Algorithms for Similarity Search

Yury Lifshits
Yahoo! Research

ESSCASS 2010

Outline

Similarity Search

1. High-level overview
2. Locality-sensitive hashing
3. Combinatorial approach

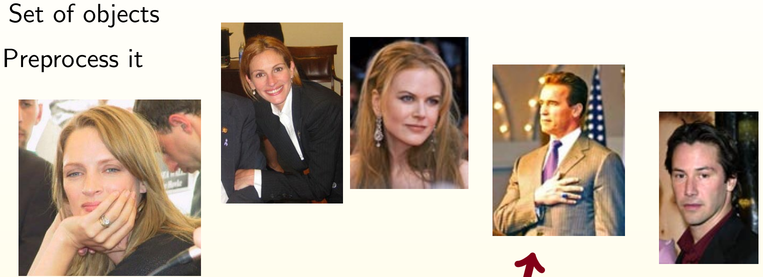
Content Optimization

1. High-level overview
2. Explore/exploit for Yahoo! frontpage
3. Ediscope project for social engagement analysis

Similarity Search: an Example

Input: Set of objects

Task: Preprocess it



Most similar

Query: New object

Task: Find the most similar one in the dataset

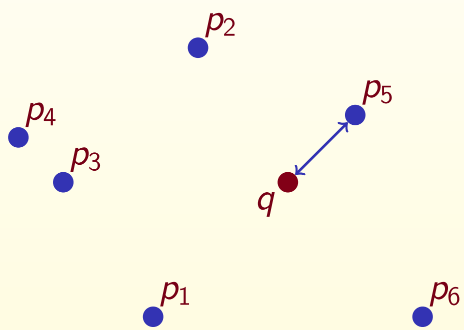
More Formally

Search space: object domain $\mathbb{U}$, similarity function σ

Input: database $S = \{p_1, \ldots, p_n\} \subseteq \mathbb{U}$

Query: $q \in \mathbb{U}$

Task: find $\text{argmax}_{p_i}\ \sigma(p_i, q)$



Applications (1/5) Information Retrieval

- Content-based retrieval (magnetic resonance images, tomography, CAD shapes, time series, texts)
- Spelling correction
- Geographic databases (post-office problem)
- Searching for similar DNA sequences
- Related pages web search
- Semantic search, concept matching

Applications (2/5) Machine Learning

- kNN classification rule: classify by majority of k nearest training examples. E.g. recognition of faces, fingerprints, speaker identity, optical characters
- Nearest-neighbor interpolation

Applications (3/5) Data Mining

- Near-duplicate detection
- Plagiarism detection
- Computing co-occurrence similarity (for detecting synonyms, query extension, machine translation...)

Key difference:
Mostly, off-line problems

Applications (4/5) Bipartite Problems

- Recommendation systems (most relevant movie to a set of already watched ones)
- Personalized news aggregation (most relevant news articles to a given user's profile of interests)
- Behavioral targeting (most relevant ad for displaying to a given user)

Key differences:
Query and database objects have different nature
Objects are described by features and connections

Applications (5/5) As a Subroutine

- Coding theory (maximum likelihood decoding)
- MPEG compression (searching for similar fragments in already compressed part)
- Clustering

Variations of the Computation Task

Additional requirements:

- Dynamic nearest neighbors: moving objects, deletes/inserts, changing similarity function

Related problems:

- Nearest neighbor: nearest museum to my hotel
- Reverse nearest neighbor: all museums for which my hotel is the nearest one
- Range queries: all museums up to 2km from my hotel
- Closest pair: closest pair of museum and hotel
- Spatial join: pairs of hotels and museums which are at most 1km apart
- Multiple nearest neighbors: nearest museums for each of these hotels
- Metric facility location: how to build hotels to minimize the sum of "museum — nearest hotel" distances

Brief History

1908 Voronoi diagram

1967 kNN classification rule by Cover and Hart

1973 Post-office problem posed by Knuth

1997 The paper by Kleinberg, beginning of provable upper/lower bounds

2006 Similarity Search book by Zezula, Amato, Dohnal and Batko

2008 First International Workshop on Similarity Search

2009 Amazon acquires SnapTell

Ideal algorithm

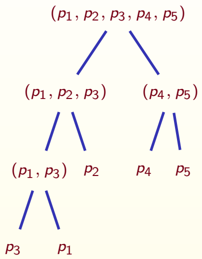
+ Any data model
+ Any dimension
+ Any dataset
+ Near-linear space
+ Logarithmic search time
+ Exact nearest neighbor
+ Zero probability of error
+ Provable efficiency

Nearest Neighbors in Theory

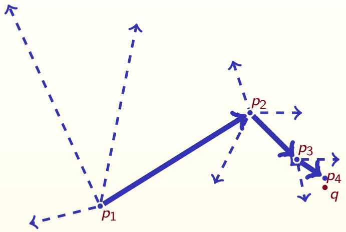
Sphere Rectangle Tree Orchard's Algorithm k-d-B tree Geometric near-neighbor access tree Excluded middle vantage point forest mvp-tree Fixed-height fixed-queries tree AESA Vantage-point tree LAESA R*-tree Burkhard-Keller tree BBD tree Navigating Nets Voronoi tree Balanced aspect ratio tree Metric tree vp^s-tree M-tree Locality-Sensitive Hashing SS-tree R-tree Spatial approximation tree Multi-vantage point tree Bisector tree mb-tree Cover tree Hybrid tree Generalized hyperplane tree Slim tree Spill Tree Fixed queries tree X-tree k-d tree Balltree Quadtree Octree Post-office tree

