Engineer

Vektorsøk

søk etter de vektorene som ligger nærmest en spørring

Hva det er

Vektorsøk finner de lagrede objektene der vektoren ligger nærmest vektoren til søket. På engelsk heter det vector search. En vektor er her en liste med tall av fast lengde som en embeddingmodell regner ut fra et tekststykke, et bilde eller en produktoppføring. Modellen er trent slik at objekter med lignende innhold får vektorer som ligger nær hverandre, og derfor kan et spørsmål og avsnittet som besvarer det, havne tett sammen selv når de ikke har et eneste ord felles.

«Nærmest» krever et mål. pgvector, PostgreSQL-utvidelsen som eksemplene i denne artikkelen bruker, støtter seks: L2-avstand (euklidsk), indreprodukt, cosinusavstand og L1-avstand, og i tillegg Hamming-avstand og Jaccard-avstand for binære vektorer. Dokumentasjonen til embeddingmodellen sier hvilket mål vektorene er laget for. For vektorer som er normalisert til lengde 1, gir cosinusavstand og indreprodukt naboene samme rekkefølge, og der anbefaler pgvector indreprodukt fordi det gir best ytelse.

Det finnes to måter å besvare søket på. Eksakt søk sammenligner søket med hver eneste lagrede vektor og returnerer de faktiske nærmeste naboene. Det er dette pgvector gjør som standard, og dokumentasjonen kaller det perfekt gjenfinning (recall). Omtrentlig søk kan bruke en indeks som bare besøker en del av dataene, og bytter bort litt gjenfinning mot fart. Hvor mange faktiske naboer som kommer med, avhenger av dataene, indeksen og innstillingene; selv et flertall er ikke garantert. Derfor advarer pgvector om at en omtrentlig indeks, i motsetning til en vanlig indeks, endrer hva en spørring returnerer.

pgvector har to typer omtrentlige indekser. HNSW bygger en graf i flere lag, en metode Malkov og Yashunin først publiserte i 2016. Hvert lag inneholder et utvalg av nodene i laget under, et søk starter i det lille øverste laget og går

nedover til det fullstendige, og artikkelen deres rapporterer at søkekostnaden vokser logaritmisk. IVFFlat deler vektorene inn i lister og søker bare i listene som ligger nærmest søket.

Bruk begrepet i en beslutning

Bruk «Hva er vektorsøk?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Dataene dine samlet der AI-agenter kan nå dem, med én versjon av hver kunde, uten at noen får se mer enn de skal, og med hvert svar sporet tilbake til kilden.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Gjorde en geodatapipeline 50 ganger raskere — 50x raskere pipelines for geografiske data

Gjorde en geodatapipeline 50 ganger raskere Ta kontakt →

Hvorfor det er viktig

Fulltekstsøk sammenligner ord. Med bare ordmatching og ordstemming finner en kunde som skriver «skjermen forblir svart», ikke avsnittet i håndboken med overskriften «displayet slår seg ikke på», fordi de to ikke deler noe ord som ordstemmingen kan føre sammen. Konfigurerte synonymer kan bygge bro over noen slike forskjeller. Vektorsøk sammenligner betydning slik embeddingmodellen har fanget den opp, så klagen og avsnittet kan havne nær hverandre. Den samme ideen virker utover tekst: et bilde som ligner dette, en supportsak som ligner en som allerede er løst, et produkt som ligner det som ligger i handlekurven.

Det er også én måte å gjøre gjenfinningen i retrieval-augmented generation på, der en stor språkmodell svarer ut fra tekstavsnitt som er hentet til spørsmålet. Spørsmålet gjøres om til en vektor med den samme modellen som avsnittene, og de nærmeste avsnittene legges inn i prompten. Svaret blir aldri bedre enn det søket fikk med seg, så søket fortjener like mye omtanke som modellen.

