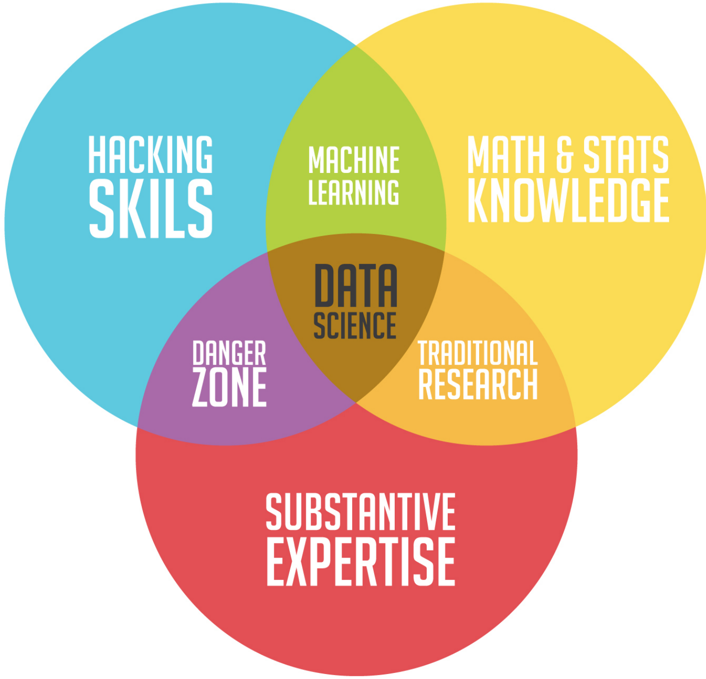
Finding Similar Items in high-dimensional spaces: Locality Sensitive Hashing

Dmitriy Selivanov

04 Sep 2015

Data Science

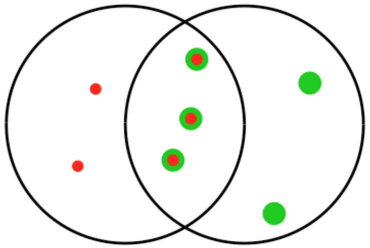


- Statistics
- Domain knowledge
- **Computer science**

Today's problem: near duplicates detection

- Given: High dimensional data points $(x_1, x_2, \ldots)$
  - Image
  - User-Item (rating, whatever) matrix
- And some distance function $d(x_1, x_2)$
  - Euclidean distance
  - Cosine distance
  - **Jaccard distance**

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$



Find all pairs of data points x_i, x_j within distance threshold $d(x_i, x_j) < s$

Near-neighbor search

Applications

- Duplicate detection (plagiarism, entity resolution)
- Recommender systems
- Clustering

High Dimensions

- Bag-of-words model
- Large-Scale Recommender Systems (many users vs many items)

High-dimensional spaces are **lonely** places

- Curse of dimensionality
    - almost all pairs of points are equally far away from one another

Finding similar text documents

Examples

- Mirror websites
- Similar news articles (google news, yandex news?)

Challenges

- Pieces of one document can appear out of order (headers, footers, etc), diffrent lengths.
- Large collection of documents can not fit in RAM
- Too many documents to compare all pairs - $O(n^2)$ complexity

Pipeline

- Pieces of one document can appear out of order (headers, footers, etc) => Shingling
- **Documents as sets**
- Large collection of documents can not fit in RAM => Minhashing
- **Convert large sets to short signatures, while preserving similarity**
- Too many documents to compare all pairs - $O(n^2)$ complexity => Locality Sensitive Hashing
- **Focus on pairs of signatures likely to be from similar documents.**

Document representation

Example phrase:

"To be, or not to be, that is the question"

Documents as sets

- Bag-of-words => **set** of words:
  - `{"to", "be", "or", "not", "that", "is", "the", "question"}`
  - **don't work well - need ordering**
- **k-shingles** or **k-gram** => unordered *set* of k-grams:
  - word level (for k = 2)
    - `{"to be", "be or", "or not", "not to", "be that", "that is", "is the", "the question"}`
  - caharacter level (for k = 3):
    - `{"to ", "o b", " be", "be ", "e o", " or", "or ", "r n", " no", "not", "ot ", "t t", " to", "e t", " th", "tha", "hat", "at ", "t i", " is", "is ", "s t", "the", "he ", "e q", " qu", "que", "ues", "est", "sti", "tio", "ion"}`

Practical notes

Optionally can compress (hash!) long shingles into 4 byte integers!

