

# PostgreSQLで ベクトル検索をやってみよう

～「犬」に近い単語は「猫」なのか「オオカミ」なのか～

2026/10/2

株式会社 SRA OSS

品田 達也

- 名前: 品田 達也 (しなだ たつや)
- 所属: 技術部 データベース技術グループ
- 担当業務:
  - PostgreSQLサポート

## 株式会社SRA OSS

**所在地:** 東京都豊島区南池袋2-32-8

**設立日:** 2022年6月17日

**株主:** 株式会社SRA  
株式会社NTTデータ

**資本金:** 7,000万円

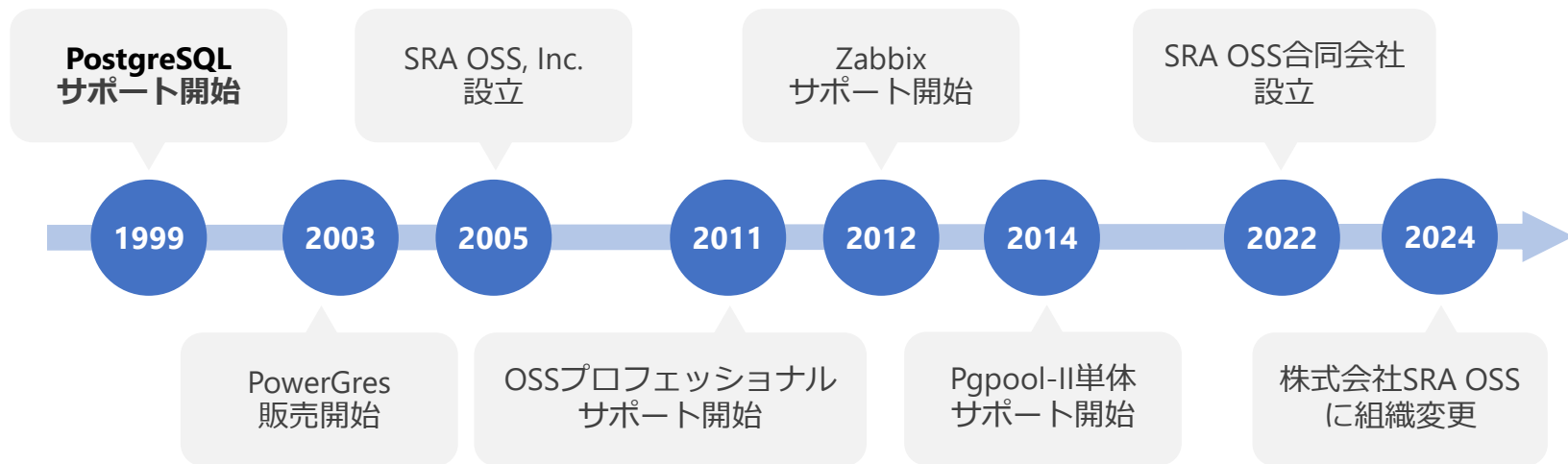
**社長:** 稲葉 香理

## 事業内容

- ・ オープンソースソフトウェア (OSS) 関連のサポート、製品開発・販売、構築・コンサル
- ・ OSSの教育、開発、コミュニティ運営支援
- ・ ソフトウェアの研究開発

**顧問:** 石井 達夫

**技術顧問:** 増永 良文 (お茶の水女子大学名誉教授)



- ベクトル検索とは
- ベクトルとは
- pgvector
  - pgvectorとは
  - pgvectorの準備
  - pgvectorの使い方
- エンベディングとは
- 単語ベクトルデータ
  - fastText学習済み単語ベクトルデータ
  - 単語ベクトルデータの登録
  - SQL文のLIKE検索
  - ベクトルデータで類似検索
- 「犬」に近い単語は「猫」なのか「オオカミ」なのか

