CS 224: Advanced Algorithms Spring 2017

Lecture 17 — 03/21, 2017

Prof. Piotr Indyk *Scribe: Artidoro Pagnoni, Jao-ke Chin-Lee*

1 Overview

In the last lecture we saw semidefinite programming, Goemans-Williamson MaxCut and approximation algorithms in other settings (streaming algorithms, and in particular distinct elements).

In this lecture we cover nearest neighbor search in high dimensions with guest lecturer Prof. Piotr Indyk from MIT.

2 Definition of the Problem and Applications

We first define the nearest neighbor search problem.

Definition 1 (Nearest Neighbor Search)**.** *Given a set P of n points in $\mathbb{R}^n$, for any query q, return the $p \in P$ minimizing $\|p - q\|$, i.e. we wish to find the point closest to q in a metric space.*

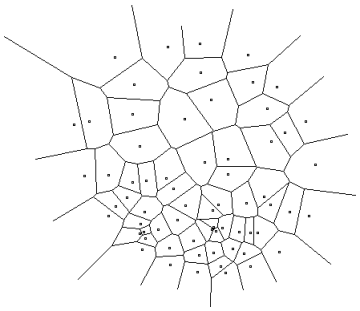
Definition 2 (r-Near Neighbor Search)**.** *Given parameter r and a set P of n points in $\mathbb{R}^n$, for any query q, if any such exist, return a $p \in P$ s.t. $\|p - q\| \leq r$, i.e. we wish to find points within distance of r from q in a metric space.*

Note we can consider r-near neighbor search to be a decision problem formulation of nearest neighbor search. Instead of optimizing, we want to know if there is any point within the radius.

Nearest neighbor search has applications in both its high-dimensional and low-dimensional version. Classification problems in machine learning are an application of the high dimensional instances. Consider, for example, the problem of classifying a new image of a hand written digit given a set of classified digits. This can be done by finding the closest image to the queried image. In this problem the pixels are the dimensions. The performance of nearest neighbor is comparable to most approaches, besides convolutional neural networks. Another example with high dimensionality is near duplicate retrieval. Each document can be represented as a bag of words, a sparse but very high dimensional vector, where the dimension is the number of words. Once the documents are represented in this feature space, the solution can be found with nearest neighbor.

Given the high dimensionality we have to avoid exponential dependence in the dimensionlity of the data.

Example: 2-d Nearest Neighbor Search
We briefly consider the simple case of $d = 2$, i.e. the problem of determining the nearest point in a plane. The standard solution is to use *Voronoi diagrams*, which split the space into regions according to proximity, in which case the problem reduces to determining point location within the Voronoi diagram.



We can do this by building a BST quickly to determining the cell containing any given point: this takes $O(n)$ space and $O(\log n)$ query. y this is a binary search tree. This approach works well in this example because of the low dimensionality.

When we consider higher dimensions, i.e. $d > 2$, however, the Voronoi diagram has size $n^{\lceil d/2 \rceil}$ (although the good querry time), which is prohibitively large. An alternative method is to simply perform a linear scan in $O(dn)$ spave and time. These are the only known general solutions (other specific approaches exist but degenerate to this worst case).

Moreover, if there was an algorithm that performed n queries in $O(dn1+\alpha)$ time for $\alpha < 1$ (including preprocessing), then there would be an algorithm solving satisfiability in (slightly) sub-exponential time. Many believe that such an algorithm does not exist, which would set a lower bound on the performance of nearest neighbor search.

3 Approximate Nearest Neighbor

We try instead to relax the problem. First, define the approximation version of nearest neighbor search.

Definition 3 (c-Approximate Nearest Neighbor Search)**.** *Given approximation factor c and a set P of n points in $\mathbb{R}^n$, for any query q, return a $p' \in P$ s.t. $\|p' - q\| \leq cr$ where $r = \|p - q\|$ and p is the true nearest neighbor, i.e. we wish to find a point at most some factor c away from the point closest to q.*

Definition 4 (c-Approximate r-Near Neighbor Search)**.** *Given approximation factor c and parameter r and a set P of n points in $\mathbb{R}^n$, for any query q, if any $p \in P$ exists s.t. $\|p - q\| \leq r$, return a $p' \in P$ s.t. $\|p' - q\| \leq rc$, i.e. we wish to a point at most some factor c away from some point within a range of q.*

Also note that if we enumerate all approximate near neighbors, we can find the exact near neighbor.

