Engineer

ベクトル検索

クエリーベクトルに最も近いベクトルを持つ保存済み項目の検索

どのようなものか

ベクトル検索は、保存した項目の中から、検索用のベクトルに最も近いベクトルを持つものを探します。ベクトル類似度検索とも呼びます。ここでいうベクトルは、埋め込みモデルが、テキスト、画像、製品の記録から作る、一定の長さの数値の並びです。モデルは、内容が似た項目のベクトルが近くなるよう訓練されています。そのため、共通の単語がなくても、質問とその答えとなる文章が近くに位置することがあります。

「近い」を決めるには尺度が必要です。この記事の例で使うPostgreSQL拡張pgvectorは、六つの尺度に対応します。L2、すなわちユークリッド距離、内積、コサイン距離、L1距離に加え、二値ベクトル用のハミング距離とジャカード距離です。どの尺度に向けたベクトルかは、埋め込みモデル自身の文書に記されています。長さ1に正規化したベクトルでは、コサイン距離と内積は近傍を同じ順に並べ、pgvectorは最高の性能を得るため、内積を勧めています。

答えを求める方法は二つあります。厳密検索は、保存したすべてのベクトルと検索用のベクトルを比較し、真の最近傍を返します。pgvectorの既定はこれで、文書では完全な再現率と説明しています。近似検索では、データの一部だけを調べるインデックスを使い、再現率の一部と引き換えに速さを得ることができます。真の近傍がいくつ返るかは、データ、インデックス、設定によります。過半数さえ保証されません。そのため、pgvectorは、通常のインデックスとは違い、近似インデックスを使うと、クエリーの返す結果が変わると警告しています。

pgvectorには、二種類の近似インデックスがあります。HNSWは、複数の層からなるグラフを作ります。MalkovとYashuninが2016年に初めて公開した手法です。各層は一つ下の層の部分集合で、検索は小さな最上層から始め、全体を持つ層まで下ります。論文は、検索コストが対数的に増えると報告しています。IVFFlatは、ベクトルを複数のリストへ分け、クエリーに近いリストだけを検索します。

概念を判断に生かす

「ベクトル検索とは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 AIエージェントが記録にアクセスでき、顧客ごとの情報が一本化され、権限を超えた情報が表示されず、回答の出所をたどれる状態にします。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 地理情報パイプラインを50倍に高速化 — 50x 地理データパイプラインの高速化

地理情報パイプラインを50倍に高速化 お問い合わせ →

なぜ重要なのか

全文検索は単語を照合します。単語の照合と語幹処理だけでは、顧客が「画面が真っ暗なまま」と書いても、「ディスプレイの電源が入らない」という見出しの説明は見つかりません。語幹処理で一つにまとめられる共通の単語がないからです。同義語を設定すれば、一部の隔たりは埋められます。ベクトル検索は、埋め込みモデルが捉えた意味を比較するので、訴えと説明が近くに位置し得ます。同じ考え方は、テキスト以外にも使えます。この画像に似た画像、解決済みの問い合わせに似た問い合わせ、カート内の製品に似た製品などです。

検索拡張生成で情報を取得する方法の一つでもあります。大規模言語モデルは、質問に合わせて取得した文章から回答します。質問を文章と同じモデルで埋め込み、近い文章をプロンプトへ渡します。取得した内容以上に良い回答は期待できないため、検索にもモデルと同じだけ注意を払う必要があります。

すでに運用しているデータベースにベクトルを保存することには、実務上の利点もあります。pgvectorの文書は、その主張から始まります。ベクトルにも、ほかの列と同じトランザクション、時点復元、結合が使えます。pgvectorは先行書き込みログへ記録するので、レプリケーションにも含まれます。顧客、日付、権限の絞り込みは、同じ行への通常の `WHERE` 句です。削除がコミットした後で取得したスナップショットには、その文書は見えませんが、古いスナップショットを使うクエリーには、まだ見える場合があります。別のベクトルストアを持てば、データを入れ、元のシステムと同期させる、二つ目のシステムが増えます。

ベクトル検索がしないことも、チームは理解する必要があります。並べる基準は近さであり、真実かどうかではありません。最も近い文章は、質問に最も似ている文章です。古い、間違っている、読者に閲覧権限がない文書から来た、といった可能性は残ります。名前、部品番号、エラーコードのように、正確な語に依存する質問もあります。pgvectorの文書は、PostgreSQLの全文検索と組み合わせるよう提案しています。この組み合わせを、ハイブリッド検索と呼びます。

実際の活用例

2026年10月時点のpgvector 0.8.7は、PostgreSQL 13以降に対応します。データベースごとに一度だけ拡張を有効にし、モデルの出力次元数を指定した `vector` 型の列をテーブルに設けます。

```
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);
```

