Locality Sensitive Hashing

in a few lines of SQL

Christopher Strecker

January 13, 2015

Outline

Finding Similar Objects

Useful for many things:

- Recommendations
- Duplicate items
- Fraud detection
- Clustering
- ...

Problem: "Obvious" algorithm needs $O(n^2)$ comparisons

- However: exact results often not required

$\Rightarrow$ LSH finds (configurable good) approximation in $O(n)$

Runtime Comparison

Back-of-the-envelope calculation

- Compare every objects with every object
- Or calculate one hash per object
- Assuming $\frac{1}{10^6}$s per comparison or $\frac{1}{10^3}$s per hash

	"Obvious" algorithm		LSH
# Objects	**# Comparisons**	Runtime	Runtime
10 000	50 Million	50 seconds	10 seconds
100 000	5 Billion	1.4 hours	1.6 minutes
1 000 000	500 Billion	6 days	16 minutes
10 000 000	50 Trillion	1.6 years	2.8 hours
100 000 000	5 Quadrillion	158 years	28 hours

Locality Sensitive Hashing

General idea

- Find “similar” numbers (within range of ± 5)

$$\{22, 32, 71, 77, 20, 69, 59, 55, 43, 61, 50, 63, 46, 38, 24, 44\}$$

Locality Sensitive Hashing

General idea

- Find "similar" numbers (within range of ± 5)

$$\{22, 32, 71, 77, 20, 69, 59, 55, 43, 61, 50, 63, 46, 38, 24, 44\}$$

- $h(x) = \text{round}\left(\frac{x}{10}\right) \cdot 10$

Bucket	Elements
20	$\{22, 20, 24\}$
30	$\{32\}$
40	$\{43, 38, 44\}$
50	$\{50, 46\}$
60	$\{59, 55, 61, 63\}$
70	$\{71, 69\}$
80	$\{77\}$

```
SELECT
  round(x / 10.0) * 10,
  array_agg(x)
FROM random_integers
GROUP BY
  round(x / 10.0) * 10
```

Locality Sensitive Hashing

General idea

- Find "similar" numbers (within range of ± 5)

$$\{22, 32, 71, 77, 20, 69, 59, 55, 43, 61, 50, 63, 46, 38, 24, 44\}$$

- $h(x) = \text{round}\left(\frac{x}{10}\right) \cdot 10$

Bucket	Elements
20	$\{22, 20, 24\}$
30	$\{32\}$
40	$\{43, 38, 44\}$
50	$\{50, 46\}$
60	$\{59, 55, 61, 63\}$
70	$\{71, 69\}$
80	$\{77\}$

