

Hadoop应用开发实战案例 第4周

【声明】 本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

<http://edu.dataguru.cn>

- 2009年基于用户地理位置信息的手机服务网站 Foursquare 上线
 - 允许用户上传自己的位置 (Self-reported positioning)
 - 利用手机的GPS功能帮助寻找自己的位置
- 两年内国内的街旁、玩转四方、切客、开开和嘀咕上线
- 2013年7月街旁发布新版本，转型搞社交
 - 至此以 LBS 签到作为独立模式的应用都已经退场或者转型



LBS的时机

- 随着智能手机的普及，GPS等传感器移动终端传感器成为标配
- 移动互联网带宽的提升

基于位置的服务（LBS）功能成为了现今应用的热点功能

- 社交网络
 - 新浪微博
 - 微信
- 生活服务
 - 大众点评
 - 滴滴打车

■ 积累海量的用户位置数据



● 位置页面 ● 景点介绍 ● 团购优惠 ● 相关微博 ● 地图



用户可以在自己的微博发送自己的所在位置

• 包括用户的正文和所在位置名称



根据自己的地理位置搜索周边商户

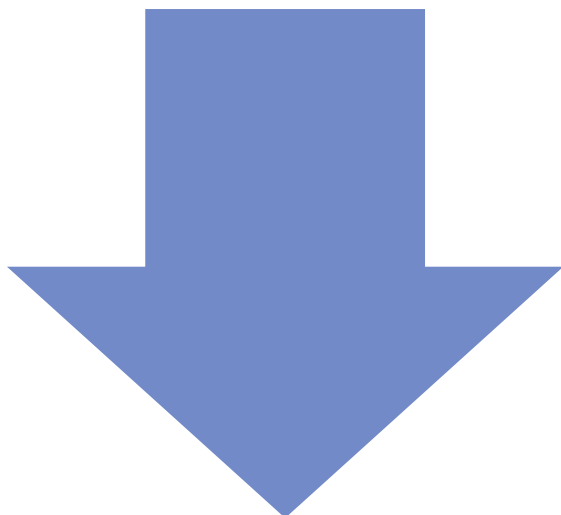


对光顾的商家进行签到



优势

- 在时间维度上数据更全面
- 不但有轨迹，还能获取用户停留信息



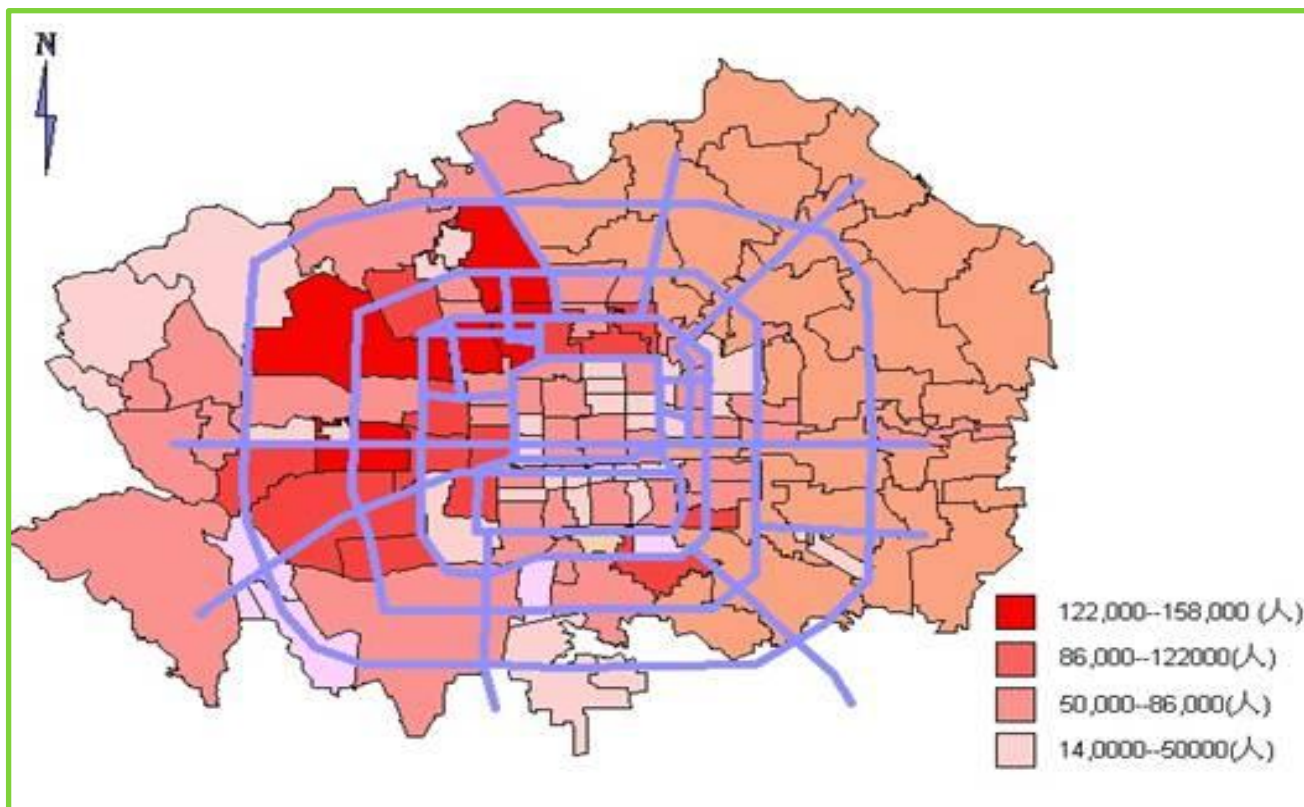
劣势

- 相比GPS在空间上可能不准确
- 可能无法确定用户移动的目的性

新店选址



土地利用



问题

