek maksimum aday sayısı (ef\_search'e denk gelir).

**3. Diğer ANN Algoritmaları ve Elasticsearch**

Elasticsearch, varsayılan olarak HNSW algoritmasını destekler ve HNSW, vektör aramalarında en etkili yöntemlerden biridir. Ancak, genel ANN literatüründe kullanılan diğer bazı algoritmalar şunlardır:

**a. Locality-Sensitive Hashing (LSH)**

* **Genel Bakış**:
  + Vektörleri, benzer olanların aynı "kovaya" düşeceği şekilde hashler.
* **Avantajları**:
  + Hızlı arama, özellikle yüksek boyutlu verilerde.
* **Dezavantajları**:
  + Elasticsearch'te doğrudan desteklenmez.
  + Doğruluk HNSW'ye kıyasla genellikle daha düşüktür.

**b. Produkt Kuantizasyonu (Product Quantization - PQ)**

* **Genel Bakış**:
  + Vektörleri alt vektörlere bölerek kuantize eder ve depolar.
* **Avantajları**:
  + Bellek verimliliği, depolama alanından tasarruf.
* **Dezavantajları**:
  + Arama doğruluğu düşebilir.
  + Elasticsearch'te doğrudan desteklenmez.

**c. Annoy (Approximate Nearest Neighbors Oh Yeah)**

* **Genel Bakış**:
  + Facebook tarafından geliştirilen, önceden inşa edilmiş ağaç yapıları kullanır.
* **Avantajları**:
  + Hızlı arama, disk tabanlı indeksleme.
* **Dezavantajları**:
  + Elasticsearch ile doğrudan entegrasyonu yoktur.
  + Bellek kullanımını optimize etmek için tasarlanmamıştır.

**d. Faiss**

* **Genel Bakış**:
  + Facebook AI tarafından geliştirilen bir kütüphane, yüksek performanslı ANN aramaları için kullanılır.
* **Avantajları**:
  + GPU desteği ile çok hızlı aramalar.
* **Dezavantajları**:
  + Elasticsearch ile doğrudan entegrasyonu yoktur.
  + Ek geliştirme ve entegrasyon çabası gerektirir.

**4. Elasticsearch'te HNSW Parametrelerinin Optimizasyonu**

**a. Doğruluk ve Performans Dengesi**

* **m ve ef\_construction**:
  + Daha yüksek değerler, daha iyi doğruluk ve daha yavaş indeksleme.
  + Daha düşük değerler, daha hızlı indeksleme ve potansiyel doğruluk kaybı.
* **ef\_search**:
  + Arama sırasında daha yüksek değerler, daha iyi doğruluk ve daha yavaş sorgular.
  + Uygulamanın ihtiyaçlarına göre ayarlanmalıdır.

**b. Pratik Öneriler**

* **Başlangıç Değerleri**:
  + m: 16 ile 48 arasında deneyin.
  + ef\_construction: 100 ile 500 arasında deneyin.
  + ef\_search: Arama sırasında k değerinin 2 ila 4 katı arasında ayarlayın.
* **Test ve Değerlendirme**:
  + Farklı parametre ayarlarıyla indeks oluşturun ve sorgular çalıştırın.
  + Doğruluk ve performansı ölçerek en uygun değerleri belirleyin.

**5. HNSW ve Mesafe Metrikleri**

**a. Desteklenen Mesafe Metrikleri**

HNSW algoritması, farklı mesafe metrikleri ile kullanılabilir:

* **Kosinüs Benzerliği**:
  + Metin embedding'leri için yaygın olarak kullanılır.
  + Elasticsearch'te "similarity": "cosine" olarak ayarlanır.
* **Öklidyen Mesafe (L2 Normu)**:
  + Görüntü ve ses verileri gibi diğer veri türleri için kullanılabilir.
  + Elasticsearch'te "similarity": "l2\_norm" olarak ayarlanır.
* **Skaler Çarpım (Dot Product)**:
  + Öneri sistemlerinde kullanılabilir.
  + Elasticsearch'te "similarity": "dot\_product" olarak ayarlanır.

