Engineer

Vektorsuche

Einträge mit dem kleinsten Vektorabstand zur Anfrage finden

Was es ist

Vektorsuche findet die gespeicherten Einträge, deren Vektoren dem Vektor einer Suchanfrage am nächsten liegen. Sie heißt auch Vektorähnlichkeitssuche. Ein Vektor ist hier eine Liste fester Länge aus Zahlen, die ein Embedding-Modell aus einem Textstück, einem Bild oder einem Produktdatensatz erzeugt. Das Modell wird so trainiert, dass ähnliche Inhalte benachbarte Vektoren erhalten. Eine Frage und die Textpassage mit ihrer Antwort können deshalb nahe beieinanderliegen, selbst wenn sie keine Wörter gemeinsam haben.

„Am nächsten“ braucht ein Maß. pgvector, die PostgreSQL-Erweiterung für die Beispiele dieses Artikels, unterstützt sechs: L2-Abstand, also euklidischen Abstand, Skalarprodukt, Kosinusabstand und L1-Abstand sowie Hamming- und Jaccard-Abstand für binäre Vektoren. Die Dokumentation des Embedding-Modells sagt, für welches Maß seine Vektoren gedacht sind. Bei auf Länge 1 normalisierten Vektoren bringen Kosinusabstand und Skalarprodukt die Nachbarn in dieselbe Reihenfolge. Für die beste Leistung empfiehlt pgvector dort das Skalarprodukt.

Die Frage lässt sich auf zwei Arten beantworten. Exakte Suche vergleicht die Anfrage mit jedem gespeicherten Vektor und liefert die tatsächlichen nächsten Nachbarn. pgvector tut das standardmäßig; seine Dokumentation nennt es perfekte Trefferabdeckung, also perfekten Recall. Näherungsweise Suche kann einen Index verwenden, der nur einen Teil der Daten besucht, und tauscht etwas Recall gegen Geschwindigkeit. Wie viele tatsächliche Nachbarn zurückkommen, hängt von den Daten, dem Index und seinen Einstellungen ab; nicht einmal eine Mehrheit ist garantiert. Deshalb warnt pgvector davor, dass ein näherungsweiser Index anders als ein gewöhnlicher Index die Ergebnisse einer Abfrage verändert.

pgvector bietet zwei Arten näherungsweiser Indizes. HNSW baut einen Graphen mit mehreren Ebenen auf, ein von Malkov und Yashunin erstmals 2016 veröffentlichtes Verfahren. Jede Ebene enthält eine Teilmenge der darunterliegenden. Die Suche beginnt auf der kleinen obersten Ebene und arbeitet sich bis zur vollständigen hinunter; die Veröffentlichung berichtet von logarithmisch wachsenden Suchkosten. IVFFlat teilt die Vektoren in Listen auf und durchsucht nur die Listen, die der Anfrage am nächsten liegen.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist Vektorsuche?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Ihre Datensätze sind für KI-Agenten erreichbar, jeder Kunde ist nur einmal erfasst, niemand sieht mehr als erlaubt und jede Antwort lässt sich bis zur Quelle zurückverfolgen.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird.
Geografische Datenpipeline 50-mal schneller — 50x schnellere Geodaten-Pipelines

Geografische Datenpipeline 50-mal schneller Kontakt aufnehmen →

Warum es wichtig ist

Volltextsuche gleicht Wörter ab. Mit Wortabgleich und Wortstammreduktion allein findet ein Kunde, der „der Bildschirm bleibt schwarz“ schreibt, nicht den Handbuchabschnitt „das Display schaltet sich nicht ein“, weil beide keinen gemeinsamen Wortstamm haben. Konfigurierte Synonyme können einige solche Lücken überbrücken. Vektorsuche vergleicht die Bedeutung, wie das Embedding-Modell sie erfasst hat, sodass Beschwerde und Abschnitt nahe beieinanderliegen können. Dieselbe Idee funktioniert über Text hinaus: ein ähnliches Bild, eine ähnliche bereits gelöste Supportanfrage, ein Produkt wie das im Warenkorb.