We build up a data structure to solve these problems; if, however, in the c-approximate r-near neighbor search, there is no such point, our data structure can return anything, so instead we can compute the distance to double check, and if the distance places us outside our ball, we know there was no such point.

The algorithms that will be described are randomized. The data structure is constructed correctly with non zero probability of success. To improve the probability, we therefore have to create many copies of the data structures.

Now we consider algorithms to solve these search problems.

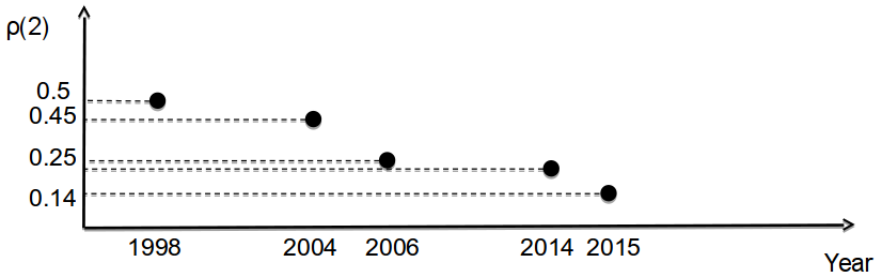
4 Algorithms

Most algorithms are randomized, i.e. for any query we succeed with high probability over an the initial randomness used to build our data structure (which is the only randomness introduced into the system). In fact, because our data structure satisfied the desired behavior with probability bounded by a constant. By working iteratively and generating new copies of the data structure, we can therefore bring our probability of failure arbitrarily close to 0.

Some algorithms and their bounds appear below.

We will focus on nearest neighbors with respect to Eucleidean norm, and present algorithms with polynomial dependence on dimension for both space and time. In particular, we will have near-linear space: $O(dn + n^{1+\rho(c)})$, $\rho(c) < 1$; and sub-linear query time: $O(dn^{\rho(c)})$.

Over time there have been constant developments of the exponent, as shown in the picture below.



5 Locality Sensitive Hashing

Recall when we covered hashing that collisions happened more or less on an independent basis. Here, we will use hashing whose probability of collision depends on the similarity between the queries.

Definition 5 (Sensitivity)**.** *We call a family $\mathcal{H}$ of functions $h : \mathbb{R}^d \to U$ (where U is our hashing universe)* (P_1, P_2, r, cr)-sensitive *for a distance function D if for any p, q:*

- $D(p, q) < r \Rightarrow \mathbb{P}[h(p) = h(q)] > P_1$*, i.e. if p and q are close, the probability of collision is high; and*
- $D(p, q) > cr \Rightarrow \mathbb{P}[h(p) = h(q)] < P_2$*, i.e. if p and q are far, the probability of collision is low.*

Special Case: Hamming distance
Suppose we have $h(p) = p_i$, i.e. we hash to the ith bit of length d bitstring p, and let $D(p, q)$ be the Hamming distance between p and q, i.e. the number of different bits (place-wise) between p and q. Then our probability of collision is always $1 - D(p, q)/d$ (since $D(p, q)/d$ is the probability of choosing a different bit).

5.1 Algorithm

In preprocessing, we build our data structure: we choose L hash functions, and hash all the data points. For the hamming distance, hashing consists in the concatenation of k hamming hashfunctions in packets. We select with replacement k dimensions (bits) of the point, and create a hash of the form $g(p) = \langle h_1(p), h_2(p), \ldots, h_k(p) \rangle$.

After selecting $g_1, \ldots, g_L$ independently and at random (where k and L are functions of c and r that we will compute later), hash every $p \in P$ to buckets $g_1(p), \ldots, g_L(p)$.

When querying, we proceed as follows:

- retrieve points from buckets $g_1(q), \ldots, g_L(q)$ until:
  - Either we have retrieved the points from all L buckets, or
  - we have retrieved more than $3L$ points in total (to avoid run time of $O(Ln)$, this is the case when many points are within the radius of search)
- answer query based on retrieved points (whether procedure terminated from first or second case above)

Note that hashing takes time d, and with L buckets, the total time here is $O(dL)$.

Next we will prove space and query performance bounds as well as the correctness of the parameters.