**b. Seçim Kriterleri**

* **Veri Türü ve Uygulama**:
  + Metin verileri için genellikle kosinüs benzerliği tercih edilir.
  + Görüntü verileri için öklidyen mesafe uygun olabilir.
* **Performans ve Doğruluk**:
  + Bazı metrikler hesaplama açısından daha maliyetli olabilir.
  + Doğruluk ihtiyaçlarınıza göre seçim yapın.

**6. HNSW'nin Uygulama Örnekleri**

**a. Metin Benzerliği ve Semantik Arama**

* **Amaç**:
  + Kullanıcının girdiği bir sorgu metninin embedding vektörünü kullanarak, büyük bir metin kümesinde semantik olarak benzer metinleri bulmak.
* **Uygulama**:
  + Metinler için embedding vektörleri oluşturulur (örneğin, BERT ile).
  + Elasticsearch'e HNSW yapılandırmasıyla indekslenir.
  + Sorgu sırasında, kullanıcı metninin embedding vektörü hesaplanır ve HNSW ile arama yapılır.

**b. Görüntü Benzerliği**

* **Amaç**:
  + Bir görüntünün embedding vektörünü kullanarak, benzer görüntüleri bulmak.
* **Uygulama**:
  + Görüntüler için özellik çıkarımı yapılarak embedding vektörleri oluşturulur (örneğin, CNN'lerle).
  + Elasticsearch'e indekslenir.
  + HNSW ve uygun mesafe metriği (örneğin, öklidyen mesafe) kullanılarak arama yapılır.

**c. Öneri Sistemleri**

* **Amaç**:
  + Kullanıcıların beğenilerine göre ürün veya içerik önerileri sunmak.
* **Uygulama**:
  + Kullanıcı ve ürün embedding'leri oluşturulur.
  + Skaler çarpım metriği kullanılarak HNSW ile arama yapılır.
  + En yüksek skorları alan ürünler önerilir.

**7. HNSW'nin Limitasyonları ve Alternatifler**

**a. Limitasyonlar**

* **Hafıza Kullanımı**:
  + HNSW indeksleri bellek içinde tutulur; büyük veri kümelerinde RAM ihtiyacı artabilir.
* **Dinamik Veri Güncellemeleri**:
  + İndekse sık sık veri ekleme ve silme işlemleri performansı etkileyebilir.
* **Paralelleştirme**:
  + HNSW'nin paralel indeksleme ve arama yetenekleri sınırlıdır.

**b. Alternatif Yaklaşımlar**

* **Disk Tabanlı ANN Algoritmaları**:
  + Bellek sınırlamalarını aşmak için disk tabanlı indeksleme kullanılır.
  + Örneğin, DiskANN gibi algoritmalar.
* **Cluster Tabanlı Yaklaşımlar**:
  + Veri kümelerini bölerek paralel arama ve indeksleme yapılabilir.
  + Elasticsearch'te, indeksleri birden fazla shard'a bölerek ölçeklendirme yapılabilir.

**8. Elasticsearch ve ANN Algoritmalarının Geleceği**

* **Yeni Algoritmaların Entegrasyonu**:
  + Elasticsearch topluluğu ve geliştiricileri, yeni ve daha verimli ANN algoritmalarını entegre edebilir.
* **GPU Desteği**:
  + GPU hızlandırmalı arama ve indeksleme, performansı artırabilir.
* **Makine Öğrenimi ile Entegrasyon**:
  + Elasticsearch'ün ML özellikleri genişletilerek daha akıllı arama ve analizler yapılabilir.