- 空间定位并不准确
- 损失POI信息
- 位置的变更可能不代表移动

思考

- 关注区域用户流动
- 只考虑可以找出明确目的性的移动信息

挖掘基站停留数据



提取人们从一个时段到另一个时段的移动



分析地区间的用户流动



用户ID	时段	位置
001	2013-09-12 00-09	中山大学
001	2013-09-12 09-17	珠江新城
002	2013-10-10 09-17	广东省博物馆
...

五百万用户

- 三个月的基站停留数据



五千个基站

- 基站数据量不均匀



分时段统计最长停留

- 计算 09-17-24 三个时段

■ 输出

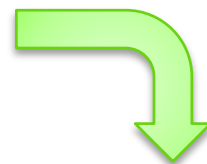
时间段	地点	下一个地点	概率
00-09	中山大学	广州信息港	0.01
00-09	珠江新城	珠江帝景	0.005
09-17	华南理工大学	广州图书馆新馆	0.001
...

- 凌晨时段在“中山大学”的人, 接下来可能去“广州信息港”的可能性是0.01
- 保留每个地点排名最高的3个预测地点

计算方法 – 提取连续位置

- 把同一个用户的连续两个时段的停留位置放在同一行，
 - 如果两次连续位置的间隔时间大于一天，则删去这一行

用户ID	时段	位置
001	2013-09-12 00-09	中山大学
001	2013-09-12 09-17	珠江新城
002	2013-10-10 09-17	广东省博物馆
...



用户ID	时段	位置	下一个时段	下一个位置
001	2013-09-12 00-09	中山大学	2013-09-12 09-17	珠江新城
...