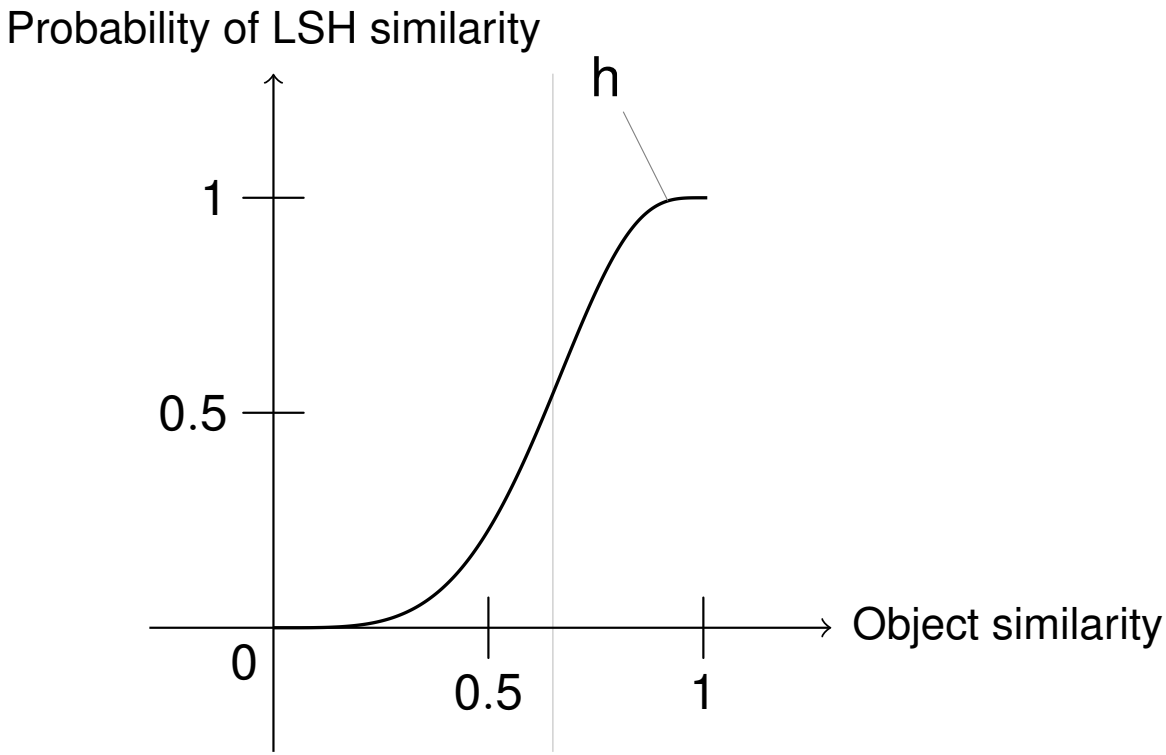
- $g(x) = \left\lfloor \frac{x}{10} \right\rfloor \cdot 10 + 5$

Bucket	Elements
25	$\{22, 20, 24\}$
35	$\{32, 38\}$
45	$\{43, 46, 44\}$
55	$\{59, 55, 50\}$
65	$\{69, 61, 63\}$
75	$\{71, 77\}$

Locality Sensitive Hashing

Improving the approximation

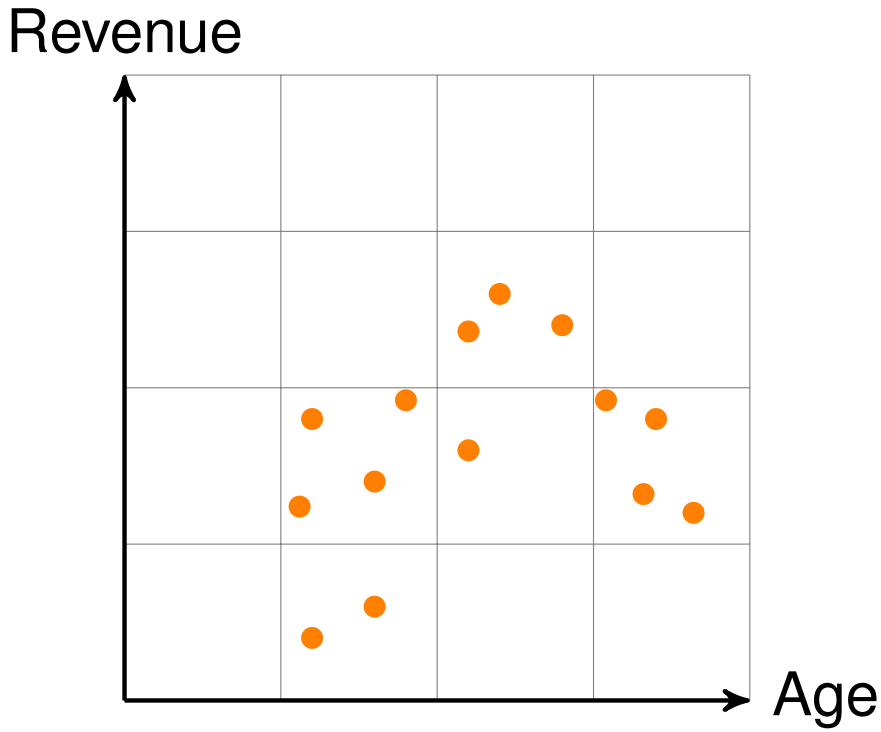
- Use a number of different hash functions h_1, h_2, h_3, h_4
- $h = (h_1 \text{ AND } h_2) \text{ OR } (h_3 \text{ AND } h_4)$



