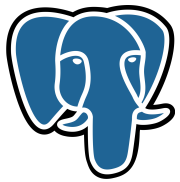
实时精准营销

(标签圈选、相似扩选、用户画像)



阿里云
digoal

目录

- 基于标签条件**圈选目标客户**
- 基于用户特征值**扩选相似**人群
- 群体**用户画像**分析

设计1

- KEY：userid
- VALUES：tagids
- index: GIN倒排
- search: 与、或、非、聚合

例子

标签阶梯化字典表

```
create table t_tag_dict (
tag int primary key,   -- 标签(人群)id
info text,  -- 人群描述
crt_time timestamp  -- 时间
);
```

生成10万标签（人群）

```
insert into t_tag_dict values (1, '男', now());
insert into t_tag_dict values (2, '女', now());
insert into t_tag_dict values (3, '大于24岁', now());
-- ...

insert into t_tag_dict select generate_series(4,100000),
md5(random()::text), clock_timestamp();
```

创建打标表

```
create table t_user_tags (
uid int8 primary key,    -- 用户id
tags int[],              -- 用户标签（人群）数组
mod_time timestamp       -- 时间
);
```

创建生成随机打标数组的函数

```
create or replace function gen_rand_tags(int,int) returns int[] as $$
  select array_agg(ceil(random()*$1)::int) from generate_series(1,$2);
$$ language sql strict;
```

```
db1=> select gen_rand_tags(100000, 63);
-[ RECORD 1 ]-+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
gen_rand_tags | {61263,18665,57580,71086,45358,16962,81930,32177,2948,37990,16159,76042,85201,16123,2606,42649,71959,75268,7612,46905,40053,55593,18563,79528,7430,49795,597,31519,46288,78602,52749,90676,75122,57761,54922,51444,74961,95001,68555,43986,83629,84311,6804,69957,3250,54395,43523,78221,80407,60576,86546,97052,50798,55786,9492,85369,66470,21976,60201,75971,74717,12527,43572}
```

给2000万用户打标

```
insert into t_user_tags select generate_series(1,10000000), array_append(gen_rand_tags(100000, 63),1), now();
insert into t_user_tags select generate_series(10000001,20000000), array_append(gen_rand_tags(100000, 63),2), now();
```

创建标签（人群）字段倒排索引

```
create index idx_t_user_tags_1 on t_user_tags using gin (tags);
```

查询包含1,3标签的人群

```
select count(uid) from t_user_tags where tags @> array[1,3];
select uid from t_user_tags where tags @> array[1,3];
```

查询包含1或3或10或200标签的人群

```
select count(uid) from t_user_tags where tags && array[1,3,10,200];
select uid from t_user_tags where tags && array[1,3,10,200];
```

设计2

- KEY: tagid
- VALUES: bitmap
- index: btree
- search: 与、或、非、聚合

roaringbitmap使用方法

- https://pgxn.org/dist/pg_roaringbitmap/0.5.0/

- RDS PG 12已支持：安装插件 – create extension roaringbitmap;
- bitmap输出格式 – set roaringbitmap.output_format='bytea|array';
- bitmap取值范围 – 40亿（int4）
- 构造bitmap – rb_build(int4[])
- 类型转换 - rb_to_array(rb) – rb_iterate(rb)
- bitmap内包含对象个数 – rb_cardinality(rb)

bit操作

- 逻辑运算
  - 与、或、异或、差
- 聚合运算
  - build rb、与、或、异或
  - 直接统计对象数(与、或、异或)
    - rb_or_cardinality_agg
    - rb_and_cardinality_agg
    - rb_xor_cardinality_agg
- 逻辑判断
  - 包含、相交、相等、不相等

例子

创建roaringbitmap插件

create extension roaringbitmap;

创建标签，用户bitmap表

```
create table t_tag_users (
  tagid int primary key,  -- 用户标签（人群）id
  uid_offset int,         -- 由于userid是int8类型，roaringbitmap内部使用int4存储，需要转换一下。
  userbits roaringbitmap,   -- 用户id聚合的 bitmap
  mod_time timestamp      -- 时间
);
```

生成标签，uid bitmap，

```
insert into t_tag_users
select tagid, uid_offset, rb_build_agg(uid::int) as userbits from
(
select
  unnest(tags) as tagid,
  (uid / (2^31)::int8) as uid_offset,
  mod(uid, (2^31)::int8) as uid
from t_user_tags
) t
group by tagid, uid_offset;
```

uid 超过int4，转换方法

https://github.com/digoal/blog/blob/master/202001/20200110_03.md

例子

查询包含1,3标签的人群

人群数量

```
select sum(ub) from
(
select uid_offset,rb_and_cardinality_agg(userbits) as ub
from t_tag_users
where tagid in (1,3)
group by uid_offset
) t;
```