**9. Sonuç ve Özet**

* **HNSW, Elasticsearch'te ANN Aramaları için Anahtar Bir Algoritmadır**:
  + Büyük ve yüksek boyutlu veri kümelerinde hızlı ve etkili arama sağlar.
* **Parametrelerin Optimizasyonu Kritik Öneme Sahiptir**:
  + Doğruluk ve performans ihtiyaçlarınıza göre m, ef\_construction ve ef\_search değerlerini ayarlayın.
* **Mesafe Metriklerinin Seçimi Uygulamaya Bağlıdır**:
  + Verinizin doğasına ve uygulamanın gereksinimlerine göre en uygun metriği seçin.
* **Alternatif Algoritmalar ve Yaklaşımlar Mevcuttur**:
  + HNSW'nin limitasyonları varsa, diğer ANN algoritmalarını veya ölçeklendirme stratejilerini değerlendirin.

Elasticsearch ile embedding vektörler üzerinden arama yapmak istediğinizde, **yakın komşu (Nearest Neighbor) arama algoritmalarını** kullanmanız gerekir. Bu tür aramalar için Elasticsearch, özellikle **HNSW (Hierarchical Navigable Small World)** graf algoritmasını destekler. Ayrıca, arama sırasında kullanabileceğiniz farklı mesafe metrikleri vardır. İşte detaylı açıklamalar ve kullanabileceğiniz yapılar:

**Elasticsearch'te Kullanılan Algoritmalar**

**1. HNSW (Hierarchical Navigable Small World) Algoritması**

* **HNSW**, yüksek boyutlu vektör uzaylarında hızlı ve etkili yakın komşu aramaları yapmak için kullanılan bir graf tabanlı algoritmadır.
* Elasticsearch’ün **k-NN (k-Nearest Neighbors) Plugin**'i ile birlikte gelir ve büyük ölçekli veri kümelerinde embedding vektör aramaları için idealdir.
* **Avantajları**:
  + Büyük veri kümelerinde yüksek performans ve hız sağlar.
  + Yüksek doğrulukta sonuçlar üretir.
* **Nasıl Kullanılır**:
  + İndeksinizdeki vektör alanlarını dense\_vector tipiyle tanımlayın ve HNSW ayarlarını yapılandırın.
  + Arama sorgularınızda HNSW algoritmasını kullanarak hızlı ve doğru sonuçlar elde edin.

**Elasticsearch'te Kullanılabilecek Mesafe Metrikleri**

Elasticsearch, vektörler arasındaki benzerliği veya mesafeyi hesaplamak için çeşitli metrikleri destekler:

**1. Cosine Similarity (Kosinüs Benzerliği)**

* **Tanım**: İki vektör arasındaki açıyı ölçerek benzerlik derecesini belirler.
* **Değer Aralığı**: -1 ile 1 arasında; 1’e yakın değerler yüksek benzerliği gösterir.
* **Kullanım Durumları**:
  + Metin verilerinde anlam benzerliğini ölçmek için idealdir.
  + Vektörlerin yönlerine odaklandığı için büyüklüklerinden bağımsızdır.
* **Elasticsearch'te Kullanımı**:
  + dense\_vector alanlarında similarity: "cosine" ayarı yapılır.
  + Sorgu sırasında cosineSimilarity fonksiyonu kullanılır.

**2. Euclidean Distance (Öklidyen Mesafe)**

* **Tanım**: İki vektör arasındaki düz çizgi (doğrusal) mesafeyi hesaplar.
* **Değer Aralığı**: 0’dan sonsuza kadar; 0 mesafe tam benzerliği gösterir.
* **Kullanım Durumları**:
  + Vektörlerin gerçek değerli koordinatlara sahip olduğu durumlarda kullanılır.
  + Büyüklük farklarının önemli olduğu durumlar için uygundur.
* **Elasticsearch'te Kullanımı**:
  + dense\_vector alanlarında similarity: "l2\_norm" ayarı yapılır.
  + Sorgu sırasında l2norm fonksiyonu kullanılır.