Euclidean Distance

“The” Distance

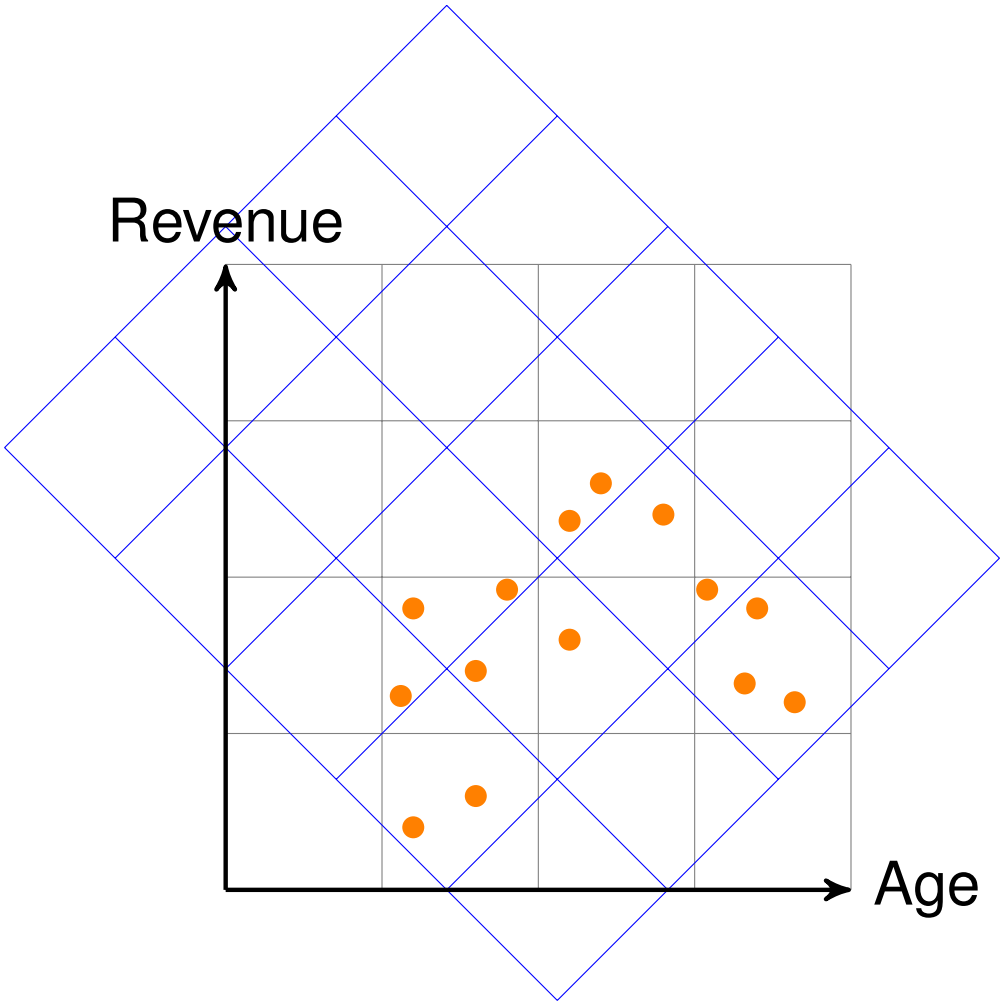
```
SELECT
    array_agg(customer_id)
FROM customers
GROUP BY
    h(age),
    h(revenue)
```