Sie ist auch eine Möglichkeit, die Suche bei Retrieval-Augmented Generation auszuführen. Dabei antwortet ein großes Sprachmodell anhand von Textpassagen, die zur Frage abgerufen wurden. Die Frage wird mit demselben Modell wie die Passagen in einen Vektor umgewandelt; die nächsten Passagen kommen in den Prompt. Die Antwort kann nur so gut sein wie das Gefundene. Die Suche verdient deshalb dieselbe Sorgfalt wie das Modell.

Die Vektoren in der bereits betriebenen Datenbank zu halten hat praktische Vorteile. Mit diesem Argument beginnt auch die pgvector-Dokumentation. Die Vektoren erhalten dieselben Transaktionen, zeitpunktgenauen Wiederherstellungen und Joins wie jede andere Spalte. Weil pgvector in das

Write-Ahead-Log schreibt, werden sie auch repliziert. Ein Filter nach Kunde, Datum oder Berechtigung ist eine gewöhnliche `WHERE` -Bedingung auf derselben Zeile. Ein Snapshot nach dem Commit einer Löschung sieht das Dokument nicht mehr; eine Abfrage mit einem älteren Snapshot kann es dagegen noch sehen. Ein gesonderter Vektorspeicher ist ein zweites System, das gefüllt und mit dem ersten synchron gehalten werden muss.

Ein Team sollte auch wissen, was Vektorsuche nicht leistet. Sie ordnet nach Nähe, nicht nach Wahrheit. Die nächste Passage ähnelt der Frage am meisten; sie kann trotzdem veraltet oder falsch sein oder aus einem Dokument stammen, das der Leser nicht sehen darf. Manche Fragen hängen von einem bestimmten Wort ab, etwa einem Namen, einer Artikelnummer oder einem Fehlercode. Die pgvector-Dokumentation empfiehlt deshalb, Vektorsuche mit der Volltextsuche von PostgreSQL zu verbinden. Diese Kombination heißt hybride Suche.

Wie es in der Praxis aussieht

pgvector unterstützt in der Version 0.8.7, Stand Oktober 2026, PostgreSQL 13 und neuer. Die Erweiterung wird einmal pro Datenbank aktiviert. Danach erhält die Tabelle eine Spalte vom Typ `vector` mit der vom Modell erzeugten Dimensionszahl:

```
CREATE EXTENSION vector;

CREATE TABLE passages (
    id        bigserial PRIMARY KEY,
    source    text NOT NULL,
    body      text NOT NULL,
    embedding vector(3)
);

CREATE INDEX ON passages USING hnsw (embedding vector_cosine_ops);
```

Drei Dimensionen halten das Beispiel lesbar; ein echtes Modell erzeugt deutlich mehr, und die Spalte deklariert dessen Dimensionszahl. Die Anwendung berechnet jedes Embedding mit dem Modell und fügt es neben dem ursprünglichen Text ein. Der Index benennt ein Abstandsmaß: `vector_cosine_ops` bedient nach Kosinusabstand sortierte Abfragen. Eine nach L2-Abstand sortierte Abfrage braucht einen zweiten Index mit `vector_l2_ops` . HNSW hat keinen Trainingsschritt; der Index kann deshalb auf einer leeren Tabelle angelegt werden.

Die Abfrage erzeugt das Embedding der Frage mit demselben Modell und fordert die nächsten Passagen an:

```
SELECT id, source, body
FROM   passages
ORDER  BY embedding <=> $1
LIMIT  5;
```

`<=>` ist der Kosinusabstand, `$1` der als Parameter übergebene Anfragevektor. Zwei Details ermöglichen dem Planer, den Index zu verwenden: Die Abfrage hat sowohl `ORDER BY` als auch `LIMIT`, und `ORDER BY` verwendet den reinen Abstandsoperator in aufsteigender Reihenfolge. Ein anzuzeigender Ähnlichkeitswert wie `1 - (embedding <=> $1)` gehört in die Auswahlliste. Die übrigen Operatoren sind `<->` für L2-Abstand, `<+>` für L1 und `<#>` für das Skalarprodukt. Letzterer liefert das negative Skalarprodukt, weil PostgreSQL einen Index über einen Operator nur in aufsteigender Reihenfolge durchsucht.

