Effective Similarity Search
In PostgreSQL

Oleg Bartunov, Teodor Sigaev
Lomonosov Moscow State University
Agenda

- Introduction
- Search similar in PostgreSQL (smlar extension)
- Simple recommender system (MovieLens database)
Similarity?

- Texts (topic, lexicon, style,...)
- Blogs, sites (topic, community, purpose..)
- Shopping items
- Pictures (topic, color, style,...)
- Music - ~400 attributes!
- Books, Movies

Wikipedia has problem with 'similarity'
Similarity Estimation

- Experts estimation
  - hard to formalize, we'll not consider!

- Use attributes of content
  - Sets of attributes (Pandora uses x100 musicians to classify music content by ~400 attributes)

- By user's interests (collaboration filtering, CF)
  - Sets of likes/dislikes, ratings
Content-based similarity

- Text –
  - Fragmentation - \{fingerprints\}, \{lexems\}, \{n-grams\}
  - \{tags\}, \{authors\}, \{languages\}, ...

Similarity (S) – numerical measurement of sets intersection, eg. \{lexems\} && \{lexems\}

Combination, eg, linear combination - $\Sigma$ Weight*S
By user's interest

- Input data - \{user, item, rating\} matrix
  - Usually, just identifiers
  - Items can be of different kinds - songs, bars, books, movies, ...
  - Matrix is big and sparse
- Exploit wisdom of crowds to capture similarities between \textit{items}. 
• Typical online shop combines several kinds of recommender systems
  - Content-based: recommend cell phones if user is about to buy for cell phone
  - CF with Content filtering: recommend cell phone accessories, compatible to the cell phone
  - CF: Recommend flowers and necklace
By user's interest

- Again, similarity as intersection of sets:
  - User-user CF – \{item\} && \{item\}
    - Intersection of sets of interesting items to find similar users
    - Recommend items, which interested for similar users
  - Item-item CF– \{user\} && \{user\}
    - Intersection of sets of interested users to find similar items
    - Recommend items, similar to interested items
Summary

• Calculation of similarity in content-based and CF methods is reduced to calculation of sets intersection

• We need some similarity metric!

• How we can do this effectively in PostgreSQL?
Requirements

- Similarity should be $0 \leq S \leq 1$
  - $S \equiv 1$ - absolutely similar objects
  - Identity of objects is not mandatory!
  - $S \equiv 0$ for absolutely non-similar objects
- $S(A,B) = S(B,A)$ - symmetry
- Two objects are similar if $S(A,B) \geq S_{\text{threshold}}$
- $A \sim B$ and $A \sim C \neq B \sim C$
Designations

- $N_a, N_b$ - # of unique elements in arrays

- $N_u$ - # of unique elements of $N_a$ union $N_b$

- $N_i$ - # of unique elements of $N_a$ intersection $N_b$
Metrics

Jaccard:

\[ S(A,B) = \frac{N_i}{(N_a + N_b - N_i)} = \frac{N_i}{N_u} \]

- \( \sim N\times\log(N) \)
- Good for large arrays of comparable sizes
Cosine (Ochiai):

\[ S(A, B) = \frac{N_i}{\sqrt{N_a \times N_b}} \]

- \( \sim N\log(N) \)
- Good for large N
Issues

- Jaccard and Cosine are vulnerable to popular items – false similarity, noise
- Need to penalize popular items

**TF*IDF metrics:**
  - TF – frequency of element in an array
  - IDF – inverted frequency of element in all arrays
Smlar extension

Functions and Operations:

- `float4 smlar(anyarray, anyarray)`
- `anyarray % anyarray`

Configuration parameters:

- `smlar.threshold = float4`  
- `smlar.type = (tfidf, cosine)`  
- Set of options for TF*IDF
Extension smlar

=# select smlar('{0,1,2,3,4,5,6,7,8,9}'::int[], '{0,1}'::int[]);
   smlar
  ----------
    0.447214
(1 row)

SET smlar.threshold=0.6;
# select '{0,1,2,3,4,5,6,7,8,9}'::int[] % '{0,1}'::int[];
   ?column?
  ----------
    f
(1 row)

2/SQRT(10*2)=0.447214
Extension smlar

Supported any data type, which has default hash opclass

```sql
=# select smlar('{one,two,three,4,5}'::text[], '{two,three}'::text[]);
   smlar
```

```
     ---
   0.632456
```

```sql
=# select '{one,two,three,4,5}'::text[] % '{two,three}'::text[];
   ?column?
```

```
   ---
    t
```
Index support

Speedup anyarray % anyarray

- Btree, hash – not applicable
- GiST – Generalized Search Tree
- GIN - Generalized Inverted Index
GiST index

Inner page

• Array key → signature
• Bitwise OR of all descendants

Leaf page

Signature key (long array): 01000101000011
Array key (short array): {234, 553, 8234, 9742, 234}
Making a Signature

- Hash each element of array into int4 using default hash opclass for given data type
- Unique and sort
- For each element v of hashed array set 
  \((v \% \text{ length of signature})\)-th bit
An idea

Traversing we should follow subtrees which have UPPER bound of similarity GREATER than threshold

- We know everything about query
- Need upper estimation for intersection
- Need lower estimation for number of elements
What is a upper bound of length of the beard?