三次元なのは、例を読みやすくするためです。実際のモデルはもっと多くの次元を出力し、列にはその数を宣言します。アプリケーションがモデルで各埋め込みを計算し、元のテキストの隣へ挿入します。インデックスが対象にする距離は一つです。 `vector_cosine_ops` はコサイン距離順のクエリー用で、L2距離順のクエリーには `vector_l2_ops` で作る二つ目のインデックスが必要です。HNSWには訓練の段階がないため、空のテーブルでもインデックスを作れます。

クエリーは、質問を同じモデルで埋め込み、最も近い文章を求めます。

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

`<=>` はコサイン距離で、 `$1` はパラメーターとして渡すクエリーベクトルです。プランナーがインデックスを使うためには、二つの条件があります。 `ORDER BY` と `LIMIT` が両方あり、 `ORDER BY` が、距離演算子そのものの昇順であることです。表示用の類似度スコア `1 - (embedding <=> $1)` は、選択する列の一覧へ置きます。ほかの演算子は、L2距離の `<->` 、L1の `<+>` 、内積の `<#>` です。最後のものは負の内積を返します。PostgreSQLが演算子を使うインデックス走査で、昇順だけに対応するためです。

HNSWインデックスには、構築時のパラメーターが二つあります。`m` は、各層でノードがつなぐ近傍の数で、指定しなければ16です。`ef_construction` は構築中の候補リストの大きさで、既定は64です。検索時の `hnsw.ef_search` は候補リストの大きさを決め、既定は40です。増やすと真の最近傍を多く見つけますが、クエリーは遅くなります。`SET LOCAL` で増やすと、現在のトランザクションの終わりまで有効です。一つのクエリーだけに限定するには、そのクエリーだけをトランザクションに入れます。インデックスが再現率に与える影響を知るには、`SET LOCAL enable_indexscan = off` を指定したトランザクションで同じクエリーを実行し、厳密な答えを得て、二つの一覧を比較します。

会社の専用AIエージェント開発では、過去の事例の記憶を、ベンダーのチャット履歴ではなく、顧客自身のデータベースに保存します。`passages` のようなテーブルは、その記憶の形の一つです。新しい事例が来たとき、最も似た過去の事例を、一回のクエリーで、ほかのデータと同じデータベース、同じ権限の下から取り出せます。

ハイブリッド検索は両方の検索を行い、二つの一覧を統合します。pgvectorの文書は、Cormack、Clarke、Büttcherの2009年の論文によるReciprocal Rank Fusionを挙げています。各文書は、登場する一覧ごとに1 / (k + rank)の点数を得て、それを合計します。著者らはkを60に固定し、予備実験では60が「ほぼ最適だが、選択は決定的ではない」と報告しました。同じテーブルに、全文検索と同じ方法で作った `tsvector` 列 `search_vec` があるとすると、統合するクエリーは次のようになります。

```
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;
```

各検索は、自身の上位20件を残し、1から順位を付けます。両方で上位の文章は二つの点数を得て上位へ進み、一方だけで見つかった文章は一つ分を得ます。コサイン距離とテキストの順位スコアは尺度が違って比較できないため、統合にはスコアではなく順位を使います。pgvectorは、一覧を組み合わせる別の方法として、クロスエンコーダーを挙げています。

問題が起きるところ

`LIMIT` **で求めた数より行が少ない。** 近似インデックスでは、インデックス走査の後で `WHERE` 句を適用します。pgvectorは計算例を示しています。条件に一致する行が10%なら、既定の `hnsw.ef_search` が40のHNSWインデックスでは、平均で約4行が残ります。0.8.0以降には反復インデックス走査があり、十分な行がフィルターを通るか、走査上限へ達するまで、インデックスを追加で走査します。`SET hnsw.iterative_scan = strict_order` で有効にするか、`relaxed_order` で順序の多少の乱れと引き換えに再現率を上げます。HNSWの `hnsw.max_scan_tuples` は、追加で調べるタプル数のおおよその上限で、既定は20,000です。最初の走査には影響しません。別のメモリー上限で追加走査が止まる場合もあります。値の種類が少ないフィルターなら、値ごとの部分インデックスも使えます。

クエリーが最近傍インデックスを使わない。 `ORDER BY 1 - (embedding <=> $1) DESC` は式で並べ替えます。また、`LIMIT` がないクエリーは、件数を限定した最近傍結果を要求しません。どちらもpgvectorがそのインデックスを使うための条件を満たしません。選ばれた実行計画が何行読むかは、フィルターや、ほかのプランナーの判断にもよります。`vector_l2_ops` で作ったインデックスは、`<=>` 順のクエリーには役立ちません。演算子そのものの昇順で、`LIMIT` を付けます。検索する距離ごとにインデックスを作り、`EXPLAIN (ANALYZE, BUFFERS)` で計画を確かめます。