Picking k

- $k = 5$ is OK for short documents
- $k = 10$ is OK for long documents

k should be picked large enough that the probability of any given shingle appearing in any given document is low

Binary term-document-matrix

- **D1** = "светило летнее солце" => s1 = `{"светило", "летнее", "солце"}`
- **D2** = "яркое летнее солнце" => s2 = `{"яркое", "летнее", "солце"}`

shingle	s1	s2	intersecton	union
....	.	.	.	.
осенняя	0	0	-	-
светило	1	0	-	+
летнее	1	1	+	+
солце	1	1	+	+
яркое	0	1	-	+
погода	0	0	-	-
....	.	.	.	.

Type of rows

type	s1	s2
a	1	1
b	1	0
c	0	1
d	0	0

A, B, C, D - # rows types a, b, c, d

$$J(s_1, s_2) = \frac{A}{(A + B + C)}$$

Minhashing

Convert large sets to short **signatures**

1. Random permutation of rows of the **input-matrix** M.
2. **Minhash function** $h(c)$ = # of first row in which column $c == 1$.
3. Use N **independent** permutations we will end with N minhash functions. => can construct **signature-matrix** from input-matrix using these minhash functions.

Minhashing example

p1	p2	p3	p4	s1	s2	s3
4	1	4	6	1	1	0
3	4	1	1	1	1	0
7	6	6	2	1	0	0
6	2	7	3	1	1	0
5	3	2	5	0	0	1
2	5	3	7	0	0	1
1	7	5	4	0	0	1

s1	s2	s3
3	3	1
1	1	3
1	1	2
1	1	4

Property

$$P_{perm}(h(s_1) = h(s_2)) = ???$$

p1	p2	p3	p4	s1	s2	s3
4	1	4	6	1	1	0
3	4	1	1	1	1	0
7	6	6	2	1	0	0
6	2	7	3	1	1	0
5	3	2	5	0	0	1
2	5	3	7	0	0	1
1	7	5	4	0	0	1

s1	s2	s3
3	3	1
1	1	3
1	1	2
1	1	4

- $= J(s_1, s_2)$
- Why? Both $\frac{A}{(A+B+C)}$
- **remember this result**

Implementation of Minhashing

- random permutation
- random lookup

One-pass implementation

1. Instead of N permutations pick N **independent** hash-functions ($N = O(1/\epsilon^2)$)
2. For column c and hash-function h_i keep slot $Sig(i, c)$. Init with `+Inf`.
3. Scan rows looking for 1
    - Suppose row r has 1 in column c
    - Then for each k_i : If $h_i(r) < Sig(i, c)$ => $Sig(i, c) := h_i(r)$)

Universal hashing

$h_i(x) = ((ax + b) \mod p)$

- a, b - integers
- p - large prime: $p > N$

Algorithm

Row	C1	C2
1	1	0
2	0	1
3	1	1
4	1	0
5	0	1

$h(x) = x \bmod 5$
$g(x) = (2x+1) \bmod 5$

	Sig1	Sig2
$h(1) = 1$	1	∞
$g(1) = 3$	3	∞
$h(2) = 2$	1	2
$g(2) = 0$	3	0
$h(3) = 3$	1	2
$g(3) = 2$	2	0
$h(4) = 4$	1	2
$g(4) = 4$	2	0
$h(5) = 0$	1	0
$g(5) = 1$	2	0

```
for each row r do begin
  for each hash function hi do
    compute hi (r);
  for each column c
    if c has 1 in row r
      for each hash function hi do
        if hi(r) is smaller than M(i, c) then
          M(i, c) := hi(r);
end;
```