人群ID

```
select uid_offset,rb_and_agg(userbits) as ub
from t_tag_users
where tagid in (1,3)
group by uid_offset;
```

查询包含1或3或10或200标签的人群

人群数量

```
select sum(ub) from
(
select uid_offset,rb_or_cardinality_agg(userbits) as ub
from t_tag_users
where tagid in (1,3,10,200)
group by uid_offset
) t;
```

人群ID

```
select uid_offset,rb_or_agg(userbits) as ub
from t_tag_users
where tagid in (1,3,10,200)
group by uid_offset;
```

方案1 VS 方案2

CASE(12.8亿 user_tags) (2000万, 64 tag per user)	方案1（user->tags）	方案2(tagid->user_bitmap)	优胜
与查询速度	42毫秒，44毫秒	1.5毫秒，1.5毫秒	方案2
或查询速度	3秒，13秒	1.5毫秒，1.7毫秒	方案2
空间占用（表）	3126MB	1390MB	方案2
空间占用（索引）	3139MB	2MB	方案2
build索引速度	20分钟	0秒	方案2

更新标签-放大问题优化

- 设计1更新标签放大问题优化
  - 批量合并，减少更新量

目录

- 基于标签条件**圈选目标客户**
- 基于用户特征值**扩选相似**人群
- 群体**用户画像**分析

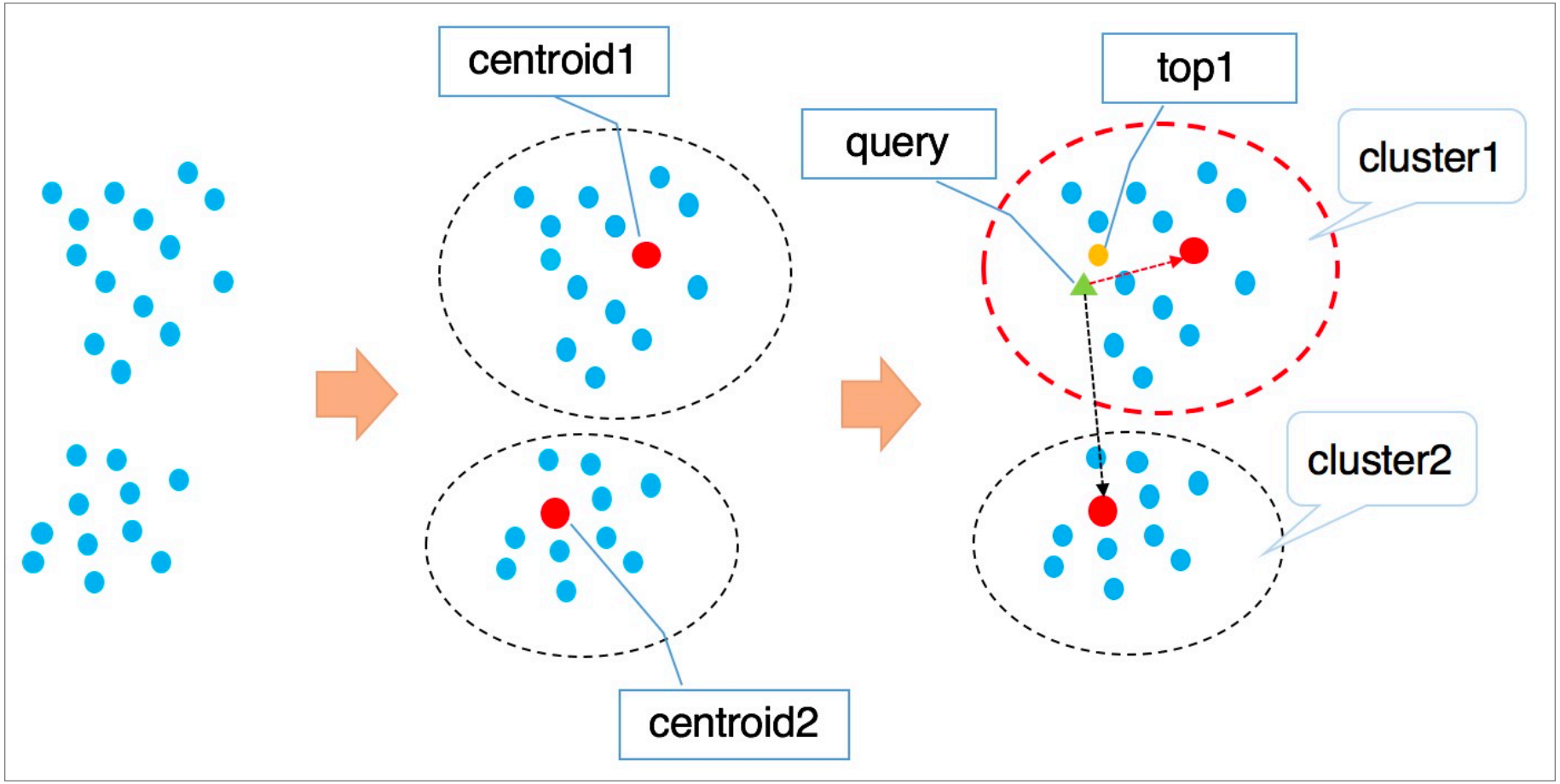
相似人群扩选

- 特征向量
  - 对象ID，对象特征向量
- 相似计算
  - 欧式距离
  - 内积距离

例子

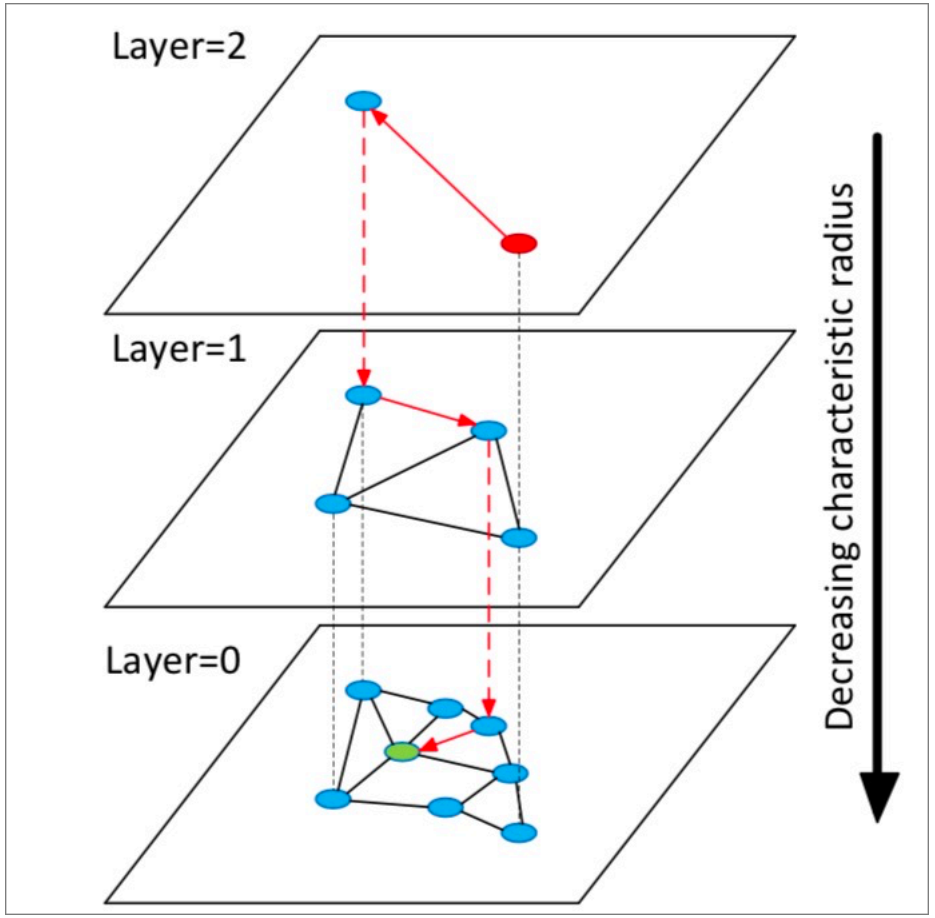
- 人群特征表
- build 向量索引
- 相似扩选