- 去掉用户信息, 只考虑同一时段用户总体行为

时段	位置	下一个位置
00-09	中山大学	珠江新城
00-09	中山大学	广州信息港



计算方法 – 计算相对频数

■ 对于连续的两次停留位置, 使用相对频数近似转移概率

- $N(A, B)$: (A, B) 在连续位置出现的次数
- $N(A)$: A 在连续位置中作为起点的次数

$$f(B|A) = \frac{N(A, B)}{N(A)} = \frac{N(A, B)}{\sum_{B'} N(A, B')}$$

时段	位置	下一个位置
00-09	中山大学	广州信息港
00-09	中山大学	广州信息港
00-09	中山大学	珠江新城
00-09	中山大学	

$N(A = 'UA花城汇影院')$

$N(A, B)$
 $A = 'UA花城汇影院'$
 $B = '旧街场咖啡厅'$

不需要频繁计算

代码量尽量小

采用Pig脚本编程

利用PiggyBank
和Datafu避免
UDF开发

用户自定义函数 (UDF) 集

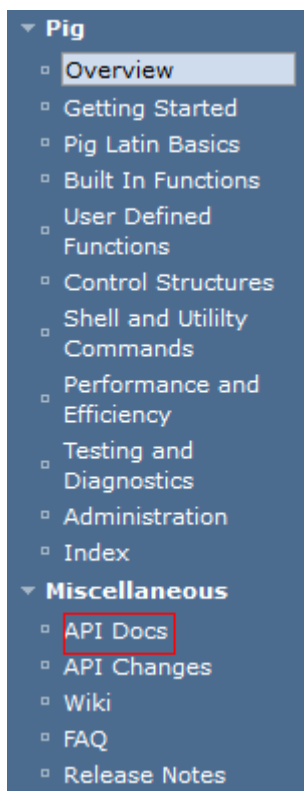
- 用户贡献的函数

包含十分有用的工具函数：

- Evaluation
 - 日期函数
 - 数学函数
 - 字符串处理
- Storage
 - CSV文件存取
 - XML文件存取
 - 分文件输出数据

Pig的Api文档 (0.12.0版本)

- <http://pig.apache.org/docs/r0.12.0/api/>



contrib: Piggybank

org.apache.pig.piggybank.evaluation

org.apache.pig.piggybank.evaluation.datetime

org.apache.pig.piggybank.evaluation.datetime.convert

org.apache.pig.piggybank.evaluation.datetime.diff

org.apache.pig.piggybank.evaluation.datetime.truncate

org.apache.pig.piggybank.evaluation.decode

org.apache.pig.piggybank.evaluation.math

org.apache.pig.piggybank.evaluation.stats

org.apache.pig.piggybank.evaluation.string

org.apache.pig.piggybank.evaluation.util

org.apache.pig.piggybank.evaluation.util.apachelogparser

org.apache.pig.piggybank.storage

org.apache.pig.piggybank.storage.allloader

org.apache.pig.piggybank.storage.apachelog

org.apache.pig.piggybank.storage.avro

org.apache.pig.piggybank.storage.hiverc

Pig的Api文档 (0.12.0版本)

- <http://pig.apache.org/docs/r0.12.0/api/>

```
public class CustomFormatToISO  
extends EvalFunc<String>
```

CustomFormatToISO converts arbitrary date formats to ISO format.

- Jodatetime: <http://joda-time.sourceforge.net/>
- ISO8601 Date Format: http://en.wikipedia.org/wiki/ISO_8601
- Jodatetime custom date formats: <http://joda-time.sourceforge.net/api-release/org/joda/time/format/DateTimeFormat.html>

Example usage: 函数用法示例

```
REGISTER /Users/me/commiter/piggybank/java/piggybank.jar ;  
REGISTER /Users/me/commiter/piggybank/java/lib/joda-time-1.6.jar ;
```

```
DEFINE CustomFormatToISO org.apache.pig.piggybank.evaluation.datetime.convert.CustomFormatToISO();  
CustomIn = LOAD 'test3.tsv' USING PigStorage('\t') AS (dt:chararray);
```

```
DESCRIBE CustomIn;  
CustomIn: {dt: chararray}
```

```
DUMP CustomIn;
```

```
(10-1-2010)
```

```
toISO = FOREACH CustomIn GENERATE CustomFormatToISO(dt, "MM-dd-YYYY") AS ISOTime:chararray;
```

Pig文件目录下 contrib/piggybank/java 目录下

从网上下载

手动编译

- <https://cwiki.apache.org/confluence/display/PIG/PiggyBank>

```
## 下载Pig
mkdir pig
cd pig
svn checkout http://svn.apache.org/repos/asf/pig/trunk/ .

## 编译Pig
ant

## 编译PiggyBank
cd contrib/piggybank/java
ant

## 此时 piggybank.jar 已经在当前目录下了
```