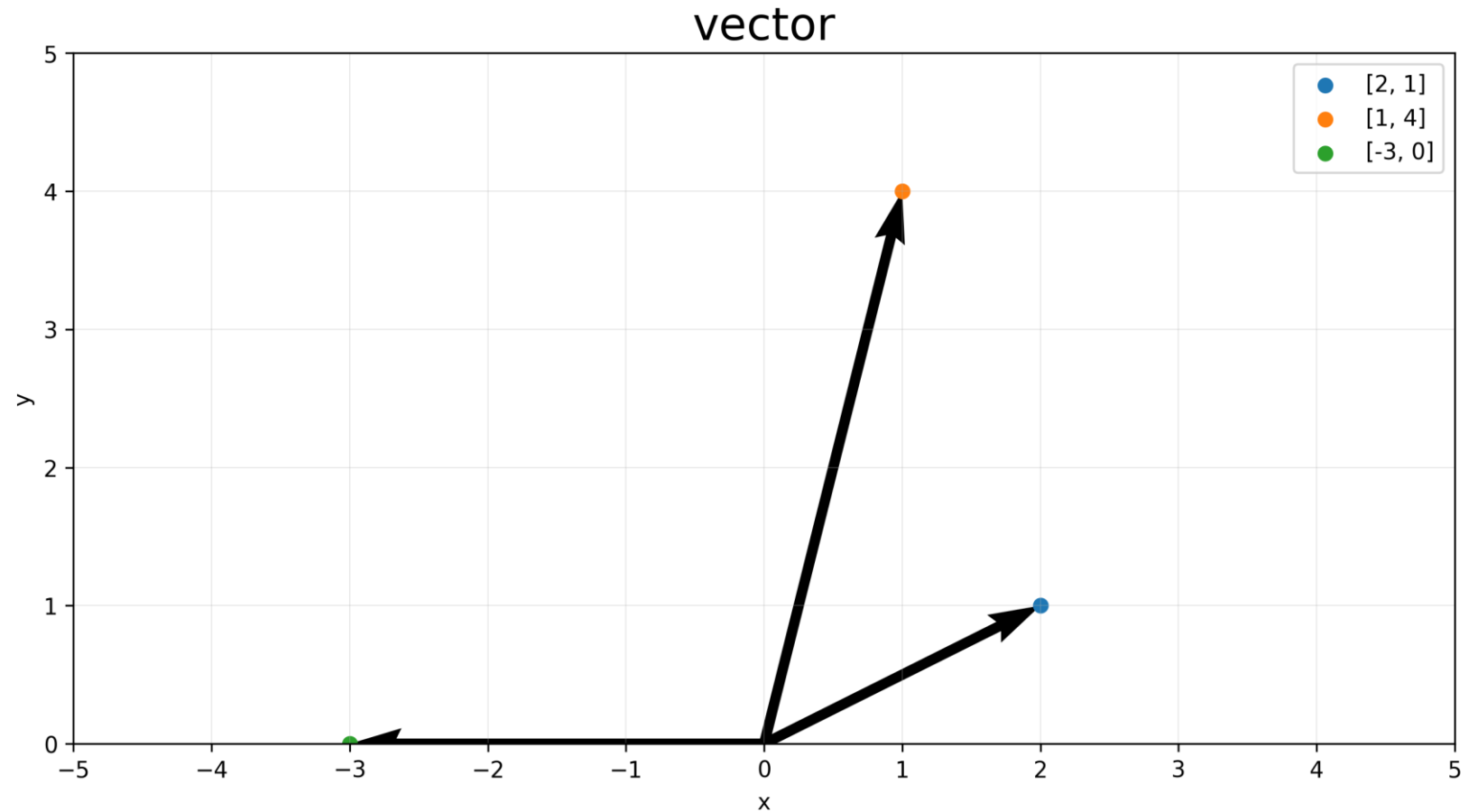
# ベクトル検索とは

- 生成AIやLLMとともに使われる技術
- ベクトルを使って意味が近い単語や画像を検索する方法
- PostgreSQLでもpgvectorを使うことで始められる