Euclidean Distance

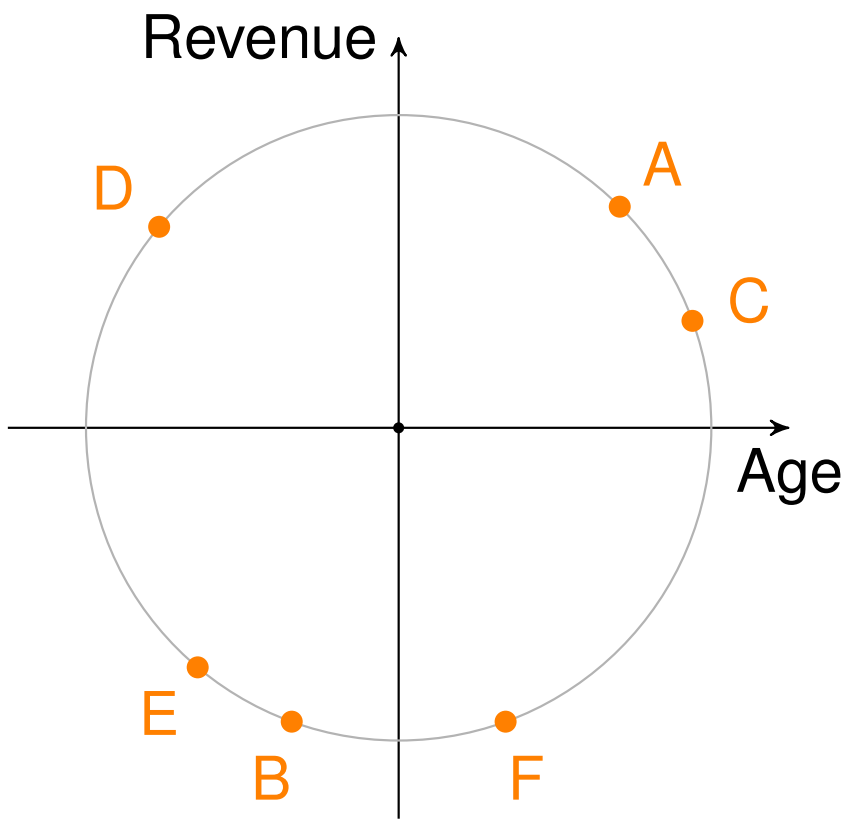
“The” Distance

```
SELECT
   array_agg(customer_id)
FROM customers
GROUP BY
   h(age),
   h(revenue)

UNION

SELECT
   array_agg(customer_id)
FROM customers
GROUP BY
   h(π(age)),
   h(π(revenue))
```