LinkedIn的用户自定义函数 (UDF) 集合

- 托管于 GitHub 的开源项目

包含一系列离线计算函数：

- 统计计算函数
 - 分位数/中位数, 方差
- 包 (Bag) 运算
 - 包的分割合并, 添加序号
- 链接分析 (Link Analysis)
 - PageRank.
- 地理函数
 - 计算两个经纬度的距离

官方页面

- <http://data.linkedin.com/opensource/datafu>

文档

- <http://linkedin.github.io/datafu/docs/current/>

Packages

<u>datafu.pig.bags</u>	A collection of general purpose UDFs for operating on bags.
<u>datafu.pig.geo</u>	UDFs for geographic computations.
<u>datafu.pig.hash</u>	UDFs for computing hashes from data.
<u>datafu.pig.linkanalysis</u>	UDFs for performing link analysis, such as PageRank.
<u>datafu.pig.random</u>	UDFs dealing with randomness.
<u>datafu.pig.sampling</u>	Sampling UDFs, including weighted sample, reservoir sampling, sampling by key, etc.
<u>datafu.pig.sessions</u>	UDFs for sessionizing data.
<u>datafu.pig.sets</u>	UDFs for set operations such as intersect and union.
<u>datafu.pig.stats</u>	Statistics UDFs for computing median, quantiles, variance, confidence intervals, etc.
<u>datafu.pig.urls</u>	UDFs for processing URLs.
<u>datafu.pig.util</u>	Other useful utilities.

官方页面

- <http://data.linkedin.com/opensource/datafu>

文档

- <http://linkedin.github.io/datafu/docs/current/>

`datafu.pig.geo`

Class HaversineDistInMiles

```
java.lang.Object
├─ org.apache.pig.EvalFunc<T>
│   └─ datafu.pig.util.SimpleEvalFunc<java.lang.Double>
│       └─ datafu.pig.geo.HaversineDistInMiles
```

```
public class HaversineDistInMiles
extends SimpleEvalFunc<java.lang.Double>
```

Computes the distance (in miles) between two latitude-longitude pairs using the [Haversine formula](#).

Example:

函数用法示例

```
-- input is a TSV of two latitude and longitude pairs
input = LOAD 'input' AS (lat1 : double, long1 : double, lat2 : double, long2 : double);
output = FOREACH input GENERATE datafu.pig.geo.HaversineDistInMiles(lat1, long1, lat2, long2) as distance;
```

- 使用 **LOAD** 关键字提取数据
- 由于使用 '|' 做分隔符, 因此调用 **PigStorage** 作为存取函数

```
grunt> data = LOAD 'data' USING PigStorage('|') AS ( imsi:chararray, time:chararray, loc:chararray );  
grunt> DESCRIBE data  
data: {imsi: chararray,time: chararray,loc: chararray}
```

- 基础数据格式 :

类型	说明	例子
int	Signed 32-bit integer	10
long	Signed 64-bit integer	Data: 10L or 10l Display: 10L
float	32-bit floating point	Data: 10.5F or 10.5f or 10.5e2f or 10.5E2F Display: 10.5F or 1050.0F
double	64-bit floating point	Data: 10.5 or 10.5e2 or 10.5E2 Display: 10.5 or 1050.0
chararray	Character array (string) in Unicode UTF-8 format	hello world

- 更新版本的 Pig 能够支持更多的函数

- 数据源文件的日期格式不规范
- 使用 PiggyBank 中的日期转换函数 CumstomFormatToISO 把日期转换成 ISO8601

Example usage:

```
REGISTER /Users/me/commiter/piggybank/java/piggybank.jar ;
REGISTER /Users/me/commiter/piggybank/java/lib/joda-time-1.6.jar ;

DEFINE CustomFormatToISO org.apache.pig.piggybank.evaluation.datetime.convert.CustomFormatToISO();
CustomIn = LOAD 'test3.tsv' USING PigStorage('\t') AS (dt:chararray);

DESCRIBE CustomIn;
CustomIn: {dt: chararray}

DUMP CustomIn;

(10-1-2010)
```