IVFFlatインデックスを早く作りすぎる。 IVFFlatは、構築時に存在する行へk-meansを適用し、その結果からベクトルをリストへ分けます。リスト数に対してデータが少なすぎる時点で作ると、要求より少ない行しか返りません。pgvectorは、データが増えるまで削除するよう勧めています。まずデータを入れ、後から構築します。開始時の目安として、百万行まではrows / 1000個のリスト、それより多ければ行数の平方根を提案しています。既定で1のprobesは、リスト数の平方根から始めます。HNSWには、この訓練段階はありません。

決して見つからない行がある。 NULLベクトルはインデックスに入りません。コサイン距離では、比較する方向がないゼロベクトルも入りません。埋め込みが未作成の行は、インデックスを使う検索から見えません。読み込みが安定してから `NOT NULL` を追加し、埋め込みがない行数を定期的に数えます。

二つのモデルのベクトルを一つの列に混ぜる。 別々の埋め込みモデル、あるいは同じモデルの別バージョンは、同じテキストを異なる位置へ置きます。次元数が一致しても比較できません。文章を埋め込んだモデルで質問も埋め込み、モデルを列に保存し、モデルを変えるときはすべてを再び埋め込みます。複数の次元数を持つ列には、pgvectorは、モデルごとの条件 `WHERE (model_id = 123)` を付けた式インデックスの例を示しています。

インデックスの次元数の上限を超える。 `vector` 型は最大16,000次元を保存しますが、HNSWとIVFFlatがインデックス化できるのは最大2,000次元です。最大4,000次元なら `halfvec` へキャストしてインデックス化し、最大64,000次元なら二値量子化を使って、候補を完全なベクトルで並べ直します。モデルが対応していれば、部分ベクトルをインデックス化する方法もあります。

ある顧客のベクトルが、ほかの顧客を遅くする。 pgvectorは、テナント間で近似インデックスを共有すると、一つのテナントのベクトルが、ほかのテナントの再現率や速さに影響すると警告しています。テーブルを顧客ごとにリストパーティションで分割するか、顧客ごとに別のテーブルを与えます。

インデックスの構築がアプリケーションを止める。 HNSWは、グラフが `maintenance_work_mem` に収まると大幅に速く構築できます。収まらなくなると、pgvectorは通知を出します。ただし、そのメモリーを大きくしすぎてサーバーのメモリーが不足するのも、障害になります。稼働中のシステムでは、書き込みを止めない `CREATE INDEX CONCURRENTLY` で構築します。

読者が見てはいけない文章を返す。 すべての文章を検索すると、利用者に閲覧権限がない文書のテキストが言語モデルへ渡り、モデルが引用できてしまいます。検索と同じ `WHERE` 句へ権限条件を入れ、その行がデータベースの外へ出ないようにします。フィルターが候補を除いたときは、走査上限の範囲で、さらに閲覧可能な文章を探せるよう、反復走査も有効にします。

距離を確信度として読む。 コサイン距離0.2は、0.4より近いという意味であり、質問に答えるという意味ではありません。一つのモデルで調整した閾値は、次のモデルには意味を持ちません。距離で並べ、 `LIMIT` で件数を制限し、正しい文章が分かっている実際の質問集で検索を評価します。

参考資料

- pgvector — 拡張自身の文書です。型、距離演算子、HNSWとIVFFlatの設定、フィルター、反復インデックス走査、ハイブリッド検索、問題の調査を扱います。
- Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs — HNSWを導入したMalkovとYashuninの論文です。
- Reciprocal Rank Fusion outperforms Condorcet and individual Rank Learning Methods — 統合の式と、定数の根拠となった予備実験を載せた、SIGIR 2009の論文です。
- 全文検索の概要 — ハイブリッド検索のもう半分である、PostgreSQLの単語に基づく検索です。
- テキスト検索の制御 — 上のキーワード検索で使う `plainto_tsquery` と `ts_rank_cd` を説明しています。
- トランザクション分離 — PostgreSQLのスナップショットにコミット済みの変更が見える時点と、削除のコミット後も古いスナップショットから文書が見える理由です。

用語集

実際の制約に合わせて選ぶ

代表的なデータセット、そこから答えを得たい問い、現在のボトルネックをお知らせください。品質、処理量、復旧の測り方を合意できます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「ベクトル検索とは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

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

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/vector-search/