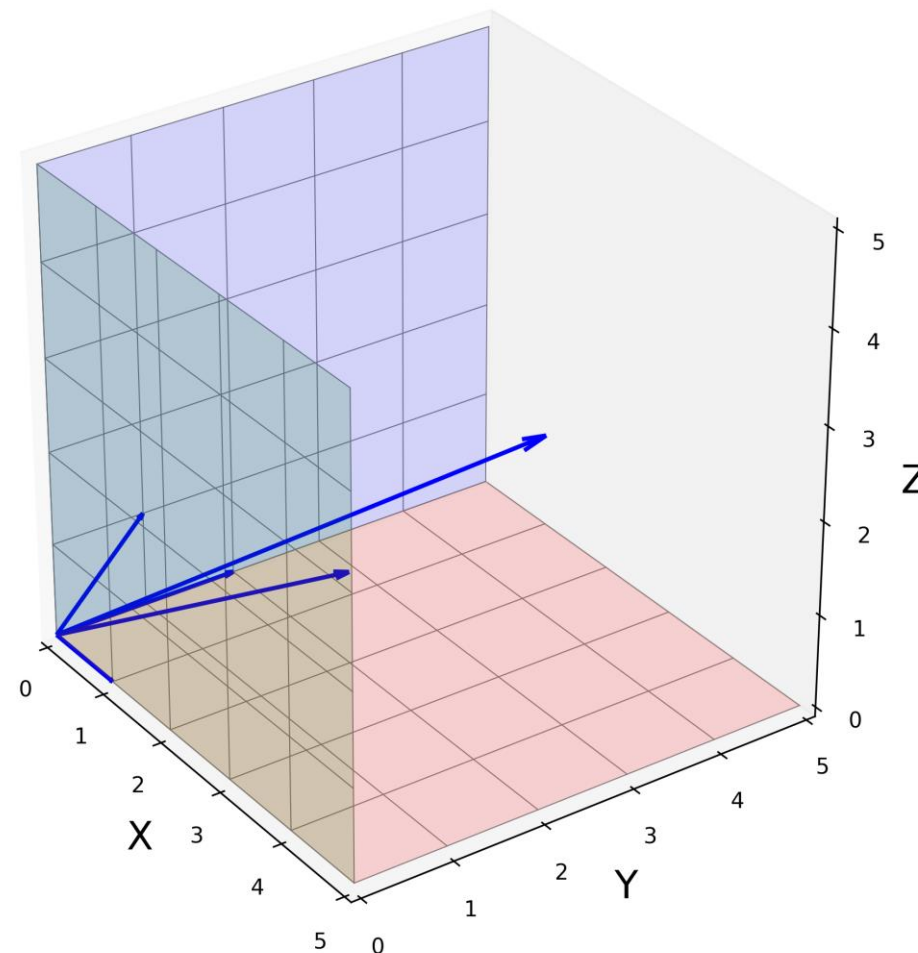
# ベクトルとは(1)

- ベクトルとは「固定個の数値の並び」のデータ
- 2個の並び(2次元ベクトル)の例
  - $[1, 7]$ 、 $[-4, 8]$ 、 $[0.2, 9.0]$ など
- 3個の並び(3次元ベクトル)の例
  - $[4, 7, -5]$ 、 $[-0.5, 100.2, 0]$ など
- 例えば10次元、180次元、なども同様に考えられる
- ベクトルはもともと数学の概念で、図示する場合は矢印として表現することが一般的

- 2次元ベクトル $[2, 1]$ 、 $[1, 4]$ 、 $[-3, 0]$ を図示



- 3次元ベクトル $[1, 0, 0]$ 、 $[0, 1, 1]$ 、 $[0, 2, 0]$ 、 $[2, 2, 1]$ 、 $[1, 5, 1]$ を図示



- ベクトルの大きさや、ベクトル同士の加算、乗算、距離が数学的に定義されている

例えば

- 2次元ベクトルの加算:  $[4, 5] + [2, 0] = [6, 5]$
- 3次元ベクトル $[3, 1, 5]$  と $[4, 0, 2]$ の
  - 内積:  $3 \times 4 + 1 \times 0 + 5 \times 2 = 22$
  - ベクトルの大きさ: 1個目  $\sqrt{3^2 + 1^2 + 5^2} = \sqrt{35} \approx 5.9160797831$   
2個目  $\sqrt{4^2 + 0^2 + 2^2} = \sqrt{20} \approx 4.472135955$
  - コサイン距離:  $1 - \frac{22}{\sqrt{35}\sqrt{20}} \approx 0.1684781594$  ※ベクトル検索で使用
  - ユークリッド距離:  $\sqrt{(3-4)^2 + (1-0)^2 + (5-2)^2} \approx 3.31662479$