需要日期格式作为参数

```
toISO = FOREACH CustomIn GENERATE CustomFormatToISO(dt, "MM-dd-YYYY") AS ISOTime:chararray;

DESCRIBE toISO;
toISO: {ISOTime: chararray}

DUMP toISO;
(2010-10-01T00:00:00.000Z)
```

- 把时间段当时间点使用
- 使用 PiggyBank 中的日期转换函数把日期转换成 ISO8601 格式

```
grunt> -- 转换格式
grunt> REGISTER piggybank.jar;
grunt> REGISTER joda-time-1.6.jar
grunt> DEFINE CustomFormatToISO org.apache.pig.piggybank.evaluation.datetime.con
vert.CustomFormatToISO();
grunt>
grunt> toISO = FOREACH data GENERATE imsi, CustomFormatToISO( SUBSTRING(time,0,1
3), 'YYYY-MM-dd HH') AS time:chararray, loc;
```

- 当需要使用用户自定义函数时, 需要使用 **REGISTER** 关键字注册 jar 包
- 如果调用的函数太长, 可以使用 **DEFINE** 关键字对函数名定义一个别称
- toISO的部分输出结果 :

```
(002,2013-09-10T00:00:00.000Z,中山大学)
(002,2013-09-10T09:00:00.000Z,珠江新城)
(001,2013-09-12T00:00:00.000Z,中山大学)
(001,2013-09-12T09:00:00.000Z,广州信息港)
(002,2013-09-13T00:00:00.000Z,中山大学)
```

- 使用 **GROUP BY** 关键字把数据按照用户ID分组
- 具有相同 ID 的数据将会放入同一个 Bag 中

```
grunt> grp = GROUP toISO BY imsi;  
grunt> DESCRIBE grp  
grp: {group: chararray,toISO: {(imsi: chararray,time: chararray,loc: chararray)}}  
}
```

- 得到的模式包括两个字段：
 - ‘group’ : 等同于原来的分组关键字, 这里是 ‘imsi’
 - ‘toISO’ : 这是一个包(Bag), 具有相同 ‘imsi’ 的数据都被放在同一个包中. 里面的每一个元组(Tuple)的格式与分组前的toISO一样
- grp 的部分输出结果, 第一列代表imsi, 具有相同imsi的数据用大括号包含放在第二列

```
(001,{(001,2013-09-12T00:00:00.000Z,中山大学),(001,2013-09-12T09:00:00.000Z,广州  
信息港),(001,2013-10-10T00:00:00.000Z,中山大学),(001,2013-10-10T09:00:00.000Z,珠  
江新城)})  
(002,{(002,2013-09-10T00:00:00.000Z,中山大学),(002,2013-09-10T09:00:00.000Z,珠江  
新城),(002,2013-09-13T00:00:00.000Z,中山大学),(002,2013-09-13T09:00:00.000Z,广州  
信息港)})  
(003,{(003,2013-09-14T00:00:00.000Z,中山大学),(003,2013-09-14T09:00:00.000Z,广州  
信息港),(003,2013-09-14T17:00:00.000Z,体育中心)})
```

- 调用 Datafu 的 MarkovPairs 函数把连续位置放入到同一行

```
grunt> REGISTER datafu-1.0.0.jar
grunt> DEFINE MarkovPairs datafu.pig.stats.MarkovPairs();
grunt>
grunt> pairs = FOREACH grp
>> {
>> sorted = ORDER toISO BY time;
>> pair = MarkovPairs(sorted);
>> GENERATE FLATTEN(pair) AS ( data:tuple(imsi, time, loc), next:tuple(imsi, time, loc) );
>> }
grunt> DESCRIBE pairs
pairs: {data: (imsi: chararray,time: chararray,loc: chararray),next: (imsi: chararray,time: chararray,loc: chararray)}
```

- **FOREACH** 关键字在 GENERATE 前可以进行一些简单的操作, 例如排序
- 由于 MarkovPairs 函数的输出结果是一个包(Bag), 因此需要 FLATTEN 关键字做解套
- 最后产生的数据有两个字段:
 - 'data' : 这是一个元组(Tuple), 包括 imsi, time, loc 三个字段, 可以看做原始数据
 - 'next' : 这是一个元组(Tuple), 包括 imsi, time, loc 三个字段, 由于 MarkovPairs 函数的输入数据按照时间排序, 因此这里的 next 元组实际上就是 data 签到的下一条位置记录