5.2 Analysis

We will prove two lemmata for query bounds, the second specifically for Hamming LSH.

Lemma 6. *The above algorithm solves c-approximate nearest neighbor with*

- $L = Cn^\rho$ *hash functions, where* $\rho = \log P_1 / \log P_2$, *where* C *is a function of* P_1 *and* P_2, *for* P_1 *bounded away from 0, and hence constant; and*
- *a constant success probability for fixed query* q

Proof. We assume that a solution exists, so define

- p point s.t. $\|p - q\| \leq r$;
- $\text{FAR}(q) = \{p' \in P : \|p' - q\| > cr\}$ the set of points "far" from q;
- $B_i(q) = \{p' \in P : g_i(p') = g_i(q)\}$ the set of points in the same bucket as q

We will show that the following two events E_1 and E_2 occur with nonzero probability.

- $E_1 : g_i(p) = g_i(q)$ for some $i = 1, \ldots, L$: the event of colliding in a bucket with the desired query;

- $E_2 : \sum_i |B_i(q) \cap \text{FAR}(q)| < 3L$: the event of total number of far points in buckets not exceeding $3L$.

The key step is to set $k = \lceil \log_{1/P_2} n \rceil$.
Observe that for $p' \in \text{FAR}(q)$, $\mathbb{P}[g_i(p') = g_i(q)] \leq P_2^k \leq 1/n$. Therefore

$$\begin{aligned}
&\mathbb{E}[|B_i(1) \cap \text{FAR}(q)|] \leq 1 \\
\Rightarrow &\mathbb{E}[\sum_i |B_i(1) \cap \text{FAR}(q)|] \leq L \\
\Rightarrow &\mathbb{P}[\sum_i |B_i(1) \cap \text{FAR}(q)| \geq 3L] \leq \frac{1}{3} \text{ by Markov}
\end{aligned}$$

and, picking $L = Cn^\rho$:

$$\begin{aligned}
&\mathbb{P}[g_i(p) = g_i(q)] \geq \frac{1}{P_1^k} \geq P_1^{1+\log_{1/P_2} n} \\
\Rightarrow &\mathbb{P}[g_i(p) = g_i(q)] \geq \frac{1}{P_1 n^\rho} =: \frac{1}{L} \text{ (we choose } L \text{ accordingly)} \\
\Rightarrow &\mathbb{P}[g_i(p) \neq g_i(q), i = 1, \ldots, L] \leq \left(1 - \frac{1}{L}\right)^L \leq \frac{1}{e} \\
\Rightarrow &\mathbb{P}[E_1 \text{ not true }] + \mathbb{P}[E_2 \text{ not true }] \leq \frac{1}{3} + \frac{1}{e} \approx .7 \\
\Rightarrow &\mathbb{P}[E_1 \cap E_2] \geq 1 - \left(\frac{1}{3} + \frac{1}{e}\right) \approx .3
\end{aligned}$$

Thus also note we can make this probability arbitrarily small. □

Lemma 7. *For Hamming LSH functions, we have $\rho = 1/c$.*

Proof. Observe that with a Hamming distance, $P_1 = 1 - r/d$ and $P_2 = 1 - cr/d$, so it suffices to show $\rho = \log P_1 / \log P_2 \leq 1/c$, or equivalently $P_1^c \geq P_2$. But $(1-x)^c \geq 1 - cx$ for any $1 > x > 0$, $c > 1$, so we are done. □

Also note that space is nL, so we have desired space bounds as well.

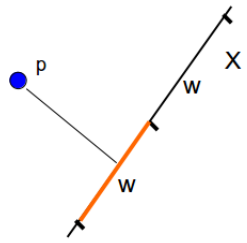
6 Beyond

Now we briefly consider other metrics beyond the Hamming distance, and ways to reduce the exponent ρ.