Å ha vektorene i databasen du allerede drifter har en praktisk side, og pgvector åpner dokumentasjonen sin med akkurat det argumentet. Vektorene får de samme transaksjonene, den samme gjenopprettingen til et tidspunkt (point-in-time recovery) og de samme joinene som alle andre kolonner, og fordi pgvector skriver til write-ahead-loggen (WAL), tar replikeringen dem med også. Et filter på kunde, dato eller tilgang er en vanlig `WHERE` -betingelse på den samme raden. Et øyeblikksbilde tatt etter at en sletting er bekreftet, ser ikke lenger dokumentet, men en spørring med et eldre øyeblikksbilde kan fortsatt se det. Et eget vektorlager er et system til som må fylles og holdes i takt med det første.

Et team bør også vite hva vektorsøk ikke gjør. Det rangerer etter nærhet, ikke etter sannhet. Det nærmeste avsnittet er det som ligner mest på spørsmålet; det kan likevel være utdatert, feil eller hentet fra et dokument leseren ikke har lov til å se. Noen spørsmål står og faller på ett eksakt ord, som et navn, et delenummer eller en feilkode, og pgvectors dokumentasjon foreslår å bruke vektorsøk sammen med fulltekstsøket i PostgreSQL. Den kombinasjonen kalles hybridsøk.

Slik ser det ut i praksis

pgvector, i versjon 0.8.7 i oktober 2026, støtter PostgreSQL 13 og nyere. Utvidelsen slås på én gang i hver database, og deretter får tabellen en kolonne av typen `vector` med det antallet dimensjoner modellen gir:

```
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);
```

Tre dimensjoner holder eksemplet lesbart; en ekte modell gir langt flere, og kolonnen oppgir det antallet modellen gir. Applikasjonen regner ut hver embedding med modellen og setter den inn ved siden av teksten den kom fra. Indeksen gjelder én avstand: `vector_cosine_ops` betjener spørringer sortert etter cosinusavstand, og en spørring sortert etter L2-avstand trenger en indeks til, bygd med `vector_l2_ops`. HNSW har ikke noe treningssteg, så indeksen kan opprettes på en tabell uten rader.

Spørsmålet gjøres om til en vektor med den samme modellen, og spørringen ber om de nærmeste avsnittene:

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

`<=>` er cosinusavstand, og `$1` er søkevektoren, sendt som parameter. To detaljer gjør at planleggeren kan bruke indeksen: spørringen har både `ORDER BY` og `LIMIT`, og `ORDER BY` står på selve avstandsoperatoren i stigende rekkefølge. En likhetsscore til visning, `1 - (embedding <=> $1)`, hører hjemme i `SELECT`-listen. De andre operatorene er `<->` for L2-avstand, `<+>` for L1 og `<#>` for indreprodukt. Den siste returnerer det negative indreproduktet, fordi PostgreSQL bare skanner en indeks på en operator i stigende rekkefølge.

HNSW-indeksen har to byggeparametere: `m`, det største antallet forbindelser per lag, 16 som standard, og `ef_construction`, størrelsen på kandidatlisten under byggingen, 64 som standard. Ved søk setter `hnsw.ef_search` størrelsen på kandidatlisten, 40 som standard. En høyere verdi gir bedre gjenfinning på bekostning av fart, og `SET LOCAL` hever den fram til slutten av den gjeldende transaksjonen. For å begrense innstillingen til én spørring må transaksjonen bare inneholde den spørringen. Vil du vite hvor mye gjenfinning indeksen koster, kjører du den samme spørringen i en transaksjon med `SET LOCAL enable_indexscan = off`, som gir det eksakte svaret, og sammenligner de to listene.

Selskapets arbeid med skreddersydde AI-agenter legger agentens minne om tidligere saker i kundens egen database, ikke i en leverandørs chathistorikk. En tabell som `passages` er én form et slikt minne kan ta: når en ny sak kommer inn, ligger de tidligere sakene som ligner mest, én spørring unna, i den samme databasen og under de samme tilgangene som resten av dataene.

