Algorithms for Nearest Neighbors

Classic Ideas, New Ideas

Yury Lifshits
Steklov Institute of Mathematics at St.Petersburg
http://logic.pdmi.ras.ru/~yura

University of Toronto, July 2007

Outline

1. Problem Statement
   - Applications
   - Data Models
2. Classic Ideas
   - Search Trees
   - Random Projections
   - Look-Up Methods
3. New Ideas
   - Proving Hardness of Nearest Neighbors
   - Probabilistic Analysis for NN
   - New Data Models

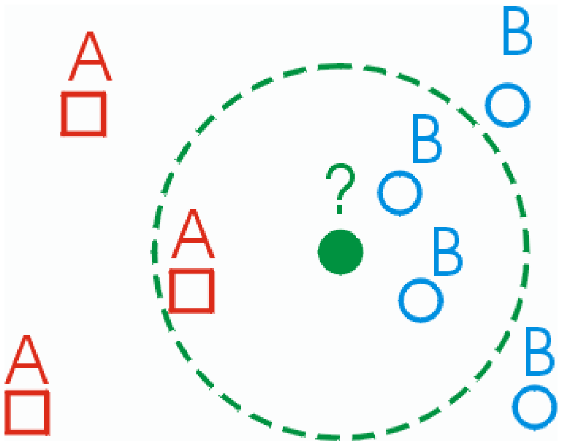
Part I
Formulating the Problem

Informal Problem Statement

To preprocess a database of n objects
so that given a query object,
one can effectively determine
its nearest neighbors in database

First Application (1960s)

Nearest neighbors for classification:



Picture from `http://cgm.cs.mcgill.ca/ soss/cs644/projects/perrier/Image25.gif`

Applications

What applications of nearest neighbors do you know?

- Text classification
- Statistical data analysis, e.g. medicine diagnosis
- Pattern recognition: characters, faces
- Code plagiarism detection
- Coding theory
- Data compression
- **Web:** recommendation systems, on-line ads, personalized news aggregation, long queries in web search, near-duplicates detection

Data Model in General

Formalization for nearest neighbors consists of:

- Representation format for objects
- Similarity function

Remark 1: Usually there is original and "reduced" representation for every object

Remark 2: Accuracy of NN-based algorithms depends solely on a data model, no matter what specific exact NN algorithm we use

Data Models (1/2)

- Vector Model
  - Similarity: l^2, scalar product, cosine
- String Model
  - Similarity: Hamming distance, edit distance
- Black-box model
  - Similarity: given by oracle
    The only knowledge is triangle inequality

Data Models (2/2)

- Set Model
  - Similarity: size of intersection
- Small graphs
  - Similarity: structure/labels matching

More data models?

Variations of the Computation Task

- Range queries: retrieve all objects within given range from query object
- Approximate nearest neighbors
- Multiple nearest neighbors (many queries)
- Nearest assignment
- All over-threshold neighbor pairs
- Nearest neighbors in dynamically changing database: moving objects, deletes/inserts, changing similarity function

Part II
Classic Ideas

Linear Scan

What is the most obvious solution for nearest neighbors?

Answer:
compare query object with every object in database

Advantages:
No preprocessing
Exact solution
Works in any data model

Directions for improvement:
order of scanning, pruning

KD-Trees

Preprocessing:

Build a kd-tree: for every internal node on level l we make partitioning based on the value of $l \mod d$-th coordinate

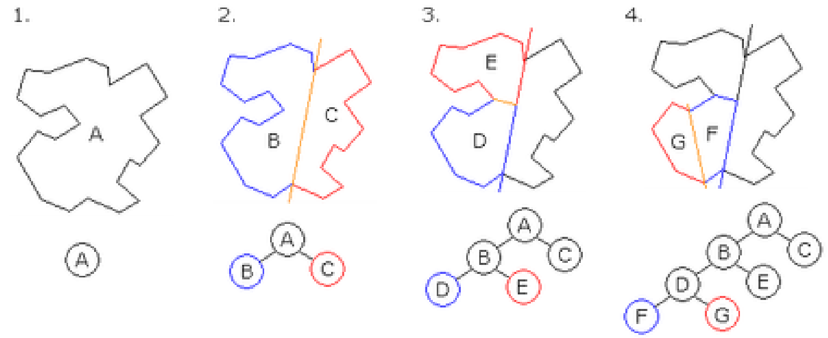
Query processing:

Go down to the leaf corresponding to the the query point and compute the distance;

(Recursively) Go one step up, check whether the distance to the second branch is larger than that to current candidate neighbor
if "yes" go up, else check this second branch

BSP-Trees

Generalization: BSP-tree allows to use any hyperplanes in tree construction



VP-Trees

Partitioning condition: $d(p,x) \ <?\ r$
Inner branch: $B(p, r(1+\varepsilon))$
Outer branch: $R^d / B(p, r(1-\delta))$

Search:

If $d(p,q) < r$ go to inner branch
If $d(p,q) > r$ go to outer branch and return minimum between obtained result and $d(p,q)$

Kleinberg Algorithm (1/3)

Preprocessing

1. Choose l random vectors $V = \{v_1, \ldots, v_l\}$ with unit norm
2. Precompute all scalar products between database points and vectors from V

Kleinberg Algorithm (2/3)

Random Projection Test

Input: points x,y and q, vectors $u_1, \ldots, u_k$
Question: what is smaller $|x - q|$ or $|y - q|$?

Test:
For all i compare $(x \cdot v_i - q \cdot v_i)$ with $(y \cdot v_i - q \cdot v_i)$
Return the point which has "smaller"
on majority of vectors

Kleinberg Algorithm (3/3)

Query Processing

1. Choose a random subset Γ of V, $|\Gamma| = \log^3 n$
2. Compute scalar products between query point q and vectors from Γ
3. Make a tournament for choosing a nearest neighbor:
   1. Draw a binary tree of height $\log n$
   2. Assign all database points to leafs
   3. For every internal point (say, x vs. y) make a random projection test using some vectors from Γ

Inverted Index

Data model: every object is a (weighted) set of terms from some dictionary

Preprocessing:
For very term store a list of all documents
in database with nonzero weight on it

Query processing:
Retrieve all point that have at least one
common term with the query documet;
Perform linear scan on them

Locality-Sensitive Hashing

Desired hash family $\mathcal{H}$:

- If $\|p - q\| \leq R$ then $\mathcal{P}r_{\mathcal{H}}[h(p) = h(q)] \geq p_1$
- If $\|p - q\| \geq cR$ then $\mathcal{P}r_{\mathcal{H}}[h(p) = h(q)] \leq p_2$

Preprocessing:
Choose at random several hash functions from $\mathcal{H}$
Build inverted index for hash values
of object in database

Query processing:
Retrieve all object that have at least one
common hash value with query object;
Perform linear scan on them

Part III
New Ideas

This section represents:

- Some of my own ideas
- Joint work with Benjamin Hoffmann and Dirk Nowotka (CSR'07)

Inclusions with Preprocessing (1/2)

Input
Family $\mathcal{F}$ of subsets of U

Query task
Given a set $f_{new} \subseteq U$ to decide
whether $\exists f \in \mathcal{F} : \quad f_{new} \subseteq f$

Constraints
Data storage after preprocessing $poly(|\mathcal{F}| + |U|)$
Time for query processing $poly(|U|)$

Open problem: is there an algorithm satisfying given constraints?

Inclusions with Preprocessing (2/2)

Reformulation in SAT style:

Input
Formula $\mathcal{F}$ in DNF with n variables

Query task
Given an assignment x to evaluate $\mathcal{F}(x)$

Constraints
Data storage after preprocessing $poly(|\mathcal{F}|)$
Time for query processing $poly(n)$

Open problem: is there an algorithm satisfying given constraints?

"NP Analogue" for Search Problems

Every problem in **SEARCH class** is characterized by poly-time computable Turing Machine M:

Input
Strings $x_1, \ldots, x_n, \qquad |x_i| = m$

Query task
Given string y of length m to answer
whether $\exists i : M(x_i, y) = yes$

Tractable problems in SEARCH

Input
Strings $x_1, \dots, x_n$, $|x_i| = m$

Query task
Given string y of length m to answer whether $\exists i : M(x_i, y) = yes$

Tractable solution
Preprocessing in $poly(m, n)$ space

Query processing in $poly(m, \log n)$ time with RAM access to preprocessed database

Inclusions is in SEARCH. Is it tractable?

Complete problems in SEARCH (1/2)

Program Search problem:

Input
Turing machines $P_1 \dots, P_n$

Query task
Given string y of length m to answer whether $\exists i : P_i(y) = yes$ after at most m steps

Open problem: is Program Search tractable?