Cosine Distance

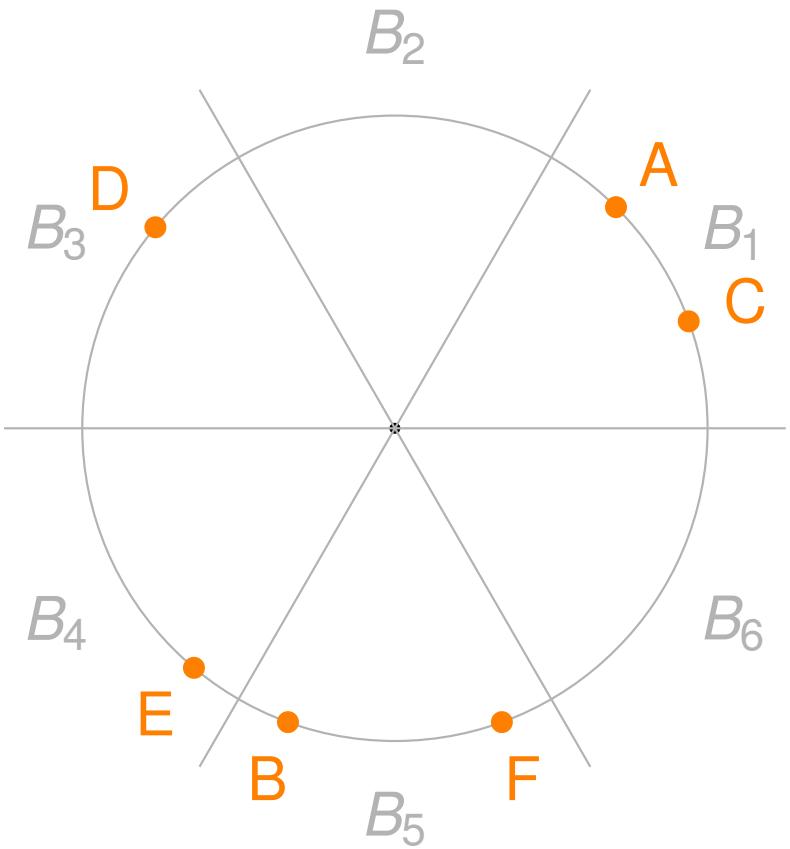
Vector Similarity



Cosine Distance

Vector Similarity

Bucket	Elements
B_1	{A, C}
B_2	{}
B_3	{D}
B_4	{E}
B_5	{B, F}
B_6	{}




```
SELECT
    array_agg(customer_id)
FROM customers
GROUP BY
    angle([1,0], [age,revenue]) / 60
```

Conclusions
More data than science

- Object similarity: “simple” problem, interesting applications
- Comparing everything with everything is difficult to scale
- Approximations are (probably) OK!
- LSH: much faster, quite easy to implement

References

Ideas from chapter 3 of the book

- Mining of Massive Datasets
  by Jure Leskovec, Anand Rajaraman, Jeffrey D. Ullman
- Online: `http://www.mmds.org/`
- MOOC at Coursera:
  `https://www.coursera.org/course/mmds`

Jaccard Similarity

Set Distance

	Items					
Cart	1	2	3	4	5	6
A	✓	-	✓	-	-	-
B	-	✓	✓	✓	-	-
C	-	-	-	-	✓	✓
D	✓	-	✓	✓	-	-

Jaccard Similarity

Set Distance

	Items					
Cart	1	2	3	4	5	6
A	✓	-	✓	-	-	-
B	-	✓	✓	✓	-	-
C	-	-	-	-	✓	✓
D	✓	-	✓	✓	-	-

$h_w = [4, 1, 2, 6, 5, 3]$

Cart	4	1	2	6	5	3
A	-	✓	-	-	-	✓
B	✓	-	✓	-	-	✓
C	-	-	-	✓	✓	-
D	✓	✓	-	-	-	✓

	h_w
h(A)	1
h(B)	4
h(C)	6
h(D)	4

Jaccard Similarity

Set Distance

	Items					
Cart	1	2	3	4	5	6
A	✓	-	✓	-	-	-
B	-	✓	✓	✓	-	-
C	-	-	-	-	✓	✓
D	✓	-	✓	✓	-	-

$h_w = [4, 1, 2, 6, 5, 3]$

```
SELECT
    cart_fk ,
    min(h_w(item_fk))
FROM item_in_cart
GROUP BY cart_fk
```

	h_w
h(A)	1
h(B)	4
h(C)	6
h(D)	4

Jaccard Similarity

Set Distance

	Items					
Cart	1	2	3	4	5	6
A	✓	-	✓	-	-	-
B	-	✓	✓	✓	-	-
C	-	-	-	-	✓	✓
D	✓	-	✓	✓	-	-

$h_w = [4, 1, 2, 6, 5, 3]$ $h_x = [4, 3, 5, 1, 6, 2]$ $h_y = [6, 1, 5, 4, 3, 2]$ $h_z = [3, 2, 6, 4, 5, 1]$

```
SELECT
    cart_fk,
    min(hw(item_fk)),
    min(hx(item_fk)),
    min(hy(item_fk)),
    min(hz(item_fk))
FROM item_in_cart
GROUP BY cart_fk
```

	h_w	h_x	h_y	h_z
h(A)	1	3	1	3
h(B)	4	4	4	3
h(C)	6	5	6	6
h(D)	4	4	1	3