Hybridsøk kjører begge søkene og fletter de to listene sammen. pgvectors dokumentasjon nevner Reciprocal Rank Fusion, fra en artikkel av Cormack, Clarke og Büttcher fra 2009: hvert dokument får 1 / (k + rang) for hver liste det står i, og andelene legges sammen. Forfatterne satte k til 60, og forsøkskjøringene deres viste at 60 var «near-optimal, but that the choice was not critical». Med en `tsvector`-kolonne `search_vec` i den samme tabellen, bygd som for fulltekstsøk, ser den flettede spørringen slik ut:

```
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('norwegian', $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;
```

Hvert søk beholder sine egne 20 beste og nummererer dem fra 1. Et avsnitt som begge søkene plasserer høyt, får to andeler og stiger til topps; et avsnitt som bare ett søk fant, beholder én. Flettingen bruker rangeringer i stedet for poengsummer, fordi en cosinusavstand og en tekstrang ligger på skalaer som ikke kan sammenlignes. Språket i `plainto_tsquery` må være det samme som i kolonnen, her `norwegian` . pgvector nevner en cross-encoder som den andre måten å kombinere listene på.

Hvor det går galt

Færre rader enn `LIMIT` ba om. Med en omtrentlig indeks brukes `WHERE` -betingelsen først etter at indeksen er skannet. pgvector viser regnestykket: hvis en betingelse treffer 10 % av radene, gir en HNSW-indeks med standardverdien 40 for `hnsw.ef_search` i snitt bare rundt 4 rader. Fra og med versjon 0.8.0 skanner iterative indeksskanninger mer av indeksen helt til nok rader har sluppet gjennom filteret eller skannegrensene er nådd. Slå dem på med `SET hnsw.iterative_scan = strict_order` , eller `relaxed_order` for bedre gjenfinning med resultater som kan stå litt i feil rekkefølge. For HNSW er `hnsw.max_scan_tuples` , 20 000 som standard, en omtrentlig grense for ytterligere besøkte tupler og påvirker ikke den første skanningen. En egen minnegrense kan også stoppe videre skanning. For et filter med bare noen få ulike verdier fungerer også en partiell indeks per verdi.

En spørring som ikke treffer nærmeste-nabo-indeksen. `ORDER BY 1 - (embedding <=> $1) DESC` sorterer etter et uttrykk, og en spørring uten `LIMIT` ber ikke om et avgrenset nærmeste-nabo-resultat. Ingen av delene oppfyller pgvectors krav for å bruke den indeksen. Hvor mange rader den valgte planen leser, avhenger også av filtre og andre valg i planleggeren. En indeks bygd med `vector_l2_ops` hjelper

ikke en spørring sortert etter `<=>`. Sorter etter selve operatoren, stigende, med `LIMIT`; bygg én indeks per avstand du søker med; og sjekk planen med `EXPLAIN (ANALYZE, BUFFERS)`.

En IVFFlat-indeks bygd for tidlig. IVFFlat kjører k-means over radene som finnes når indeksen bygges, og deler vektorene inn i lister ut fra resultatet. En indeks som er opprettet med for lite data for antallet lister, returnerer færre rader enn spørringen ba om, og pgvector råder til å fjerne den til tabellen har fått mer data. Last inn dataene først, og bygg deretter. pgvector foreslår å begynne med rader / 1000 lister opp til en million rader og kvadratroten av antall rader over det, og å sette probes, 1 som standard, til kvadratroten av antall lister. HNSW har ikke noe slikt treningssteg.

Rader som aldri blir funnet. NULL-vektorer indekseres ikke, og det gjør heller ikke nullvektorer ved cosinusavstand, siden de ikke har noen retning å sammenligne. En rad som mangler embedding, er usynlig for et søk via indeksen. Legg til `NOT NULL` når innlastingen er pålitelig, og kjør en planlagt opptelling av rader som fremdeles mangler embedding.