- 如果一行的某个字段是元组, 提取元组中的数据有两种方法:
 - 如果需要提取整个元组的所有数据, 直接使用 **FLATTEN** 关键字解嵌套
 - 如果需要提取元组的某个字段, 使用 '元组名.字段名', 中间用点号连接

```
grunt> -- 展开数据
grunt> prj = FOREACH pairs GENERATE data.imsi AS imsi, data.time AS time, next.t
ime AS next_time, data.loc AS loc, next.loc AS next_loc;
```

- prj 的部分输出结果, 连续数据被放在同一行, 存在时间不邻接的数据

```
(001,2013-09-12T00:00:00.000Z,2013-09-12T09:00:00.000Z,中山大学,广州信息港)
(001,2013-09-12T09:00:00.000Z,2013-10-10T00:00:00.000Z,广州信息港,中山大学)
(001,2013-10-10T00:00:00.000Z,2013-10-10T09:00:00.000Z,中山大学,珠江新城)
(002,2013-09-10T00:00:00.000Z,2013-09-10T09:00:00.000Z,中山大学,珠江新城)
(002,2013-09-10T09:00:00.000Z,2013-09-13T00:00:00.000Z,珠江新城,中山大学)
(002,2013-09-13T00:00:00.000Z,2013-09-13T09:00:00.000Z,中山大学,广州信息港)
(003,2013-09-14T00:00:00.000Z,2013-09-14T09:00:00.000Z,中山大学,广州信息港)
(003,2013-09-14T09:00:00.000Z,2013-09-14T17:00:00.000Z,广州信息港,体育中心)
```


过滤不在同一天的数据

- 调用 Piggybank 的 **时间差函数** 计算两者的时间差
- 把时间间隔大于12小时的数据删去

```
grunt> DEFINE ISODaysBetween org.apache.pig.piggybank.evaluation.datetime.diff.ISODaysBetween();
grunt> flt = FILTER prj BY ISODaysBetween(next_time, time) == 0L;
grunt> DESCRIBE flt
flt: {id: chararray,time: chararray,next_time: chararray,loc: chararray,next_loc: chararray}
```

- flt 的部分输出结果

```
(001,2013-09-12T00:00:00.000Z,2013-09-12T09:00:00.000Z,中山大学,广州信息港)
(001,2013-10-10T00:00:00.000Z,2013-10-10T09:00:00.000Z,中山大学,珠江新城)
(002,2013-09-10T00:00:00.000Z,2013-09-10T09:00:00.000Z,中山大学,珠江新城)
(002,2013-09-13T00:00:00.000Z,2013-09-13T09:00:00.000Z,中山大学,广州信息港)
(003,2013-09-14T00:00:00.000Z,2013-09-14T09:00:00.000Z,中山大学,广州信息港)
(003,2013-09-14T09:00:00.000Z,2013-09-14T17:00:00.000Z,广州信息港,体育中心)
```

- 对于每一个位置, 计算其作为起点的数据条数
- 为了代码的简洁性, **GROUP BY** 关键字可以嵌套入 **FOREACH** 里面

```
grunt> total_count = FOREACH ( GROUP flt BY loc ) GENERATE group AS loc, COUNT(flt) AS total;  
grunt> DESCRIBE total_count  
total_count: {loc: chararray,total: long}
```

- 为了计算一个位置在某个位置之后出现的概率, 计算连续位置对 (Pair) 的数目

```
grunt> pairs_count = FOREACH ( GROUP flt BY (loc, next_loc) )  
>> GENERATE FLATTEN(group) AS (loc, next_loc), COUNT(flt) AS cnt;  
grunt> DESCRIBE pairs_count  
pairs_count: {loc: chararray,next_loc: chararray,cnt: long}
```

- **GROUP BY** 关键字可以使用多个字段分组, 此时分组的字段实际上是多个字段组成的元组(Tuple)
 - 因此分组结果中的 **group** 字段实际上也是一个元组, 需要使用**FLATTEN**关键字展开