Speed of Light

* Age

?
Estimation for leaf sign (cosine)

{foo,bar} => {125,553}

# intersected bits as upper estimation of common elements of arrays
Estimation for leaf sign (cosine)

- Query: \{foo, bar\} hashed to \{124, 553\}
- Use \# intersected bits as upper estimation of common elements of arrays (several query's elements may mapped in the same bit)
- Use \# set bits as lower estimation of \(N_{\text{elem}}\)
  \[N_{\text{bits}} \leq N_{\text{elem}}\text{ because of collisions}\]

\[N_{\text{intersected}} / \sqrt{N_{\text{bits}} \times N_{\text{query}}} \geq \text{exact similarity}\]
Estimation for inner sign (cosine)

• Query: \{foor, bar\} hashed to \{124, 553\}

• \(N_{\text{intersected}} \geq \) original value (the same + signature is bitwise OR of all descendants)

• We don't have lower bound for number of elements, so use a \(N_{\text{intersected}}\) as estimation

\[
\frac{N_{\text{intersected}}}{\sqrt{N_{\text{intersected}} \times N_{\text{query}}}} =
\]

\[
\sqrt{\frac{N_{\text{intersected}}}{N_{\text{query}}}} \geq \text{exact similarity of any successor}
\]
GIN

- $N \text{ intersect} - \text{exact value}$
- $N \text{ intersect} \text{ as lower bound of } N \text{ elements}$
- We know everything about query

\[
\frac{N_{\text{intersected}}}{\sqrt{N_{\text{intersected}} \times N_{\text{query}}}} = \sqrt{\frac{N_{\text{intersected}}}{N_{\text{query}}}} \geq \text{exact similarity}
\]
Other features

- `float4 smlar( compositetype[], compositetype[], bool useIntersect )`
- `CREATE TYPE compositetype AS (id text, w float4);`
- GIN index
- TF*IDF metrics
- `float4 smlar( anyarray, anyarray, text Formula )`
- `text[] tsvector2textarray( tsvector )`
- `anyarray array_unique(anyarray)`
- `float4 inarray( anyarray, anyelement [, float4 found, float4 notfound])`
Availability

`git clone git://sigaev.ru/smlar.git`
TODO

- Index support for ratings
- Index optimizations
- GIN per row storage?
- TF*IDF speedup
Recommender Systems

- Recommender systems: eBay, Amazon, last.fm, Pandora,...
  - Content filtering – based on content attributes (Music Genome Project lists \(~400\) attributes)! Match attributes of content I like.
  - Collaborative filtering – based on preferences of many users
    - User-based, item-based
Recommender System

- We use item-item CF (more stable)
  - Similarity metric: cosine
- Input data from MovieLens
  - 1mln rates: 6000 users on 4000 movies
  - 10 mln rates: 72000 users on 10,000 movies
Recommender System

• Initial data:
  - movies(mid,title,genre,description)
  - rates(uid,mid,rate)

• Step 1: Transform ratings to likes
  u: r=1 if r>avg(rate)
  rates(uid,mid,like)

• Produce table
  ihu(itemid,{users}, {rates})
Recommender System

- Step 2. item-item matrix
- Precompute item-item matrix \( ii(itemid1, itemid2, sml) \) from ihu table
- Step 3. Evaluations
  - Q1: for given movie provide a list of similar movies
  - Q2: for given user provide a list of recommendations
Recommender System

• Step 1.
  - Produce table ihu (itemid, {users})
  - Create index to accelerate % operation

  CREATE INDEX ihu_users_itemid_idx ON ihu
  USING gist (users_int4_sml_ops, itemid);
**Step 2. Item-Item**

```
SELECT
    r1.itemid as itemid1,
    r2.itemid as itemid2,
    smlar(r1.users, r2.users) as sml
INTO ii
FROM
    ihu AS r1,
    ihu AS r2
WHERE
    r1.users % r2.users AND
    r1.itemid > r2.itemid;
```