Complete problems in SEARCH (2/2)

Parallel Run problem:

Input
$x_1 \dots, x_n$

Query task
Given poly-time computable P to answer whether $\exists i : P(x_i) = yes$

Open problem: is Parallel Run tractable?

Probabilistic Analysis in a Nutshell

- We define a probability distribution over databases
- We define probability distribution over query objects
- We construct a solution that is efficient/accurate with high probability over "random" input/query

Zipf Model

- Terms $t_1, \ldots, t_m$
- To generate a document we take every t_i with probability $\frac{1}{i}$
- Database is n independently chosen documents
- Query document has exactly one term in every interval $[e^i, e^{i+1}]$
- Similarity between documents is defined as the number of common terms

Magic Level Theorem

Magic Level $q = \sqrt{2 \log_e n}$

Theorem

1. *With very high probability there exists a document in database having* $q - \varepsilon$ **top** *terms of query document*
2. *With very small probability there exists a document in database having* **any** $q + \varepsilon$ *overlap with query document*

Sparse Vector Model

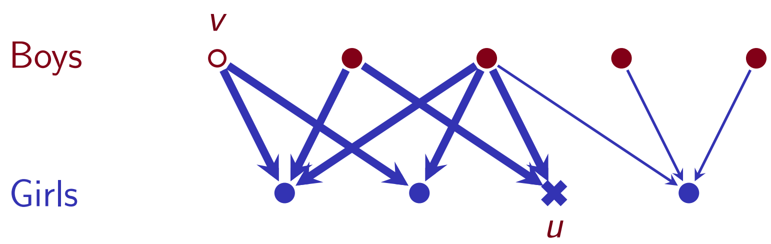
Database: points in R^d,
every point has at most $k \ll d$ nonzero coordinates

Similarity: scalar product

Constraints:
$poly(n + d)$ for preprocessing time,
$poly(k) \cdot polylog(n + d)$ for query

Open Problem: solve NN for sparse vector model within given constraints

Amazon Recommendations



Amazon Nearest Neighbors

Database: Bipartite graph with n vertices,
every vertex of the first part
has out degree at most $k \ll n$

Query: Given a new vertex u in the first part
to find a vertex u in the second part
having maximal number of 3-step paths to v

Constraints:
$poly(n)$ time for preprocessing
$poly(k) \cdot polylog(n)$ for query

Open Problem: solve NN for Amazon model within given constraints

Conclusions

Directions for Further Research

- Extend classical NN algorithms to new data models and new search task variations
- Develop theoretical analysis of existing heuristics. Find subcases with provably efficient solutions
- Build complexity theory for problems with preprocessing

Call for Feedback

- Any relevant work?
- How to improve this talk for the next time?
- **Give my open problems to your students!**

Summary

- Classic ideas: search trees, random projections, locality-sensitive hashing, inverted index
- New ideas: SEARCH class, NN for random texts, Amazon and sparse vector models
- Open problems: lower bound for inclusions with preprocessing, algorithm for 3-step similarity

Thanks for your attention! Questions?

References (1/2)

Search "**Lifshits**" or visit http://logic.pdmi.ras.ru/~yura/

B. Hoffmann, Y. Lifshits and D. Nowotka
Maximal Intersection Queries in Randomized Graph Models
http://logic.pdmi.ras.ru/~yura/en/maxint-draft.pdf

P.N. Yianilos
Data structures and algorithms for nearest neighbor search in general metric spaces
http://www.pnylab.com/pny/papers/vptree/vptree.ps

J. Zobel and A. Moffat
Inverted files for text search engines
http://www.cs.mu.oz.au/~alistair/abstracts/zm06compsurv.html

K. Teknomo
Links to nearest neighbors implementations
http://people.revoledu.com/kardi/tutorial/KNN/resources.html

References (2/2)

J. Kleinberg
Two Algorithms for Nearest-Neighbor Search in High Dimensions
http://www.ece.tuc.gr/~vsam/csalgo/kleinberg-stoc97-nn.ps

P. Indyk and R. Motwani
Approximate nearest neighbors: towards removing the curse of dimensionality
http://theory.csail.mit.edu/~indyk/nndraft.ps

A. Andoni and P. Indyk
Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions
http://theory.lcs.mit.edu/~indyk/FOCS06final.ps

P. Indyk
Nearest Neighbors Bibliography
http://theory.lcs.mit.edu/~indyk/bib.html