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 - Σ 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 *items*.

Similarity ?

- 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

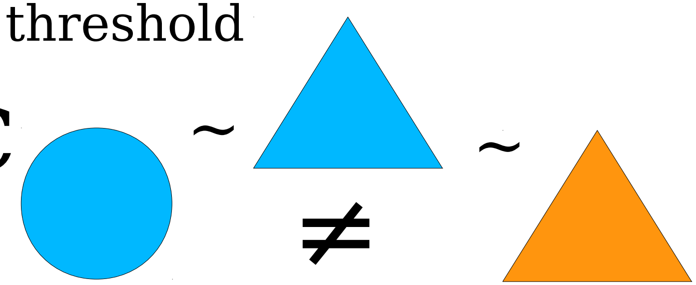
- Calculation of similarity in content-based and CF methods is reduced to calculation of sets intersection
- We need some similarity metric !
- How we can do this effectively in PostgreSQL?

Requirements

- Similarity should be $0 \le S \le 1$
  - $S \equiv 1$ – 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

$$S(A,B) \ge S_{threshold}$$

- $A \sim B$ and $A \sim C \ne B \sim C$



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:

$$S(A,B) = N_i / (N_a + N_b - N_i) = N_i / N_u$$

- ~ N*log(N)
- Good for large arrays of *comparable* sizes

Metrics

Cosine (Ochiai):

$S(A, B) = N_i / \mathrm{sqrt}(N_a * N_b)$

- ~ 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

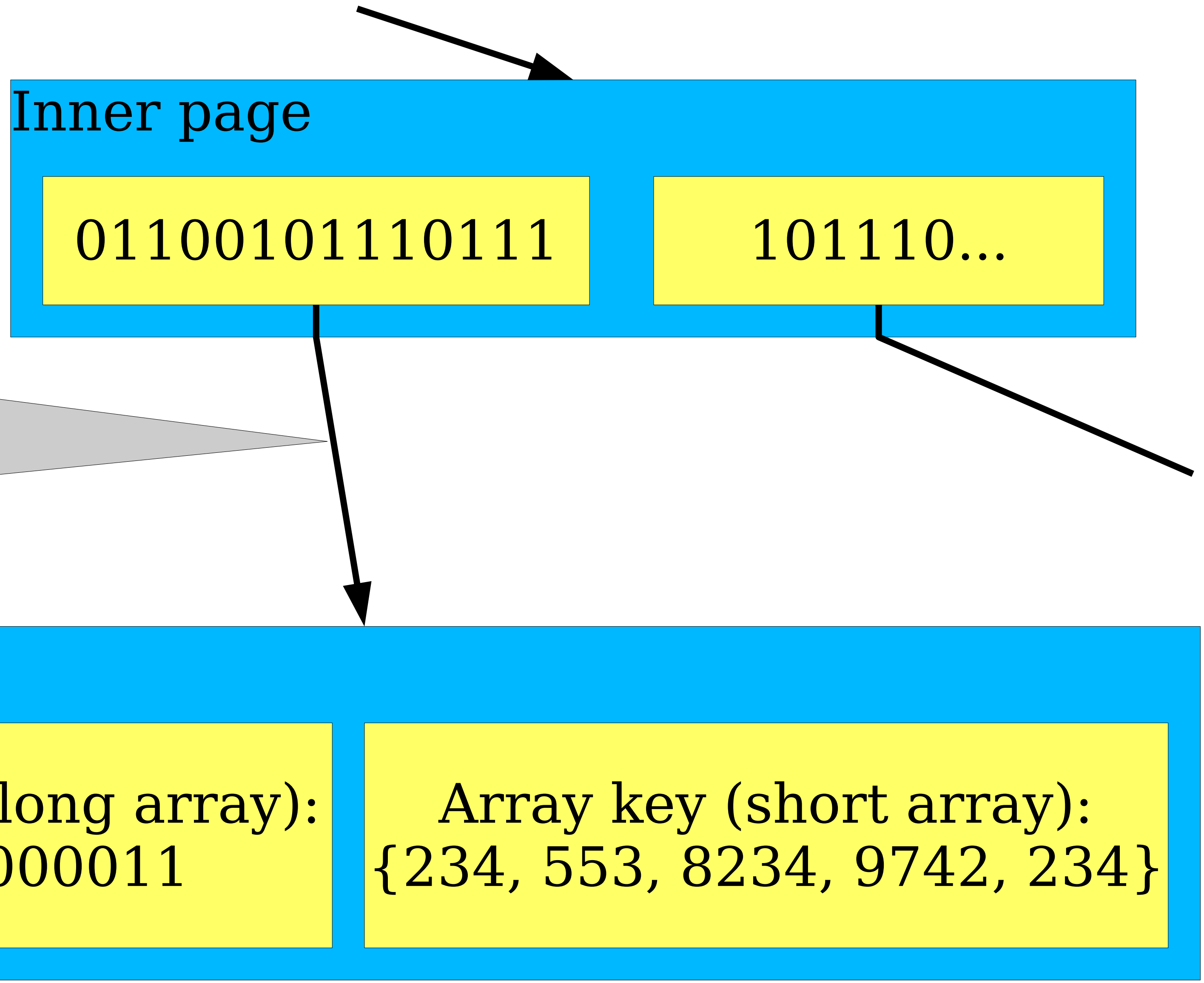
```
=# select smlar('{one,two,three,4,5}'::text[],
    '{two,three}'::text[]);
  smlar
----------
 0.632456
=# select '{one,two,three,4,5}'::text[] %
'{two,three}'::text[];
 ?column?
----------
 t
```

Index support

`Speedup anyarray % anyarray`

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

GiST index



- Array key → signature
- Bitwise OR of all descendants

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 % 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 upperl bound of length of the beard ?