**3. Dot Product (Skaler Çarpım)**

* **Tanım**: İki vektörün bileşenlerinin çarpımlarının toplamını hesaplar.
* **Kullanım Durumları**:
  + Vektörlerin büyüklüklerinin ve yönlerinin önem taşıdığı durumlarda kullanılır.
  + Öneri sistemleri ve puanlama modellerinde yaygındır.
* **Elasticsearch'te Kullanımı**:
  + dense\_vector alanlarında similarity: "dot\_product" ayarı yapılır.
  + Sorgu sırasında dotProduct fonksiyonu kullanılır.

**Örnek Kullanım Durumu: Anahtar Kelimenin İlgili Chunk'larda Embedding Vektörlerle Aranması**

Bu senaryoda, bir anahtar kelimenin metin parçaları (chunk'lar) içinde semantik olarak eşleşenlerini bulmak istiyorsunuz. İşte bu görev için kullanabileceğiniz yapılar ve adımlar:

**1. Metinlerin Embedding Vektörlerini Oluşturma**

* **Transformer Modelleri**: BERT, RoBERTa veya Sentence-BERT gibi modeller kullanarak metinlerin embedding vektörlerini oluşturabilirsiniz.
* **Adımlar**:
  + Anahtar kelimenizi ve metin chunk'larınızı embedding vektörlerine dönüştürün.
  + Bu vektörler, metinlerin anlamını çok boyutlu bir uzayda temsil eder.

**2. Elasticsearch'te Vektör Alanlarını Yapılandırma**

* **İndeks Oluşturma**:
  + dense\_vector tipiyle bir alan (örneğin, chunk\_embedding\_vector) tanımlayın.
  + HNSW ayarlarını yaparak indeksinizi performans için optimize edin.
* **Örnek Ayarlar**:

json

Kodu kopyala

{

"mappings": {

"properties": {

"chunk\_text": { "type": "text" },

"chunk\_embedding\_vector": {

"type": "dense\_vector",

"dims": 768,

"index": true,

"similarity": "cosine",

"index\_options": {

"type": "hnsw",

"m": 16,

"ef\_construction": 100

}

}

}

}

}

**3. Vektörleri Elasticsearch'e İndeksleme**

* Metin chunk'larınızı ve onların embedding vektörlerini Elasticsearch indeksinize yükleyin.
* Her döküman şu şekilde olabilir:

json

Kodu kopyala

{

"chunk\_text": "Metin parçası burada",

"chunk\_embedding\_vector": [0.123, 0.456, ..., 0.789]

}

**4. HNSW ile Dense Vector Araması Yapma**

* **Arama Sorgusu**:
  + Anahtar kelimenizin embedding vektörünü sorgu sırasında hesaplayın.
  + HNSW algoritmasını kullanarak en yakın komşu araması yapın.
* **Sorgu Örneği**:

json

Kodu kopyala

{

"size": 5,

"query": {

"script\_score": {

"query": { "match\_all": {} },

"script": {

"source": "cosineSimilarity(params.query\_vector, 'chunk\_embedding\_vector') + 1.0",

"params": { "query\_vector": [0.123, 0.456, ..., 0.789] }

}

}

}

}

* **Açıklama**:
  + cosineSimilarity fonksiyonu ile sorgu vektörü ve döküman vektörü arasındaki benzerlik hesaplanır.
  + +1.0 ekleyerek skorun pozitif olması ve Elasticsearch'ün skor sistemine uygun hale getirilmesi sağlanır.

**5. Sonuçları İşleme ve Sıralama**

* **Skorlama**:
  + Elasticsearch, sorgu sonuçlarını skorlarına göre otomatik olarak sıralar.
  + En yüksek skorlu dökümanlar, anahtar kelimenizle en semantik olarak ilişkili chunk'ları temsil eder.
* **Sonuçların Kullanımı**:
  + Elde edilen chunk'ları kullanıcıya sunabilir veya başka işlemlerde kullanabilirsiniz.