6.1 Random Projection LSH for L_2

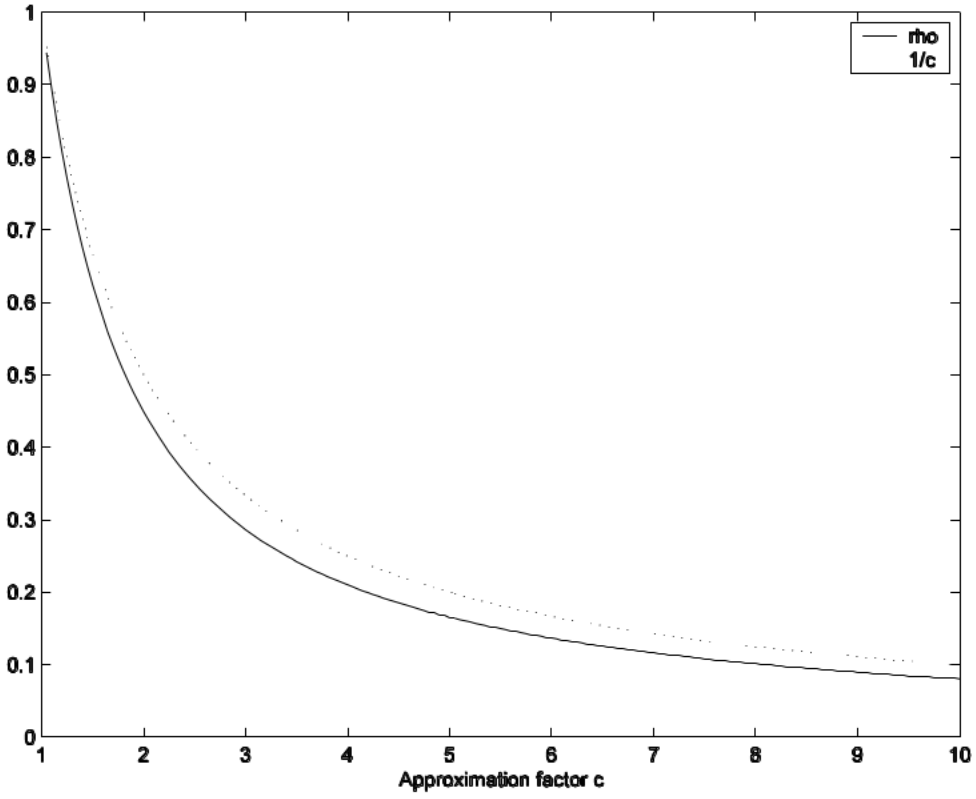
This was introduce by Mayur Datar, Nicole Immorlica, Piotr Indyk and Vahab S. Mirrokni in '04 [DIIM04].

We define $h_{X,b}(p) = \lfloor (p \cdot X + b)/w \rfloor$, where $w \approx r$, $X = (X_1, \ldots, X_d)$, where X_i are i.i.d. Gaussian random variables, and b is a scalar chosen uniformly at random from $[0, w]$. Conceptually, then, our hash function takes p, projects it on to X, shifts it by some amount b, then determines the interval of length w containing it. If points are very far they are unlikely to fall in the same interval.



For the analysis, we need to compute $\mathbb{P}[h(p) = h(q)]$ as a function of $\|p - q\|$ and w. This defines P_1 and P_2. Then, for each c we need to choose w such that $\rho = \log_{1/P_2}(1/P_1)$ is minimized.

This only has a small improvement over the $1/c$ bound, but the hash function is very simple and works directly in L_2. In the following image is a plot of the value of ρ with respect to the $1/c$ bound.

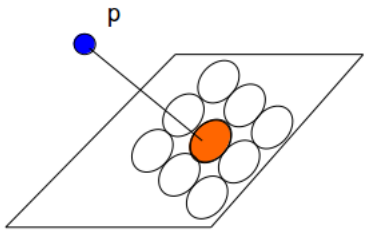


The exponent was reduce from 0.5 to 0.45 for $c = 2$, which is not a dramatic improvement. Therefore we consider alternative ways of projection.

6.2 Ball Lattice Hashing

This covers work by Alexandr Andoni and Piotr Indyk from '06 [AI06].

Instead of projecting onto $\mathbb{R}^1$, we project onto $\mathbb{R}^t$ for constant t. We quantize with a lattice of balls instead of a segment, when we hit empty space, we rehash until we do hit a ball (observe that using a grid would degenerate back to the 1-d case).



With this, we have $\rho = 1/c^2 + O(\log t/\sqrt{t})$, but hashing time increases to $t^{O(t)}$ because our changes of hitting a ball gets smaller and smaller with higher dimensions.

Furthermore, ODonnell, Wu and Zhou showed in '09 that using this approach $\rho \geq 1/c^2 - O(1)$, which set a lower bound on ρ [OWZ09]. This was thought for a long time to be the optimal value.