Theory: Four Techniques

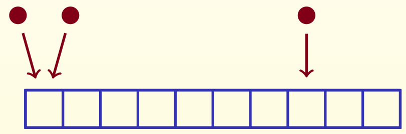
Branch and bound



Greedy walks

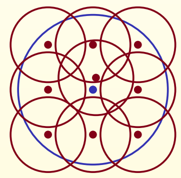


Mappings: LSH,
random projections, minhashing



Epsilon nets

Works for small intrinsic dimension



Future of Similarity Search

- Cloud implementation
  - Hadoop stack
  - API (as in OpenCalais, Google Language API, WolframAlpha)
- Focus on memory

Locality Sensitive Hashing

LSH vs. Ideal algorithm

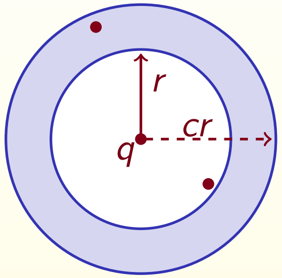
- – Any data model
- – Exact nearest neighbor
- – Zero probability of error

- + Provable efficiency
- + Any dimension
- + Any dataset

- $+^*$ Near-linear space
- $+^*$ Logarithmic search time

Approximate Algorithms

***c*-Approximate *r*-range query:** if there at least one $p \in S: \; d(q,p) \le r$ return some $p': \; d(q,p') \le cr$



***c*-Approximate nearest neighbor query:** return some $p' \in S: \; d(p',q) \le cr_{NN}$, where $r_{NN} = \min_{p \in S} d(p,q)$

Today we consider only range queries

Today's Focus

Data model:

- d-dimensional Euclidean space: $\mathbb{R}^d$

Our goal: provable performance bounds

- Sublinear search time, near-linear preprocessing space

Still an open problem: approximate nearest neighbor search with logarithmic search and linear preprocessing

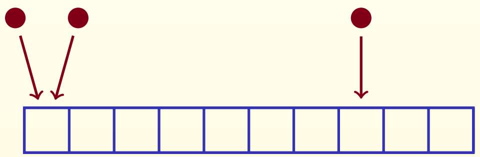
Locality-Sensitive Hashing: General Scheme

Definition of LSH

Indyk&Motwani'98

Locality-sensitive hash family $\mathcal{H}$ with parameters (c, r, P_1, P_2):

- If $\|p - q\| \leq r$ then $\mathcal{Pr}_{\mathcal{H}}[h(p) = h(q)] \geq P_1$
- If $\|p - q\| \geq cr$ then $\mathcal{Pr}_{\mathcal{H}}[h(p) = h(q)] \leq P_2$



The Power of LSH

Notation: $\rho = \frac{\log(1/P_1)}{\log(1/P_2)} < 1$

Theorem

Any (c, r, P_1, P_2)-locality-sensitive hashing leads to an algorithm for c-approximate r-range search with (roughly) n^ρ query time and $n^{1+\rho}$ preprocessing space

Proof in the next four slides

LSH: Preprocessing

Composite hash function: $g(p) =< h_1(p), \ldots, h_k(p) >$

Preprocessing with parameters L, k:

1. Choose at random L composite hash functions of k components each
2. Hash every $p \in S$ into buckets $g_1(p), \ldots, g_L(p)$

Preprocessing space: $\mathcal{O}(Ln)$

LSH: Search

1. Compute $g_1(q), \ldots, g_L(q)$
2. Go to corresponding buckets and explicitly check $d(p, q) \leq ?cr$ for every point there
3. Stopping conditions: (1) we found a satisfying object or (2) we tried at least $3L$ objects

Search time is $\mathcal{O}(L)$

LSH: Analysis (1/2)

$1 - x \leq e^{-x}$ for $x \in [0, 1]$

In order to have probability of error at most δ we set k, L such that:

- The expected number of cr-far objects to be tried is $P_2^k L n = L$
- The chance to never be hashed to the same bucket as q for a true r-neighbor $(1 - P_1^k)^L \leq e^{-P_1^k L}$ is at most δ

Rewriting these constraints:

$$P_2^k n = 1 \qquad L = P_1^{-k}(-log\delta)$$

LSH: Analysis (2/2)

$$P_2^k n = 1 \qquad L = P_1^{-k}(-log\delta)$$

Solution:

$$k = \frac{\log n}{\log(1/P_2)}$$

$$L = P_1^{-\frac{\log n}{\log(1/P_2)}} \log(1/\delta) = n^{\frac{\log(1/P_1)}{\log(1/P_2)}} \log(1/\delta) = n^{\rho} \log(1/\delta)$$

Preprocessing space $\mathcal{O}(Ln) \approx n^{1+\rho+o(1)}$
Search $\mathcal{O}(L) \approx n^{\rho+o(1)}$

LSH Based on p-Stable Distributions

Datar, Immorlica, Indyk and Mirrokni'04

$$h(p) = \lfloor \frac{p \cdot r}{w} + b \rfloor$$

r: random vector, every coordinate comes from $N(0, 1)$
b: random shift from $[0, 1]$
w: quantization width

$$\rho(c) < \frac{1}{c}$$