**Özetle Kullanmanız Gereken Yapılar**

* **Embedding Vektörleri**: Anahtar kelimeniz ve metin chunk'larınız için oluşturulmuş vektörler.
* **Elasticsearch k-NN Plugin ve HNSW Algoritması**: Büyük veri kümelerinde hızlı ve doğru arama için.
* **Cosine Similarity**: Metinlerin anlam benzerliğini ölçmek için uygun mesafe metriği.
* **Dense Vector Alanları**: Vektörleri indekslemek ve aramak için dense\_vector tipiyle tanımlanmış alanlar.
* **Sorgu Yapısı**: script\_score ve cosineSimilarity kullanarak arama sorguları oluşturma.

**Ek Öneriler**

* **Performans Optimizasyonu**:
  + HNSW parametrelerini (m, ef\_construction, ef\_search) ihtiyaçlarınıza göre ayarlayarak arama performansını optimize edebilirsiniz.
* **Hybrid Arama Yaklaşımı**:
  + Hem keyword araması (BM25) hem de vektör aramasını birleştirerek daha kapsamlı sonuçlar elde edebilirsiniz.
  + bool sorguları ile should klauzu kullanarak iki arama türünü birleştirebilirsiniz.

**Sonuç**

Elasticsearch'te embedding vektörler üzerinden arama yapmak için HNSW algoritması ve cosine similarity metriğini kullanmanız en doğru yaklaşım olacaktır. Bu yapı, özellikle büyük veri kümelerinde yüksek performans ve doğruluk sağlar. Anahtar kelimenizin embedding vektörünü kullanarak, metin chunk'ları arasında semantik olarak en ilgili olanları hızlı ve etkili bir şekilde bulabilirsiniz.,

**KNN ve ANN Algoritmalarının Özellikleri ve Anahtar Kelime Kullanım Durumuna Göre Açıklaması**

**Giriş**

Elasticsearch ve genel veri arama sistemlerinde, **K En Yakın Komşu (KNN)** ve **Yaklaşık En Yakın Komşu (ANN)** algoritmaları, özellikle vektör tabanlı aramalarda önemli bir rol oynar. Bu algoritmalar, bir sorgu vektörüne en yakın veri noktalarını bulmak için kullanılır. Anahtar kelime kullanım durumunda, bu algoritmalar metinlerin veya belgelerin embedding (gömme) vektörleri üzerinden semantik benzerliği ölçmek için kullanılabilir.

Aşağıda KNN ve ANN algoritmalarının özelliklerini, nasıl çalıştıklarını ve anahtar kelime kullanım durumuna nasıl uygulandıklarını detaylı bir şekilde açıklayacağım.

**1. KNN (K En Yakın Komşu) Algoritması**

**Özellikleri:**

* **Tanım:** KNN algoritması, bir sorgu noktasına en yakın **k** veri noktalarını bulmayı amaçlayan bir sınıflandırma ve regresyon algoritmasıdır.
* **Çalışma Prensibi:**
  + **Mesafe Hesaplama:** Sorgu vektörü ile veri kümesindeki her vektör arasındaki mesafe hesaplanır.
  + **Sıralama ve Seçim:** Mesafeler sıralanır ve en küçük mesafeye sahip **k** veri noktası seçilir.
* **Mesafe Metrikleri:** Genellikle **Öklidyen mesafe**, **Manhattan mesafesi**, **Minkowski mesafesi** gibi metrikler kullanılır.
* **Avantajları:**
  + **Basitlik:** Anlaşılması ve uygulanması kolaydır.
  + **Doğruluk:** Küçük veri kümelerinde yüksek doğruluk sağlar.
* **Dezavantajları:**
  + **Performans:** Büyük veri kümelerinde hesaplama maliyeti yüksektir, çünkü her aramada tüm veri kümesi taranır.
  + **Bellek Kullanımı:** Tüm veri kümesinin bellekte tutulması gerekebilir.