Vektorer fra to modeller i én kolonne. To embeddingmodeller, eller to versjoner av samme modell, plasserer den samme teksten på forskjellige steder, og vektorene deres kan ikke sammenlignes selv når antallet dimensjoner er likt. Gjør spørsmålet om til en vektor med modellen som laget vektorene for avsnittene, lagre modellen i en kolonne, og regn ut alle vektorene på nytt når modellen byttes. For en kolonne med vektorer i flere størrelser viser pgvector en uttrykksindeks med en betingelse per modell, `WHERE (model_id = 123)`.

For mange dimensjoner for indeksen. Typen `vector` lagrer opptil 16 000 dimensjoner, men HNSW og IVFFlat indekserer høyst 2 000. Indekser en `halfvec`-konvertering for å nå opptil 4 000, bruk binær kvantisering for opptil 64 000 og ranger kortlisten på nytt med de fullstendige vektorene, eller indekser en delvektor når modellen støtter det.

Én kundes vektorer bremser en annens. pgvector advarer om at når flere kunder deler en omtrentlig indeks, kan vektorene til én av dem påvirke gjenfinning og fart for de andre. Partisjoner tabellen etter kunde med listepartisjonering, eller gi hver kunde sin egen tabell.

Et indeksbygg som blokkerer applikasjonen. HNSW bygges mye raskere når grafen får plass i `maintenance_work_mem`, og pgvector skriver ut en melding når den ikke lenger gjør det; å sette det minnet så høyt at serveren får for lite minne, gir nedetid i seg selv. På et system i drift bygger du med `CREATE INDEX CONCURRENTLY`, som ikke blokkerer skriving.

Avsnitt leseren ikke skal se. Et søk over alle avsnitt kan gi en språkmodell tekst fra et dokument brukeren ikke har rett til å lese, og modellen kan sitere den. Legg tilgangsbetingelsen i den samme `WHERE`-betingelsen som søket, slik at en slik rad aldri forlater databasen, og slå på iterative skanninger for å lete etter flere tillatte avsnitt når filtreringen fjerner kandidater, innenfor skannegrensene.

En avstand lest som visshet. En cosinusavstand på 0,2 sier at et avsnitt ligger nærmere enn et på 0,4, ikke at det besvarer spørsmålet. En terskel som er justert for én modell, betyr ingenting for den neste. Sorter etter avstand, begrens med `LIMIT`, og vurder søket mot et sett ekte spørsmål der de riktige avsnittene er kjent.

Les mer

- pgvector — utvidelsens egen dokumentasjon: typer, avstandsoperatorer, innstillinger for HNSW og IVFFlat, filtrering, iterative indeksskanninger, hybridsøk og feilsøking.
- Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs — artikkelen av Malkov og Yashunin som introduserte HNSW.
- Reciprocal Rank Fusion outperforms Condorcet and individual Rank Learning Methods — SIGIR-artikkelen fra 2009 med flettingsformelen og forsøkskjøringene bak konstanten.
- Introduction to full text search — ordbasert søk i PostgreSQL, den andre halvdelen av en hybrid spørring.
- Controlling Text Search — `plainto_tsquery` og `ts_rank_cd`, slik de brukes i ordsøket ovenfor.
- Transaction Isolation — når PostgreSQLs øyeblikksbilder ser bekreftede endringer, blant annet hvorfor et eldre øyeblikksbilde fortsatt kan se et dokument etter at slettingen er bekreftet.

Ordliste

Velg ut fra deres faktiske rammer

Ta med et representativt datasett, spørsmålene det skal besvare og dagens flaskehals. Vi kan avtale målinger av kvalitet, gjennomstrømning og gjenoppretting.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er vektorsøk?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

welcome@engineer.company +45 50 10 61 84 København, Danmark

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/vector-search/