- PostgreSQL にベクトルデータ用の型や関数を追加する  
拡張モジュール
- 検索を高速化するためのインデックス機能も提供
- PostgreSQL に追加してインストールする必要がある
- 今回はRocky Linux 9.6 + PostgreSQL 18.6 + pgvector 0.8.6 で構築
- PostgreSQLのrpmはOSやPGDG公式などリポジトリが色々あり複雑
  - 今回は、単純にソースコードを取得してコンパイルする方法を紹介

- PostgreSQL 18.6 のインストール

```
# curl -O https://ftp.postgresql.org/pub/source/v18.6/postgresql-18.6.tar.bz2
% Total      % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 21.5M  100 21.5M    0     0 2605k      0  0:00:08  0:00:08 --:--:-- 3148k
```

```
# tar jxf postgresql-18.6.tar.bz2 -C /usr/local/src/
```

```
# chown -R postgres. /usr/local/src/postgresql-18.6/
```

```
# mkdir /usr/local/pgsql-18.6
```

```
# chown postgres. /usr/local/pgsql-18.6
```

```
# mkdir /data2/pgsql-18.6
```

```
# chown postgres. /data2/pgsql-18.6
```

```
# sudo -u postgres -i
```

```
$ cd /usr/local/src/postgresql-18.6/ --prefixオプションでインストール先を指定
```

```
$ ./configure --prefix=/usr/local/pgsql-18.6 >/tmp/pgsql-18.6.log 2>&1
```

```
$ make world >>/tmp/pgsql-18.6.log 2>&1
```

```
$ make install-world >>/tmp/pgsql-18.6.log 2>&1
```

```
$ make world; make install-world でcontrib含めソースコード全てをコンパイル/インストール
```

- pgvector 0.8.6 のインストール

```
# cd /usr/local/src/
# curl -OL https://github.com/pgvector/pgvector/archive/refs/tags/v0.8.6.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
   0      0   0      0   0      0     0     0  --:--:--   0:00:01 --:--:--    0
100 126k   0 126k   0      0 16658     0  --:--:--   0:00:07 --:--:-- 23954
# tar xf v0.8.6.tar.gz
# mv pgvector-0.8.6/ pgvector
# chown -R postgres. pgvector
# sudo -u postgres -i
$ cd /usr/local/src/pgvector/
$ make PG_CONFIG=/usr/local/pgsql-18.6/bin/pg_config >>/tmp/pgsql-18.6.log 2>&1
$ make PG_CONFIG=/usr/local/pgsql-18.6/bin/pg_config install >>/tmp/pgsql-
18.6.log 2>&1
$
```

pg\_configでインストール対象の  
PostgreSQLの情報を取得する

- PostgreSQL + pgvector のセットアップ

```
# sudo -u postgres -i
$ export PATH=/usr/local/pgsql-18.6/bin/:"$PATH"
$ export LD_LIBRARY_PATH=/usr/local/pgsql-18.6/lib/:"$LD_LIBRARY_PATH"
$ export PGDATA=/data2/pgsql-18.6/pgdata
$ initdb --encoding=UTF8 --no-locale
$ pg_ctl -l /tmp/logfile start
$ createuser -d testuser
$ createdb -O testuser testdb
$ psql -U postgres -d testdb
testdb=# CREATE EXTENSION vector;
CREATE EXTENSION
testdb=# \dx
```

コンソールでPostgreSQL  
コマンドのための環境変数

拡張モジュールの利用

```
                                List of installed extensions
   Name   | Version | Default version | Schema | Description
-----+-----+-----+-----+-----
plpgsql   | 1.0     | 1.0             | pg_catalog | PL/pgSQL procedural language
vector    | 0.8.6   | 0.8.6           | public    | vector data type and ivfflat and hns w access
methods
(2 rows)
```