- 按照 loc 字段进行连接以获得 $N(A, B)$ 和 $N(A)$, 从而计算相对频数
- Pig 提供多种表连接的实现方式
 - **Standard Joins** : 普通的hash Join, 分为 Inner Join 和 [Left | Right | FULL] Outer Join
 - **Replicated Joins** : 把其中一份 (或多份) 数据分配到每一台机器的内存中, 在Map端连接
 - **Skewed Joins** : 如果连接关键字在表中的分布不均匀, 则需要把无法载入内存的某个取值分片
 - **Merge Joins** : 假如连接的两个表都已经按照关键字排好序, 则能够以索引一个表的方式连接
- 可以认为 total_count 的结果足够小, 因此这里使用 Replicated Join, 此时必须把较大的表放在前面

```
grunt> jnd = JOIN pairs_count BY loc, total_count BY loc USING 'replicated';
grunt> DESCRIBE jnd
jnd: {pairs_count::loc: chararray,pairs_count::next_loc: chararray,pairs_count::
cnt: long,total_count::loc: chararray,total_count::total: long}
```

- 按照 loc 字段进行连接以获得 $N(A, B)$ 和 $N(A)$, 从而计算相对频数
- 可以认为 total_count 的结果足够小, 因此这里使用 Replicated Join, 此时必须把较大的表放在前面

```
grunt> jnd = JOIN pairs_count BY loc, total_count BY loc USING 'replicated';  
grunt> DESCRIBE jnd  
jnd: {pairs_count::loc: chararray,pairs_count::next_loc: chararray,pairs_count::  
cnt: long,total_count::loc: chararray,total_count::total: long}
```

- 在Pig中, Join之后的结果的字段名为了避免歧义都是用了 '::' 字符消歧义
 - 输出结果模式的字段都是 “原关系名::字段名”
- jnd 的输出结果, 可以看到连接关键字 'loc' 重复了

```
(中山大学,珠江新城,2,中山大学,5)  
(中山大学,广州信息港,3,中山大学,5)  
(广州信息港,体育中心,1,广州信息港,1)
```

计算相对频数并保留最高的三个可能

- 使用 FOREACH 嵌套找出概率最高的三个预测位置

```
grunt> prob = FOREACH jnd GENERATE pairs_count::loc AS loc, pairs_count::next_lo  
c AS next_loc, (double)cnt / (double)total AS probability;  
grunt> DESCRIBE prob  
prob: {loc: chararray,next_loc: chararray,probability: double}  
grunt>  
grunt> top3 = FOREACH ( GROUP prob BY loc )  
>> {  
>> sorted = ORDER prob BY probability DESC;  
>> top = LIMIT sorted 3;  
>> GENERATE FLATTEN(top);  
>> }  
grunt> DESCRIBE top3  
top3: {top::loc: chararray,top::next_loc: chararray,top::probability: double}
```

- 使用 **STORE INTO** 关键字把结果存入到 HDFS 文件系统
 - 此时将会产生一个或者多个 MapReduce 任务

```
grunt> STORE top3 INTO 'output';
```

- 最后的输出结果

```
grunt> cat output
中山大学      广州信息港      0.6
中山大学      珠江新城        0.4
广州信息港    体育中心        1.0
```

- **Dataguru（炼数成金）是专业数据分析网站，提供教育，媒体，内容，社区，出版，数据分析业务等服务。我们的课程采用新兴的互联网教育形式，独创地发展了逆向收费式网络培训课程模式。既继承传统教育重学习氛围，重竞争压力的特点，同时又发挥互联网的威力打破时空限制，把天南地北志同道合的朋友组织在一起交流学习，使到原先孤立的学习个体组合成有组织的探索力量。并且把原先动辄成千上万的学习成本，直线下降至百元范围，造福大众。我们的目标是：低成本传播高价值知识，构架中国第一的网上知识流转阵地。**
- **关于逆向收费式网络的详情，请看我们的培训网站 <http://edu.dataguru.cn>**



Thanks

FAQ时间