**Anahtar Kelime Kullanım Durumuna Uygulama:**

* **Embedding Vektörleri ile Arama:**
  + **Sorgu Vektörü Oluşturma:** Anahtar kelimenin embedding vektörü hesaplanır.
  + **Mesafe Hesaplama:** Veri kümesindeki her belgenin veya metin parçasının embedding vektörü ile sorgu vektörü arasındaki mesafe hesaplanır.
  + **En Yakın Komşuların Seçimi:** En yakın **k** komşu seçilir; bunlar semantik olarak anahtar kelimeye en yakın metinlerdir.
* **Özellikler:**
  + **Doğruluk:** Küçük veri kümelerinde yüksek doğruluk sağlar.
  + **Performans Sorunları:** Büyük veri kümelerinde arama süresi uzun olabilir, çünkü tüm vektörler karşılaştırılır.

**2. ANN (Yaklaşık En Yakın Komşu) Algoritmaları**

**Özellikleri:**

* **Tanım:** ANN algoritmaları, KNN'nin büyük veri kümelerindeki performans sorunlarını çözmek için geliştirilmiştir. Tam doğruluk yerine, kabul edilebilir bir doğruluk seviyesinde çok daha hızlı arama sonuçları üretirler.
* **Çalışma Prensibi:**
  + **Veri Yapıları Kullanımı:** Veri kümesini belirli veri yapıları veya indeksler kullanarak organize ederler.
  + **Hızlı Arama:** Arama sırasında tüm veri kümesini taramak yerine bu yapıları kullanarak hızlıca en yakın komşuları bulurlar.
* **Yaygın ANN Algoritmaları:**
  + **HNSW (Hierarchical Navigable Small World):**
    - **Graf Tabanlı Yaklaşım:** Veri noktalarını çok katmanlı bir graf yapısında organize eder.
    - **Hızlı Gezinme:** Küçük dünya graf teorisi prensiplerine dayanarak hızlı arama sağlar.
  + **LSH (Locality-Sensitive Hashing):**
    - **Hash Tabanlı Yaklaşım:** Benzer vektörleri aynı hash kovasına yerleştirir.
  + **Product Quantization (PQ):**
    - **Kuantizasyon:** Vektörleri kuantize ederek depolama ve arama maliyetini azaltır.
* **Avantajları:**
  + **Performans:** Büyük veri kümelerinde çok hızlı arama sağlar.
  + **Ölçeklenebilirlik:** Milyonlarca vektör üzerinde etkili çalışır.
* **Dezavantajları:**
  + **Doğruluk:** KNN'ye göre biraz daha düşük doğruluk seviyeleri olabilir.
  + **Karmaşıklık:** Uygulanması ve optimize edilmesi daha karmaşık olabilir.

**Anahtar Kelime Kullanım Durumuna Uygulama:**

* **Embedding Vektörleri ile Hızlı Semantik Arama:**
  + **Sorgu Vektörü Oluşturma:** Anahtar kelimenin embedding vektörü hesaplanır.
  + **Veri Yapısı Oluşturma:** Veri kümesindeki metinlerin embedding vektörleri, önceden bir ANN yapısı (örneğin, HNSW grafı) kullanılarak organize edilir.
  + **Hızlı Arama:** Arama sırasında bu yapı kullanılarak sorgu vektörüne en yakın komşular hızlıca bulunur.
* **Özellikler:**
  + **Hız:** Büyük veri kümelerinde bile milisaniye seviyesinde arama sonuçları elde edilir.
  + **Doğruluk:** KNN'ye yakın veya uygulamaya göre ayarlanabilir doğruluk seviyeleri.
* **Elasticsearch Örneği:**
  + **k-NN Plugin ve HNSW:** Elasticsearch'te k-NN Plugin ile HNSW algoritması kullanılır.
  + **Vektör Alanları:** Embedding vektörleri dense\_vector tipiyle tanımlanır ve HNSW parametreleri (örneğin, m, ef\_construction) ayarlanır.
  + **Arama Sorguları:** Arama sırasında script\_score veya knn sorguları kullanılarak hızlı ve etkili arama yapılır.