Der HNSW-Index hat zwei Aufbauparameter: `m`, die Zahl der Nachbarn, mit denen sich ein Knoten auf jeder Ebene verbindet, ohne andere Einstellung 16, und `ef_construction`, die Größe der Kandidatenliste beim Aufbau, voreingestellt 64. Bei einer Abfrage bestimmt `hnsw.ef_search` die Größe der Kandidatenliste, voreingestellt 40. Ein höherer Wert findet mehr tatsächliche nächste Nachbarn und verlangsamt jede Abfrage. `SET LOCAL` erhöht ihn bis zum Ende der aktuellen Transaktion. Soll die Einstellung nur für eine Abfrage gelten, darf die Transaktion nur diese Abfrage enthalten. Um die Recall-Kosten des Index zu ermitteln, wird dieselbe Abfrage in einer Transaktion mit `SET LOCAL enable_indexscan = off` ausgeführt. Das liefert das exakte Ergebnis, mit dem die andere Liste verglichen werden kann.

Die Arbeit des Unternehmens mit maßgeschneiderten KI-Agenten speichert die Erinnerung eines Agenten an frühere Fälle in der eigenen Datenbank des Kunden statt im Chatverlauf eines Anbieters. Eine Tabelle wie `passages` ist eine mögliche Form dieser Erinnerung: Bei einem neuen Fall sind die ähnlichsten früheren Fälle nur eine Abfrage entfernt, in derselben Datenbank und mit denselben Berechtigungen wie die übrigen Daten.

Hybride Suche führt beide Suchen aus und verbindet die beiden Listen. Die pgvector-Dokumentation nennt dafür Reciprocal Rank Fusion aus einer Veröffentlichung von Cormack, Clarke und Büttcher aus dem Jahr 2009: Jedes

Dokument erhält für jede Liste, in der es vorkommt, den Wert 1 / (k + rank), und die Werte werden addiert. Die Autoren legten k auf 60 fest. Ihre Vorversuche zeigten, dass 60 „nahezu optimal war, die Wahl aber nicht entscheidend“. Mit einer `tsvector`-Spalte `search_vec` in derselben Tabelle, aufgebaut wie für die Volltextsuche, lautet die zusammengeführte Abfrage:

```
WITH semantic AS (
    SELECT id, row_number() OVER (ORDER BY dist) AS r
    FROM  (SELECT id, embedding <=> $1 AS dist
           FROM   passages
           ORDER  BY dist
           LIMIT  20) AS nearest
),
keyword AS (
    SELECT id, row_number() OVER (ORDER BY score DESC) AS r
    FROM  (SELECT id, ts_rank_cd(search_vec, query) AS score
           FROM   passages, plainto_tsquery('english', $2) AS query
           WHERE  search_vec @@ query
           ORDER  BY score DESC
           LIMIT  20) AS matches
)
SELECT id,
       coalesce(1.0 / (60 + s.r), 0)
     + coalesce(1.0 / (60 + k.r), 0) AS rrf
FROM   semantic AS s
FULL   JOIN keyword AS k USING (id)
ORDER  BY rrf DESC
LIMIT  5;
```

Jede Suche behält ihre eigenen besten 20 Treffer und nummeriert sie ab 1. Eine Passage, die in beiden Suchen weit oben steht, erhält zwei Anteile und steigt nach oben; eine nur von einer Suche gefundene behält einen Anteil. Die Zusammenführung nutzt Ränge statt Bewertungen, weil Kosinusabstand und Textrang auf nicht vergleichbaren Skalen liegen. Als weitere Möglichkeit, die Listen zu verbinden, nennt pgvector einen Cross-Encoder.