6.3 Data-dependent hashing

This covers work by Andoni and Razenshteyn from '15 [AR15].

They were able to break the previous lower bound. In fact, the aforementioned LSH schemes of Ball Lattice Hashing are optimal, but only for data oblivious hashing. Which consists in selecting $g_1, \ldots, g_L$ independently at random, hashing all points $p \in P$ into buckets $g_1(p), \ldots, g_L(p)$, and then retrieving points from the buckets $g_1(q), g_2(q), \ldots$.

The new schemes are data dependent, and can therefore go beyond the lower bound of $1/c^2$. Data dependent hashing means that the hash functions are selected based on the clusters of the data.

With this method, there was an improvement of the exponent from $1/c^2$ to $1/(2c^2 - 1$. Which is the best know solution to the problem.

7 Other LSH Methods

We have seen Hamming metric, which is a projection on coordinates, and L_2 norm with random projection and quantization. There are however, other LSH methods.
Some provable:

- L_1 norm, using random shifted grid.
- Vector angle, by Charikar [Cha02] based on Goemans and Williamson [GW94].
- Jaccard coefficient by Broder, Charikar, Frieze and Mitzenmacher in '98 [BCFM98]

Some empirical:

- Inscribed polytopes, by Terasawa and Tanaka from '07 [TT07]
- Orthogonal partition, by Neylon in '10 [Ney10]

And others applied: semantic hashing, spectral hashing, kernelized LSH, Laplacian co-hashing, , BoostSSC, WTA hashing...

7.1 Min-Wise Hasing

In many applications, the vectors tend to be quite sparse (high dimension, very few 1s). It is therefore easier to think about them as sets.

For two sets A and B define the Jaccard coefficient: $J(A, B) = |A \cap B|/|A \cup B|$. The Jaccard coefficient will be equal to 1 if the sets are equal and 0if the are disjoint. We can design LSH families for $J(A, B)$.

Let h be a random permutation of the elements in the universe. Hashing a set consists in taking the minimum of the permuted elements: $g(A) = \min_{a \in A} h(a)$. It then turns out that $\mathbb{P}[g(A) = g(B)] = J(A, B)$.

Proof. The hashes of two sets will be equal if the two permuted sets share same minimum. This means that the minimum has to be the permutation of an element in the intersection of the two sets. Furthermore, since h is a random permutation, all elements are equally likely to be permuted to the minimum. This means that the probability that sets A and B hash to the same number is $J(A, B)$. □

References

[AR15] Alexandr Andoni, Ilya P. Razenshteyn: Optimal Data-Dependent Hashing for Approximate Near Neighbors. *STOC '15*, 793–801, 2015.

[Ney10] Tyler Neylon. A Locality-Sensitive Hash for Real Vectors. *SODA '10*, 1179–1189, 2010.

[OWZ09] Ryan O'Donnell, Yi Wu, Yuan Zhou: Optimal lower bounds for locality sensitive hashing (except when q is tiny). *Electronic Colloquium on Computational Complexity (ECCC) 16*, 130, 2009.

[AI06] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *FOCS '06*, 459-468, 2006.

[Cha02] Moses S. Charikar. Similarity Estimation Techniques from Rounding Algorithms. *STOC '02*, 380–388, 2002.

[DIIM04] Mayur Datar, Nicole Immorlica, Piotr Indyk and Vahab S. Mirrokni. Locality-sensitive Hashing Scheme Based on P-stable Distributions. *SCG '04*, 253–262, 2004.

[TT07] Kengo Terasawa, Yuzuru Tanaka. Spherical LSH for Approximate Nearest Neighbor Search on Unit Hypersphere *WADS '07*, 27–38, 2007.

[GIM99] Aristides Gionis, Piotr Indyk and Rajeev Motwani. Similarity search in high dimensions via hashing. *VLDB*, 518–529, 1999.

[IM98] Piotr Indyk and Rajeeve Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. *STOC '98*, 604–613, 1998.

[BCFM98] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, Michael Mitzenmacher. Min-Wise Independent Permutations. *STOC 1998*, 327–336, 1998.

[GW94] Michel X. Goemans, David P. Williamson. .879-approximation algorithms for MAX CUT and MAX 2SAT. *STOC '94*, 422–431, 1994.