Speed of Light

*****

Age

?

Estimation for leaf sign (cosine)

query

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

01**1**00101110111

key

2 vs **1**

125,234,355,401,450

original array

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_{elem}
($N_{bits} \leq N_{elem}$ because of collisions)

$$N_{intersected} / \mathrm{sqrt}(N_{bits} * N_{query}) \geq \text{exact similarity}$$

Estimation for inner sign (cosine)

- Query: {foor, bar} hashed to {124, 553}
- N intersected ≥ 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 intersected as estimation

$$N_{intersected} / \mathrm{sqrt}(N_{intersected} * N_{query}) =$$

$$\mathrm{sqrt}(N_{intersected} / N_{query}) \geq \text{exact similarity of any successor}$$

GIN

- N intersect – exact value
- N intersect as lower bound of N elements
- We know everything about query

$$N_{intersected} / \mathrm{sqrt}(N_{intersected} * N_{query}) =$$

$$\mathrm{SQRT}(N_{intersected} / N_{query}) \geq \text{exact similarity}$$

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

```
=# select itemid, sml,title from smlmovies(1104,10);
 itemid |        sml          |                    title
--------+-------------------+-----------------------------------------
   1104 |                   1 | Streetcar Named Desire, A (1951)
   1945 | 0.436752468347549 | On the Waterfront (1954)
   1952 | 0.397110104560852 | Midnight Cowboy (1969)
   1207 | 0.392107665538788 | To Kill a Mockingbird (1962)
   1247 | 0.387987941503525 | Graduate, The (1967)
   2132 | 0.384177327156067 | Who's Afraid of Virginia Woolf? (1966)
    923 | 0.381125450134277 | Citizen Kane (1941)
    926 | 0.377328515052795 | All About Eve (1950)
   1103 | 0.363485038280487 | Rebel Without a Cause (1955)
   1084 | 0.356647849082947 | Bonnie and Clyde (1967)
(10 rows)

Time: 5.780 ms
```

Step 3. Evaluations

```
# 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

```
=# select itemid, sml,title from smlmovies(919,10);
 itemid |        sml         |                          title
--------+--------------------+-----------------------------------------
    919 |                  1 | Wizard of Oz, The (1939)
    260 | 0.495729923248291 | Star Wars: Episode IV - A New Hope (197
    912 | 0.483502447605133 | Casablanca (1942)
   1198 | 0.481675773859024 | Raiders of the Lost Ark (1981)
   1196 | 0.468295514583588 | Star Wars: Episode V - The Empire Strik
   1028 | 0.460547566413879 | Mary Poppins (1964)
   1097 | 0.455985635519028 | E.T. the Extra-Terrestrial (1982)
   1247 | 0.449493944644928 | Graduate, The (1967)
    858 | 0.446784257888794 | Godfather, The (1972)
    594 |  0.44676461815834 | Snow White and the Seven Dwarfs (1937)
(10 rows)

Time: 10.207 ms
```

I like these movies

```
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)
 3148 | Cider House Rules, The (1999)
  356 | Forrest Gump (1994)
  919 | Wizard of Oz, The (1939)
 2943 | Indochine (1992)
  362 | Jungle Book, The (1994)
 2116 | Lord of the Rings, The (1978)
(10 rows)
```

Give me recommendations