- ベクトルデータを格納するテーブル
  - 例) 3次元のvector 型のvec 項目を定義
  - vector 型は、各要素はreal 型と同様の単精度浮動小数点数、最大16000次元

```
$ psql -U testuser -d testdb
testdb=> CREATE TABLE items (id bigserial PRIMARY KEY, vec vector(3));
testdb=> INSERT INTO items VALUES(1, '[1.0, 0.0, 0.0]');
testdb=> INSERT INTO items VALUES(2, '[0.0, 1.0, 1.0]');
testdb=> INSERT INTO items VALUES(3, '[2.0, 2.0, 1.0]');
testdb=> INSERT INTO items VALUES(4, '[1.0, 5.0, 1.0]');
testdb=> INSERT INTO items VALUES(5, '[0.0, 2.0, 0.0]');
testdb=> SELECT * FROM items ORDER BY id;
 id |  vec
-----+-----
  1 | [1,0,0]
  2 | [0,1,1]
  3 | [2,2,1]
  4 | [1,5,1]
  5 | [0,2,0]
(5 rows)
```

- ベクトル計算の実行

- 例) 加算、ユークリッド距離( $\leftrightarrow$ )で原点[0, 0, 0]に近い順にソート

```
testdb=> SELECT * FROM items ORDER BY id;
```

| id | vec     |
|----|---------|
| 1  | [1,0,0] |
| 2  | [0,1,1] |
| 3  | [2,2,1] |
| 4  | [1,5,1] |
| 5  | [0,2,0] |

```
testdb=> SELECT SUM(vec) FROM items;
```

| sum      |
|----------|
| [4,10,3] |

```
testdb=> SELECT * FROM items ORDER BY vec <-> '[0, 0, 0]';
```

| id | vec     |
|----|---------|
| 1  | [1,0,0] |
| 2  | [0,1,1] |
| 5  | [0,2,0] |
| 3  | [2,2,1] |
| 4  | [1,5,1] |

- ベクトル計算の実行

- 例) 3次元ベクトル[3, 1, 5] と[4, 0, 2]の

内積の符号を反転した値(<#>)、

ベクトルの大きさ(vector\_norm 関数)を計算

```
testdb=> SELECT '[3, 1, 5]':vector <#> '[4, 0, 2]':vector;  
?column?
```

```
-----  
      -22  
(1 row)
```

```
testdb=> SELECT vector_norm('[3, 1, 5]':vector), vector_norm('[4, 0,  
2]':vector);
```

```
      vector_norm |      vector_norm  
-----+-----  
 5.916079783099616 | 4.47213595499958  
(1 row)
```

- ベクトル計算の実行

- 例) 3次元ベクトル[3, 1, 5] と[4, 0, 2]の

コサイン距離(<=>)、

※ベクトル検索で使用

ユークリッド距離(<->)を計算

```
testdb=> SELECT '[3, 1, 5]':vector <=> '[4, 0, 2]':vector;  
           ?column?
```

```
-----  
0.16847815937970012  
(1 row)
```

```
testdb=> SELECT '[3, 1, 5]':vector <-> '[4, 0, 2]':vector;  
           ?column?
```

```
-----  
3.3166247903554  
(1 row)
```

- エンベディング(Embedding)とは
  - 文章、単語、画像、音声などのデータをベクトルに変換する技術
  - 埋め込み表現とも言う
  - ベクトルの次元は数百～数千以上の高次元になりえる
  - 意味が近いデータ同士を、距離が近いベクトル同士になるように変換
- エンベディングでデータ間の類似関係を扱うことが可能になる
  - 例えばこの画像は犬より猫に似ている、  
この音声はこの人物の声に一番近いなど

- 例えば「サッカーボール」「パンダ」「柴犬」という単語の、  
3次元ベクトルへのエンベディングを考える  
各次元に大きさ、色、毛の量といった意味を割り当てる
  - 大きさの次元では「サッカーボール」と「柴犬」は似ているので近い数値、  
「パンダ」は他の2つから離れた数値になる
  - 色の次元では「サッカーボール」と「パンダ」は白黒で似ているので近い数値、  
「柴犬」は離れた数値になる
  - 毛の量の次元では「柴犬」と「パンダ」は毛が生えているので近い数値、  
「サッカーボール」は離れた数値になる