Wo es schiefgeht

Weniger Zeilen als von `LIMIT` angefordert. Bei einem näherungsweisen Index wird die `WHERE`-Bedingung erst nach dem Durchsuchen des Index angewendet. pgvector nennt das Rechenbeispiel: Trifft eine Bedingung auf 10 % der Zeilen zu, bleiben bei einem HNSW-Index mit der Vorgabe 40 für `hnsw.ef_search` durchschnittlich etwa 4 Zeilen übrig. Seit Version 0.8.0 durchsuchen iterative Indexscans weitere Teile des Index, bis genügend Zeilen den Filter passieren oder

Scangrenzen erreicht werden. Aktiviert werden sie mit `SET hnsw.iterative_scan = strict_order` oder mit `relaxed_order` für besseren Recall bei geringfügig abweichender Ergebnisreihenfolge. Für HNSW ist `hnsw.max_scan_tuples`, voreingestellt 20.000, eine ungefähre Grenze für zusätzlich besuchte Tupel; sie betrifft den anfänglichen Scan nicht. Auch eine gesonderte Speichergrenze kann weitere Scans beenden. Bei einem Filter mit nur wenigen verschiedenen Werten funktioniert auch ein partieller Index je Wert.

Eine Abfrage verfehlt den Index für nächste Nachbarn. `ORDER BY 1 - (embedding <=> $1) DESC` sortiert nach einem Ausdruck. Eine Abfrage ohne `LIMIT` fordert kein begrenztes Ergebnis nächster Nachbarn an. Beides erfüllt die Anforderungen von pgvector für die Verwendung dieses Index nicht. Wie viele Zeilen der gewählte Plan liest, hängt außerdem von Filtern und anderen Entscheidungen des Planers ab. Ein mit `vector_l2_ops` erstellter Index hilft einer nach `<=>` sortierten Abfrage nicht. Sortiert werden sollte nach dem reinen Operator, aufsteigend und mit einem `LIMIT`. Für jedes abgefragte Abstandsmaß ist ein eigener Index anzulegen; geprüft wird der Plan mit `EXPLAIN (ANALYZE, BUFFERS)`.

Ein IVFFlat-Index wird zu früh erstellt. IVFFlat führt k-means über die beim Aufbau vorhandenen Zeilen aus und teilt die Vektoren anhand des Ergebnisses in Listen. Ein mit zu wenigen Daten für seine Listenzahl erstellter Index liefert weniger Zeilen als angefordert. pgvector rät, ihn zu entfernen, bis die Tabelle mehr Daten enthält. Zuerst werden die Daten geladen, danach wird der Index gebaut. pgvector empfiehlt als Ausgangspunkt bis zu einer Million Zeilen die Zeilenzahl / 1000 Listen, darüber die Quadratwurzel der Zeilenzahl. Die Zahl der probes, voreingestellt 1, sollte auf die Quadratwurzel der Listenzahl gesetzt werden. HNSW hat keinen vergleichbaren Trainingsschritt.

Zeilen werden nie gefunden. NULL-Vektoren werden nicht indiziert, bei Kosinusabstand auch keine Nullvektoren, denn ihnen fehlt eine vergleichbare Richtung. Eine Zeile mit noch fehlendem Embedding ist für eine indexgestützte Suche unsichtbar. Sobald das Laden zuverlässig funktioniert, sollte `NOT NULL` ergänzt und die Zahl noch fehlender Embeddings regelmäßig ermittelt werden.

Vektoren zweier Modelle liegen in einer Spalte. Zwei Embedding-Modelle oder zwei Versionen eines Modells platzieren denselben Text an unterschiedlichen Stellen. Ihre Vektoren sind deshalb auch bei gleicher Dimensionszahl nicht vergleichbar. Die Frage muss mit demselben Modell wie die Passagen eingebettet, das Modell in einer Spalte gespeichert und bei einem

Modellwechsel alles neu eingebettet werden. Für Spalten mit Vektoren verschiedener Größen zeigt pgvector einen Ausdrucksindex mit einer Bedingung je Modell, `WHERE (model_id = 123)`.