ivfflat -> 中心点收敛



hnsw -> 图层收敛

1、从概略图（最高层）选一个随机点开始搜索
找到离目标**绿色点**最近的点x
2、从x这个点的下一层开始搜索，找到离目标**绿色点**最近的点y。
3、重复1，2，直到最下层
4、在最下层找到离目标**绿色点**最近的点



例子

```
create extension pase;  -- 目前仅RDS PG 11支持，即将覆盖RDS 新版本
create table t_user_vec (uid serial8 primary key, vec float4[]);

create or replace function gen_float4_arr(int,int) returns float4[] as $$
select array_agg(trunc(random()*$1)::float4) from generate_series(1,$2);
$$ language sql strict volatile;

insert into t_user_vec (vec) select gen_float4_arr(10000,16) from generate_series(1,1000000);

create index idx_t_user_vec_1 on t_user_vec using pase_ivfflat (vec) with
(clustering_type = 1, distance_type = 0, dimension = 16, clustering_params = "10,1001");  -- 采样10/1000，生成1001个中心点

create index idx_t_user_vec_2 on t_user_vec using pase_hnsw (vec) with
(dim = 16, base_nb_num = 16, ef_build = 40, ef_search = 200, base64_encoded = 0);
```

db2=> select vec from t_user_vec limit 1;