# エンベディングとは(3)

- 同じような文脈で登場する単語は、ベクトル同士で近い距離になる
- 各次元の意味は、例えば大きさや色など、人間に理解できる場合もあるが、学習アルゴリズムによって、人間が直接解釈できない数値で各次元が定義構成される場合が多い
- どういう特徴が次元に選ばれるかはエンベディングの方法や入力データ次第
- ベクトルの次元を大きくすれば、データ量は大きくなるが、それだけ柔軟性の高い類似関係を扱うことが可能

- fastText はMeta社が開発したテキスト表現やテキスト分類器を学習できるオープンソースで無料の軽量ライブラリ
- fastText を用いてCommon CrawlとWikipediaで学習させた、157言語の事前学習済み単語ベクトルデータが配布されている
- これらの単語ベクトルデータは、CC BY-SA 3.0 ライセンスに基づいて配布されている
  - 提供元 : <https://fasttext.cc/docs/en/crawl-vectors.html>
  - ライセンス : Creative Commons Attribution-ShareAlike 3.0 (CC BY-SA 3.0)  
<https://creativecommons.org/licenses/by-sa/3.0/>

# 単語ベクトルデータの登録(1)

- fastText が配布している日本語の学習済み単語ベクトルデータ
  - 取得
  - 書式を変換
  - PostgreSQL + pgvector のテーブルを作成
  - psql の¥copy コマンドを使用してCSV形式でテーブルに登録
    - ¥copy コマンドはテーブルとファイル(または標準入出力)間で、データを高速に入出力するためのコマンド

=> 登録後に類似検索を実行してみる

- fastText の単語ベクトルデータを取得

```
[postgres@rocky9 ~]$ curl -O https://dl.fbaipublicfiles.com/fasttext/vectors-crawl/cc.ja.300.vec.gz
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             Dload  Upload  Total  Spent    Left   Speed
100 1220M  100 1220M    0     0 2992k      0  0:06:57  0:06:57 --:--:-- 3583k
[postgres@rocky9 ~]$ ls -lh cc.ja.300.vec.gz
-rw-r--r--. 1 postgres postgres 1.2G  8月 31 14:42 cc.ja.300.vec.gz
```

- 単語ベクトルデータに、例えば「犬」という単語が含まれている

```
[postgres@rocky9 ~]$ zcat cc.ja.300.vec.gz | sed -n 903p
犬 -0.2146 -0.2391 0.0720 0.1586 0.0262 0.2139 0.0414 0.1504 0.2698 0.2355 0.0339 0.0797
0.4562 0.0650 -0.3562 -0.1534 -0.0898 0.1315 -0.2982 0.0327 0.1018 -0.1499 -0.3323 -
(略)

0.1112 0.1538 0.4049 -0.1077 0.2075 -0.2512 0.1062 -0.0319 -0.2691 0.1189 -0.1704 -0.1565
-0.0541 0.1039 0.1737 0.2176 -0.0586 -0.2596 -0.0547 -0.1838 -0.2234 0.2169 -0.0021
0.3145 -0.2677
[postgres@rocky9 ~]$
```

スペース区切りのデータ

- 単語ベクトルデータには下記のような”単語”も含まれている
  - カンマや二重引用符が単語の一部として使われている

```
[postgres@rocky9 ~]$ zcat cc.ja.300.vec.gz | sed -n '5369p;25818p' | awk '{print $1}'  
'',  
,",  
,
```

カンマや二重引用符が含まれている例

- psql の¥copyコマンドで読み込むため、下記処理を実施
  - スペース区切りのデータを、単語とベクトルデータでタブ区切りに変換
  - ¥copyコマンドではデフォルト値から下記設定に変更
    - DELIMITER文字(区切り文字)をカンマからタブ
    - QUOTE文字を二重引用符からバックスペース