Candidates

Still $O(n^2)$ comlexity

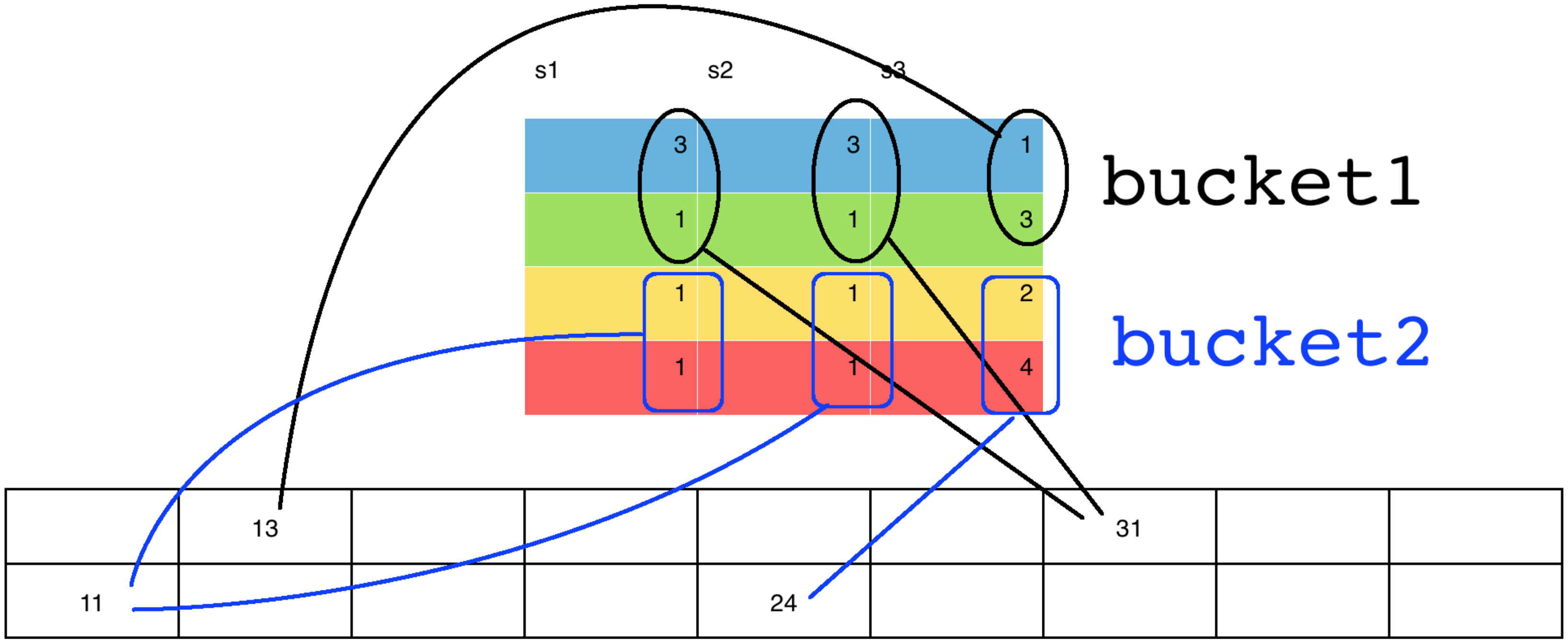
Bruteforce

```
library('microbenchmark')
library('magrittr')
jaccard <- function(x, y) {
  set_intersection <- intersect(x, y) %>% length
  set_union <- length(x) + length(y) - set_intersection
  return(set_intersection / set_union)
}
elements <- sapply(seq_len(1e5), function(x) paste(sample(letters, 4), collapse = '')) %>% unique
set_1 <- sample(elements, 100, replace = F)
set_2 <- sample(elements, 100, replace = F)
microbenchmark(jaccard(set_1, set_2))
```

```
## Unit: microseconds
##                   expr    min     lq     mean  median     uq     max neval
##  jaccard(set_1, set_2) 56.812 58.693 65.64373 61.8075 69.184 161.986   100
```

Job for LSH:

1. Divide M into b bands, r rows each
2. Hash each column in b_i band into table with large number of buckets
3. Column become candidate if fall into same bucket for any band



Bands number tuning

1. The probability that the signatures agree in all rows of one particular band is s^r
2. The probability that the signatures disagree in at least one row of a particular band is $1 - s^r$
3. The probability that the signatures disagree in at least one row of each of the bands is $(1 - s^r)^b$
4. The probability that the signatures agree in all the rows of at least one band, and therefore become a candidate pair, is $1 - (1 - s^r)^b$

Example

- 100k documents =>
- 100 hash-functions => signatures of 100 integers = 40mb
- $b = 20, r = 5$
- find all documents, similar at least $s = 0.8$

1. Probability C1, C2 identical in one particular band: $(0.8)^5 = 0.328$
2. Probability C1, C2 are not similar in all of the 20 bands: $(1 - 0.328)^{20} = 0.00035$

Goal : tune b and r to catch most similar pairs, but few non-similar pairs

LSH function families

- Minhash - jaccard similiarity
- Random projections (random hypeplanes) - cosine similiarity
- p-stable-distributions - Euclidean distance (and L_p norm)
- Edit distance
- Hamming distance

References

- Jure Leskovec, Anand Rajaraman, Jeff Ullman: Mining of Massive Datasets. http://mmds.org/
- P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality.
- Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji: Hashing for Similarity Search: A Survey
- Alexandr Andoni and Piotr Indyk : Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions
- Mayur Datar, Nicole Immorlica, Piotr Indyk, Vahab S. Mirrokni: Locality-Sensitive Hashing Scheme Based on p-Stable Distributions

https://www.coursera.org/course/mmds

R package LSHR - https://github.com/dselivanov/LSHR