{8669,3850,7236,3424,7704,159,1496,1981,3344,1208,4635,2011,7466,1832,7585,1553}

select uid,vec <#> '8680,3850,7236,3424,7704,159,1496,1981,3344,1209,4635,2011,7466,1832,7589,1553'::pase from t_user_vec order by vec <#> '8669,3850,7236,3424,7704,159,1496,1981,3344,1208,4635,2011,7466,1832,7585,1553'::pase limit 10; -- ivfflat

select uid,vec from t_user_vec order by vec <?> '.....'::pase limit 10; -- hnsw

```
db2=> select uid,vec <#> '8680,3850,7236,3424,7704,159,1496,1981,3344,1209,4635,2011,7466,1832,7589,1553'::pase   from t_user_vec order by vec <#> '8669,3850,7236,3424,7704,159,1496,1981,3344,12
08,4635,2011,7466,1832,7585,1553'::pase limit 10;    -- ivfflat
  uid   |  ?column?
--------+------------
      1 |         138
 555694 | 2.51917e+07
 261623 | 3.02557e+08
 263157 | 3.15618e+08
 762230 | 2.85415e+08
 761459 |  2.8758e+08
 907278 | 2.67914e+08
 908126 | 2.97016e+08
 878044 | 2.59149e+08
 878608 | 2.88415e+08
(10 rows)

Time: 2.719 ms
```

目录

- 基于标签条件**圈选目标客户**
- 基于用户特征值**扩选相似**人群
- 群体**用户画像**分析

用户画像例子

- 产生人群集合(新标签)
  - 方案2圈选结果: generate newtag, offset, uidbitmaps
- 针对集合人群进行透视

```
select 基础表 where uid = any(array(
    select offset*(2^31)::int8+unnest(rb_to_array(uidbitmaps))
))
```

  - 并行计算 (第7课内容)

参考资料

- 相似扩选
  - https://help.aliyun.com/document_detail/147837.html
  - https://github.com/digoal/blog/blob/master/201912/20191219_02.md
- 用户圈选
  - https://github.com/digoal/blog/blob/master/201911/20191118_01.md
  - https://pgxn.org/dist/pg_roaringbitmap/0.5.0/
- MySQL手册
  - https://www.mysqltutorial.org/
  - https://dev.mysql.com/doc/refman/8.0/en/
- PG 管理、开发规范
  - https://github.com/digoal/blog/blob/master/201609/20160926_01.md
- PG手册
  - https://www.postgresql.org/docs/current/index.html
  - https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-vs-mysql/
- GIS手册
  - http://postgis.net/docs/manual-3.0/

一期开课计划(PG+MySQL联合方案)

- - 2019.12.30 19:30 RDS PG产品概览，如何与MySQL结合使用
- - 2019.12.31 19:30 如何连接PG，GUI，CLI的使用
- - 2020.1.3 19:30 如何压测PG数据库、如何瞬间构造海量测试数据
- - 2020.1.6 19:30 MySQL与PG对比学习(面向开发者)
- - 2020.1.7 19:30 如何将MySQL数据同步到PG（DTS）
- - 2020.1.8 19:30 PG外部表妙用 - mysql_fdw, oss_fdw（直接读写MySQL数据、冷热分离）
- - 2020.1.9 19:30 PG应用场景介绍 - 并行计算，实时分析
- - 2020.1.10 19:30 PG应用场景介绍 - GIS
- - 2020.1.13 19:30 PG应用场景介绍 - 用户画像、实时营销系统
- - 2020.1.14 19:30 PG应用场景介绍 - 多维搜索
- - 2020.1.15 19:30 PG应用场景介绍 - 向量计算、图像搜索
- - 2020.1.16 19:30 PG应用场景介绍 - 全文检索、模糊查询
- - 2020.1.17 19:30 PG 数据分析语法介绍
- - 2020.1.18 19:30 PG 更多功能了解：扩展语法、索引、类型、存储过程与函数。如何加入PG技术社群

本课程习题

- 基于标签圈选用户有哪几种典型的设计
- 用户圈选不同设计方法的优劣
- 什么插件可以存储用户ID映射而成的比特流
- 多维向量之间的距离有哪些计算方法
- 多维向量按距离由近到远排序有什么加速方法
- pase插件已经支持了哪些向量搜索算法
- 用户画像透视有什么加速方法

技术社群

PG技术交流钉钉群(3500+人)