- 単語とベクトルデータでTAB区切りに変換して標準出力に出力する

## Pythonスクリプト

```
$ cat mktsv.py
import gzip

INPUT_FILE = "cc.ja.300.vec.gz"

with gzip.open(INPUT_FILE, "rt", encoding="utf-8") as f:
    # ヘッダ行スキップ
    next(f)
    for line in f:
        parts = line.rstrip().split()
        word = parts[0]
        vector = parts[1:]
        vec_str = "[" + ",".join(vector) + "]"
        print(f"{word}¥t{vec_str}")
```

- Pythonスクリプトの出力の確認

```
[postgres@rocky9 ~]$ python mktsv.py | sed -n 902p
犬          [-0.2146,-
0.2391,0.0720,0.1586,0.0262,0.2139,0.0414,0.1504,0.2698,0.2355,0.0339,0.0797,0.
4562,0.0650,-0.3562,-0.1534,-0.0898,0.1315,-0.2982,0.0327,0.1018,-0.1499,-
0.3323,-0.0690,0.0231,0.0225,-0.0954,0.1700,0.0996,0.1583,0.1552,-0.1448,-
0.1736,-0.0798,0.0191,
(略)
-0.0319,-0.2691,0.1189,-0.1704,-0.1565,-0.0541,0.1039,0.1737,0.2176,-0.0586,-
0.2596,-0.0547,-0.1838,-0.2234,0.2169,-0.0021,0.3145,-0.2677]
[postgres@rocky9 ~]$
```

- 単語と300次元のベクトルデータがタブ区切りで出力

- 単語ベクトルデータを登録するword\_embeddingテーブルを作成

```
testdb=> CREATE TABLE word_embedding (word text, embedding vector(300));  
CREATE TABLE  
testdb=>
```

- word 項目→単語文字列
- embedding 項目→ 300次元のベクトルデータ
- 標準出力のデータをword\_embedding テーブルに、CSVモードでcopyする

```
$ python mktsv.py | psql -U testuser -d testdb -c "¥copy word_embedding(word,  
embedding) FROM STDIN WITH (FORMAT csv, DELIMITER E'¥t', QUOTE E'¥b')"  
COPY 2000000  
$
```

DELIMITER文字(区切り文字)をカンマからタブ  
QUOTE文字を二重引用符からバックスペース

- 200万行の単語ベクトルデータが登録される

- word 項目に「犬」の文字列を含むものをLIKE検索で出力

```
testdb=> SELECT * FROM (SELECT word, embedding <=> (SELECT embedding FROM word_embedding WHERE word = '犬') AS distance FROM word_embedding WHERE word LIKE '%犬%' LIMIT 15) ORDER BY distance;
```

| word | distance            |
|------|---------------------|
| 犬    | 0                   |
| 飼い犬  | 0.29386458644211555 |
| 愛犬   | 0.3394016673188782  |
| 子犬   | 0.34997457291011136 |
| 仔犬   | 0.35545523543390556 |
| 柴犬   | 0.3645315747019586  |
| 猟犬   | 0.3718109961648757  |
| 番犬   | 0.39780196159139114 |
| 野良犬  | 0.43790815517052395 |
| 盲導犬  | 0.5108898104678992  |
| 狂犬病  | 0.5451759160868502  |
| 狛犬   | 0.5660464561998243  |
| 犬飼   | 0.5968548559394272  |
| 犬山   | 0.6391950750268807  |
| 負け犬  | 0.6570291002745876  |

(15 rows)

※「犬」の文字を含む単語が出力されているが、動物の「犬」とは関係がない単語も存在する

# ベクトルデータで類似検索

- embedding 項目が「犬」のモノとコサイン距離(<=>)に近い順に出力

```
testdb=> SELECT word, embedding <=> (SELECT embedding FROM word_embedding WHERE word = '犬') AS  
distance FROM word_embedding ORDER BY distance LIMIT 15;
```

| word   | distance            |
|--------|---------------------|
| 犬      | 0                   |
| 飼い犬    | 0.29386458644211555 |
| ワンコ    | 0.30949939025763584 |
| 飼い主    | 0.3367078166695586  |
| 愛犬     | 0.3394016673188782  |
| 猫      | 0.3494613660216169  |
| 子犬     | 0.34997457291011136 |
| 仔犬     | 0.35545523543390556 |
| 柴犬     | 0.3645315747019586  |
| ドーベルマン | 0.36693292505199593 |
| 猟犬     | 0.3718109961648757  |
| しば犬    | 0.3724911956844441  |
| 飼犬     | 0.3748259599879523  |
| チワワ    | 0.3870314058462826  |
| イヌ     | 0.3886612615445969  |