```
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

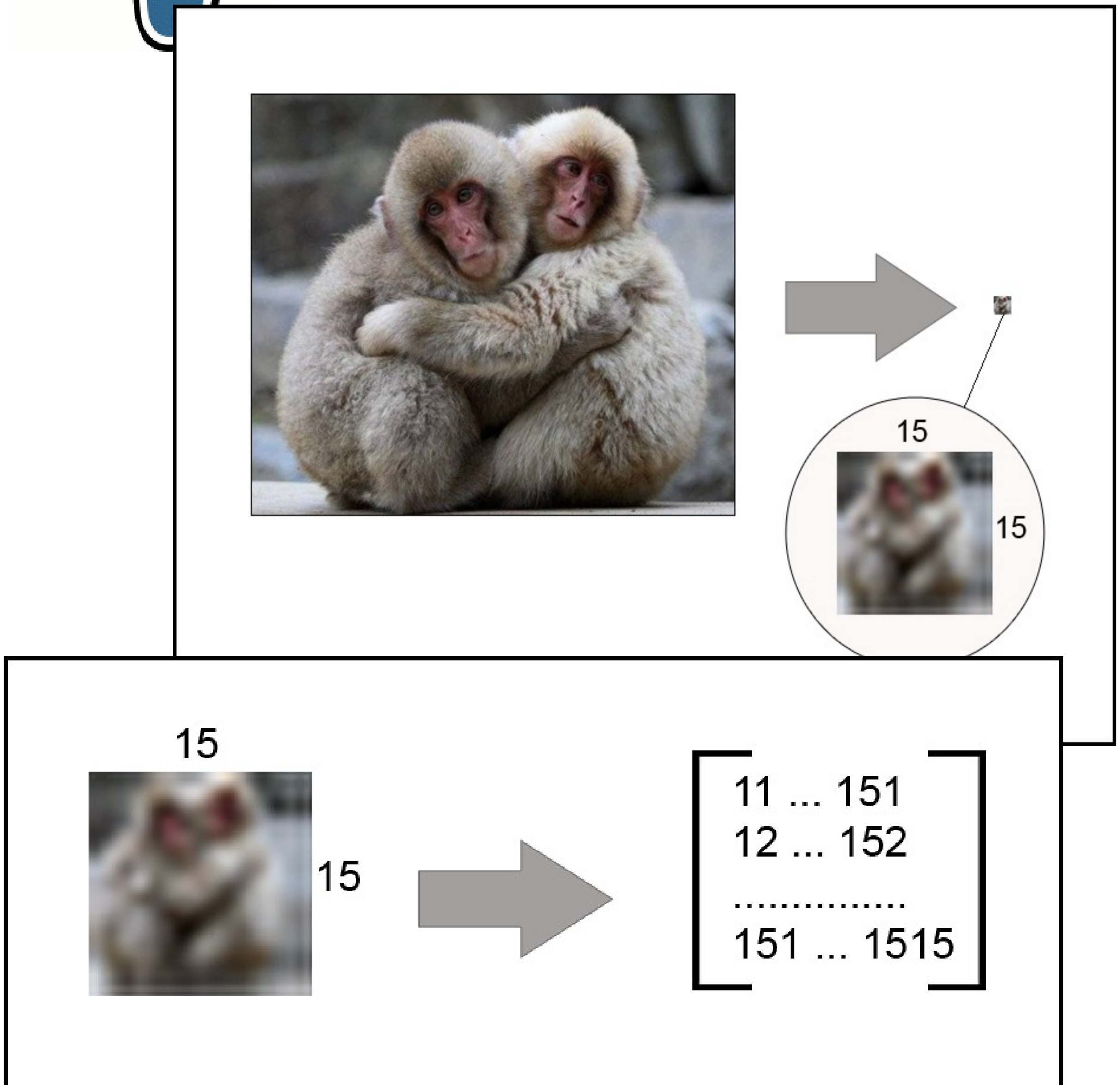
```
 itemid | sml  |                             title
--------+------+-------------------------------------------------------------------
   1196 | 0.71 | Star Wars: Episode V - The Empire Strikes Back (1980)
    260 | 0.67 | Star Wars: Episode IV - A New Hope (1977)
   1198 | 0.67 | Raiders of the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

Time: 207.965 ms
```

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)

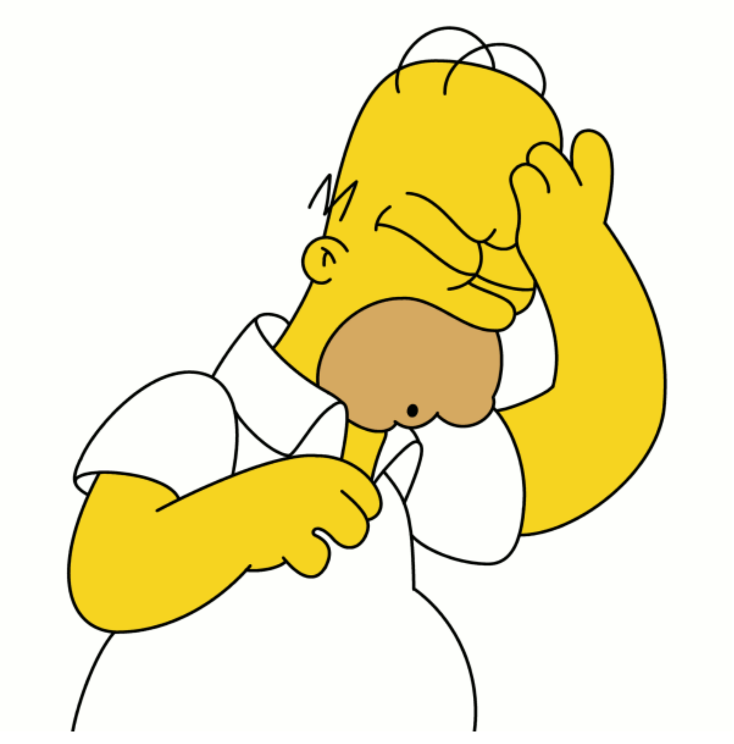
Content-based similarity



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

smlar(arr1,arr2)

Content-based similarity



23.56% similarity

Thanks !