Zu viele Dimensionen für den Index. Der Typ `vector` speichert bis zu 16.000 Dimensionen, HNSW und IVFFlat indizieren aber höchstens 2.000. Ein indizierter `halfvec`-Cast erlaubt bis zu 4.000, binäre Quantisierung bis zu 64.000 mit anschließender Neusortierung der Vorauswahl anhand der vollständigen Vektoren. Unterstützt das Modell es, kann auch ein Teilvektor indiziert werden.

Die Vektoren eines Kunden bremsen einen anderen. pgvector warnt, dass in einem zwischen Mandanten geteilten näherungsweisen Index die Vektoren eines Mandanten den Recall und die Geschwindigkeit für die anderen beeinflussen. Die Tabelle sollte mit Listenpartitionierung nach Kunden aufgeteilt werden, oder jeder Kunde erhält eine eigene Tabelle.

Ein Indexaufbau blockiert die Anwendung. HNSW wird wesentlich schneller aufgebaut, wenn der Graph in `maintenance_work_mem` passt. pgvector meldet, wenn das nicht mehr der Fall ist. Den Speicher so hoch anzusetzen, dass dem Server für andere Aufgaben zu wenig bleibt, verursacht ebenfalls einen Ausfall. Auf einem laufenden System sollte `CREATE INDEX CONCURRENTLY` verwendet werden; es blockiert Schreibvorgänge nicht.

Passagen, die der Leser nicht sehen darf. Eine Suche über alle Passagen kann einem Sprachmodell Text aus einem Dokument geben, das der Benutzer nicht lesen darf. Das Modell kann ihn zitieren. Die Berechtigungsbedingung gehört in dieselbe `WHERE`-Bedingung wie die Suche, damit eine solche Zeile die Datenbank nie verlässt. Iterative Scans sollten aktiviert werden, um innerhalb ihrer Scangrenzen nach weiteren erlaubten Passagen zu suchen, wenn der Filter Kandidaten entfernt.

Ein Abstand wird für Gewissheit gehalten. Ein Kosinusabstand von 0,2 bedeutet, dass eine Passage näher liegt als eine mit 0,4, nicht, dass sie die Frage beantwortet. Ein für ein Modell eingestellter Schwellenwert sagt nichts über das nächste aus. Nach Abstand sortieren, mit `LIMIT` begrenzen und die Suche anhand echter Fragen mit im Voraus bekannten richtigen Passagen beurteilen.

Weiterführende Lektüre

- pgvector — die eigene Dokumentation der Erweiterung: Typen, Abstandsoperatoren, HNSW- und IVFFlat-Optionen, Filter, iterative Indexscans, hybride Suche und Fehlerbehebung.
- Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs — die Veröffentlichung von Malkov und Yashunin, die HNSW vorstellte.
- Reciprocal Rank Fusion outperforms Condorcet and individual Rank Learning Methods — die SIGIR-Veröffentlichung von 2009 mit der Zusammenführungsformel und den Vorversuchen hinter ihrer Konstante.
- Introduction to full text search — die wortbasierte Suche von PostgreSQL als andere Hälfte einer hybriden Abfrage.
- Controlling Text Search — `plainto_tsquery` und `ts_rank_cd`, wie sie oben in der Volltexthälfte verwendet werden.
- Transaction Isolation — wann PostgreSQL-Snapshots festgeschriebene Änderungen sehen und warum ein älterer Snapshot ein Dokument noch nach dem Commit seiner Löschung sehen kann.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Bringen Sie einen repräsentativen Datensatz, die Fragen, die er beantworten muss, und den aktuellen Engpass mit. Wir können vereinbaren, wie Qualität, Durchsatz und Wiederherstellung gemessen werden.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist Vektorsuche?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/vector-search/