Smlar.threshold=0.2
SELECT 209657
Index | no-index
526195 ms | 1436433
Speedup 2.7

Smlar.threshold=0.4
SELECT 8955
Index | no-index
253378 ms | 1172432
Speedup 4.6
Step 2. Item-Item

CREATE INDEX ii_itemid1_idx on ii(itemid1);
CREATE INDEX ii_itemid2_idx on ii(itemid2);

CREATE OR REPLACE VIEW ii_view AS
SELECT itemid1, itemid2, sml FROM ii
UNION ALL
SELECT itemid2, itemid1, sml FROM ii;
Step 3. Evaluations

CREATE OR REPLACE FUNCTION smlmovies(
    movie_id integer, num_movies integer,
    itemid OUT integer, sml OUT float, title OUT text)
RETURNS SETOF RECORD AS $$
SELECT s.itemid, s.sml::float, m.title
FROM movies m,
    ( SELECT itemid2 AS itemid, sml FROM ii_view
        WHERE itemid1 = movie_id
        UNION ALL
        SELECT movie_id, 1 -- just to illustration
    ) AS s
WHERE
    m.mid=s.itemid
GROUP BY s.itemid, rates, s.sml, m.title
ORDER BY s.sml DESC
LIMIT num_movies;
$$ LANGUAGE SQL IMMUTABLE;
### Step 3. Evaluations

```sql
=# select itemid, sml, title from smlmovies(1104,10);

<table>
<thead>
<tr>
<th>itemid</th>
<th>sml</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1104</td>
<td>1</td>
<td>Streetcar Named Desire, A (1951)</td>
</tr>
<tr>
<td>1945</td>
<td>0.436752468347549</td>
<td>On the Waterfront (1954)</td>
</tr>
<tr>
<td>1952</td>
<td>0.397110104560852</td>
<td>Midnight Cowboy (1969)</td>
</tr>
<tr>
<td>1207</td>
<td>0.392107665538788</td>
<td>To Kill a Mockingbird (1962)</td>
</tr>
<tr>
<td>1247</td>
<td>0.387987941503525</td>
<td>Graduate, The (1967)</td>
</tr>
<tr>
<td>2132</td>
<td>0.384177327156067</td>
<td>Who's Afraid of Virginia Woolf? (1966)</td>
</tr>
<tr>
<td>923</td>
<td>0.381125450134277</td>
<td>Citizen Kane (1941)</td>
</tr>
<tr>
<td>926</td>
<td>0.377328515052795</td>
<td>All About Eve (1950)</td>
</tr>
<tr>
<td>1103</td>
<td>0.363485038280487</td>
<td>Rebel Without a Cause (1955)</td>
</tr>
<tr>
<td>1084</td>
<td>0.356647849082947</td>
<td>Bonnie and Clyde (1967)</td>
</tr>
</tbody>
</table>

(10 rows)
```

Time: 5.780 ms
## Step 3. Evaluations

```sql
# select itemid, sml, title from smlmovies(364,10);
itemid |    sml    |                  title
--------+------------+----------------------------------------
      364 |          1 | Lion King, The (1994)
      595 | 0.556357622146606 | Beauty and the Beast (1991)
      588 | 0.547775387763977 | Aladdin (1992)
        1 | 0.472894549369812 | Toy Story (1995)
     2081 | 0.4552321434021 | Little Mermaid, The (1989)
     1907 | 0.442262977361679 | Mulan (1998)
     1022 | 0.41527932882309 | Cinderella (1950)
      594 | 0.407131761312485 | Snow White and the Seven Dwarfs (1937)
     2355 | 0.405456274747849 | Bug's Life, A (1998)
     2078 | 0.389742106199265 | Jungle Book, The (1967)
```

(10 rows)
### Step 3. Evaluations

```sql
=\# \text{select itemid, sml, title from smlmovies(919,10);}\\
\begin{array}{lll}
\text{itemid} & \text{sml} & \text{title} \\
919 & 1 & \text{Wizard of Oz, The (1939)} \\
260 & 0.495729923248291 & \text{Star Wars: Episode IV - A New Hope (1977)} \\
912 & 0.483502447605133 & \text{Casablanca (1942)} \\
1198 & 0.481675773859024 & \text{Raiders of the Lost Ark (1981)} \\
1196 & 0.468295514583588 & \text{Star Wars: Episode V - The Empire Strikes Back (1980)} \\
1028 & 0.460547566413879 & \text{Mary Poppins (1964)} \\
1097 & 0.455985635519028 & \text{E.T. the Extra-Terrestrial (1982)} \\
1247 & 0.449493944644928 & \text{Graduate, The (1967)} \\
858 & 0.446784257888794 & \text{Godfather, The (1972)} \\
594 & 0.44676461815834 & \text{Snow White and the Seven Dwarfs (1937)} \\
\end{array}
```