**3. KNN ve ANN Algoritmalarının Karşılaştırılması**

| **Özellik** | **KNN** | **ANN** |
| --- | --- | --- |
| **Doğruluk** | Yüksek (Tam Doğruluk) | Yüksek (Kabul Edilebilir Doğruluk) |
| **Performans** | Yavaş (Büyük Veri Kümelerinde) | Hızlı (Büyük Veri Kümelerinde Etkili) |
| **Hesaplama** | Tüm Veri Kümesi Tarama | Özel Veri Yapıları ile Hızlandırılmış Arama |
| **Bellek Kullanımı** | Yüksek (Tüm Veri Kümesi Bellekte) | Optimizasyonlarla Daha Verimli Bellek Kullanımı |
| **Uygulama Zorluğu** | Basit Uygulanabilir | Daha Karmaşık Yapılandırma ve Parametre Ayarı Gerektirir |

**4. Anahtar Kelime Kullanım Durumunda Hangi Algoritma Tercih Edilmeli?**

* **Küçük Veri Kümeleri İçin:**
  + **KNN Algoritması** yeterli olabilir.
  + **Basit Uygulama:** Karmaşık yapılandırma gerektirmez.
  + **Yüksek Doğruluk:** Tüm veri kümesini taradığı için en yakın komşuları tam olarak bulur.
* **Büyük Veri Kümeleri İçin:**
  + **ANN Algoritmaları** tercih edilmelidir.
  + **Performans:** Büyük veri kümelerinde hızlı arama sağlar.
  + **Ölçeklenebilirlik:** Veri büyüklüğü arttıkça performans avantajı daha belirgin hale gelir.
  + **Elasticsearch Kullanımı:** HNSW gibi algoritmalarla optimize edilmiş arama yapısı oluşturulabilir.

**5. Sonuç ve Özet**

* **KNN ve ANN algoritmaları**, vektör tabanlı aramalarda kullanılan temel yaklaşımlardır.
* **Anahtar Kelime Kullanım Durumunda**, metinlerin embedding vektörleri üzerinden semantik benzerlik aramaları yapmak için bu algoritmalar kullanılır.
* **KNN**, basit ve yüksek doğruluklu ancak büyük veri kümelerinde yavaş ve hesaplama maliyeti yüksektir.
* **ANN**, büyük veri kümelerinde hızlı ve ölçeklenebilir bir çözüm sunar, ancak biraz daha düşük doğruluk seviyeleri olabilir.
* **Elasticsearch'te**, ANN algoritmaları (özellikle HNSW) kullanılarak embedding vektörler üzerinden hızlı ve etkili aramalar yapılabilir.

**Öneriler:**

* **Veri Kümeniz Büyükse**, Elasticsearch ve HNSW gibi ANN algoritmalarını kullanarak performansı optimize edin.
* **Doğruluk ve Performans Dengesini Ayarlayın**, ANN algoritmalarının parametrelerini (örneğin, m, ef\_construction, ef\_search) uygulamanızın ihtiyaçlarına göre ayarlayarak en iyi sonuçları elde edebilirsiniz.
* **Hibrit Arama Yaklaşımlarını Düşünün**, hem geleneksel anahtar kelime araması hem de vektör tabanlı semantik aramayı birleştirerek daha kapsamlı ve ilgili sonuçlar elde edebilirsiniz.

Bu açıklamalarla, KNN ve ANN algoritmalarının özelliklerini ve anahtar kelime kullanım durumunda nasıl uygulandıklarını detaylı bir şekilde öğrenmiş oldunuz. Uygulamanızın gereksinimlerine göre en uygun algoritmayı seçerek, arama performansınızı ve doğruluğunuzu optimize edebilirsiniz.