(15 rows)

「犬」の「飼い主」

「犬」と同様にペットとして人気の「猫」

「犬」の種類のひとつである「ドーベルマン」

- 「猫」「オオカミ」「象」「クジラ」「富士山」「数学」を「犬」に近い順に出力

```
testdb=> SELECT word, embedding <=> (SELECT embedding FROM word_embedding WHERE word = '犬') AS  
distance FROM word_embedding WHERE word IN ('犬', '猫', 'オオカミ', '象', 'クジラ', '富士山', '数  
学') ORDER BY distance;
```

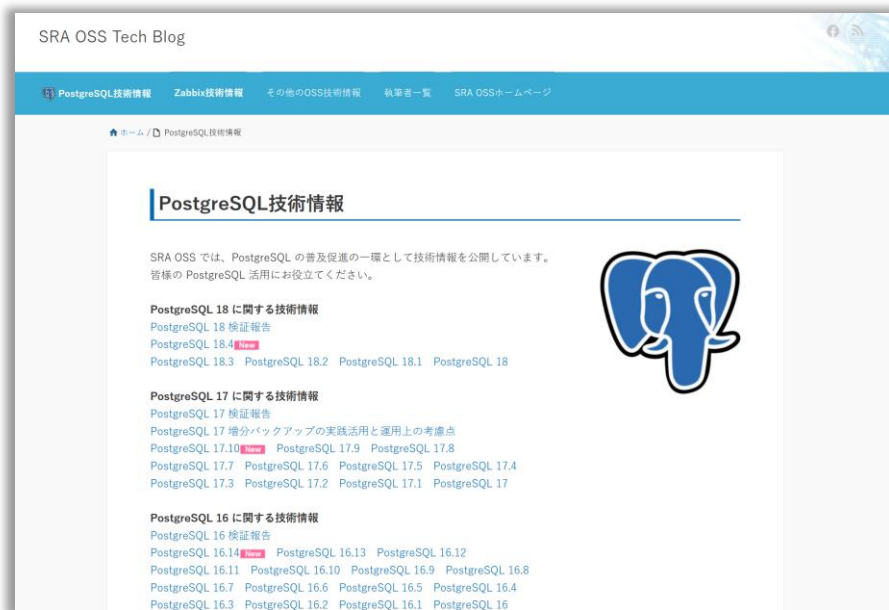
| word | distance           |
|------|--------------------|
| 犬    | 0                  |
| 猫    | 0.3494613660216169 |
| オオカミ | 0.5284062761527533 |
| 象    | 0.6512547480298522 |
| クジラ  | 0.7085417264456112 |
| 富士山  | 0.7515740417800283 |
| 数学   | 0.8187312494283205 |

(7 rows)

⇒上記の結果より、fastText のベクトルデータを利用する前提のもと、  
「犬」に近い単語は「猫」(0.349)、「オオカミ」(0.528)の順となった

- ベクトルを使えば「意味」を数値で表現することができる
- pgvectorを利用するとPostgreSQLでベクトル検索ができる
- LIKE検索では見つからない「意味的に近いデータ」を検索できる
- fastTextの学習済み単語ベクトルを利用して類似検索を実演した
- 生成AIやLLMなどの基盤技術として活用できる

-  SRA OSS Tech Blog
  - <https://www.sraoss.co.jp/tech-blog/>
  - PostgreSQLのリリースノートの日本語訳など様々なOSS技術情報を掲載中
-  SRA OSS 公式 Youtube チャンネル
  - <https://www.youtube.com/c/sraoss-official>
  - 過去のセミナー動画を公開中



ご清聴ありがとうございました