Time: 10.207 ms
```sql
CREATE TABLE myprofile (mid integer);
INSERT INTO myprofile VALUES
(912),(1961),(1210),(1291),(3148),(356),(919),(2943),(362),(2116);

=# select p.mid, m.title from movies m, myprofile p where m.mid=p.mid;
mid | title
---+-------------------------------------
912 | Casablanca (1942)
1961 | Rain Man (1988)
1210 | Star Wars: Episode VI - Return of the Jedi (1983)
1291 | Indiana Jones and the Last Crusade (1989)
356  | Forrest Gump (1994)
919  | Wizard of Oz, The (1939)
2943 | Indochine (1992)
2116 | Lord of the Rings, The (1978)
(10 rows)
```
SELECT t.itemid2 as itemid, t.sml::float, m.title
FROM movies m,
(
    WITH usermovies AS (
        SELECT mid  FROM myprofile
    ),
    mrec AS (
        SELECT itemid2, sml
        FROM ii_view ii, usermovies um
        WHERE
            ii.itemid1=um.mid  AND
            ii.itemid2 NOT IN ( SELECT *  FROM usermovies)
        ORDER BY itemid2 ASC
    )
    SELECT itemid2, sml,  rank()
    OVER (PARTITION BY itemid2 ORDER BY sml DESC) FROM mrec
) t
WHERE t.itemid2=m.mid AND t.rank = 1
ORDER BY t.sml DESC
LIMIT 10;
## Recommendations

<table>
<thead>
<tr>
<th>itemid</th>
<th>sml</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1196</td>
<td>0.71</td>
<td>Star Wars: Episode V - The Empire Strikes Back (1980)</td>
</tr>
<tr>
<td>260</td>
<td>0.67</td>
<td>Star Wars: Episode IV - A New Hope (1977)</td>
</tr>
<tr>
<td>1198</td>
<td>0.67</td>
<td>Raiders of the Lost Ark (1981)</td>
</tr>
<tr>
<td>1036</td>
<td>0.58</td>
<td>Die Hard (1988)</td>
</tr>
<tr>
<td>2571</td>
<td>0.57</td>
<td>Matrix, The (1999)</td>
</tr>
<tr>
<td>1240</td>
<td>0.56</td>
<td>Terminator, The (1984)</td>
</tr>
<tr>
<td>2115</td>
<td>0.56</td>
<td>Indiana Jones and the Temple of Doom (1984)</td>
</tr>
<tr>
<td>589</td>
<td>0.54</td>
<td>Terminator 2: Judgment Day (1991)</td>
</tr>
<tr>
<td>592</td>
<td>0.54</td>
<td>Batman (1989)</td>
</tr>
<tr>
<td>923</td>
<td>0.53</td>
<td>Citizen Kane (1941)</td>
</tr>
<tr>
<td>1270</td>
<td>0.53</td>
<td>Back to the Future (1985)</td>
</tr>
<tr>
<td>1197</td>
<td>0.52</td>
<td>Princess Bride, The (1987)</td>
</tr>
<tr>
<td>480</td>
<td>0.51</td>
<td>Jurassic Park (1993)</td>
</tr>
<tr>
<td>1200</td>
<td>0.51</td>
<td>Aliens (1986)</td>
</tr>
<tr>
<td>457</td>
<td>0.51</td>
<td>Fugitive, The (1993)</td>
</tr>
<tr>
<td>1374</td>
<td>0.50</td>
<td>Star Trek: The Wrath of Khan (1982)</td>
</tr>
<tr>
<td>2000</td>
<td>0.50</td>
<td>Lethal Weapon (1987)</td>
</tr>
<tr>
<td>2628</td>
<td>0.50</td>
<td>Star Wars: Episode I - The Phantom Menace (1999)</td>
</tr>
<tr>
<td>2028</td>
<td>0.49</td>
<td>Saving Private Ryan (1998)</td>
</tr>
<tr>
<td>1610</td>
<td>0.49</td>
<td>Hunt for Red October, The (1990)</td>
</tr>
</tbody>
</table>

(20 rows)
Recommender System

- This is a very simple recommender system!
- But it works!
- Recompute item-item if needed
  (10 mln ratings took <10 minutes on macbook)
- Need some content filtering, for example, categories matching
  (expert in movies may not be expert in cooking)
Content-based similarity

For each image
{
1. Scale ->
15x15
2. Array of intensities
}

\( smlar(arr1, arr2) \)
Content-based similarity

23.56% similarity
Thanks!