CS 224: Advanced Algorithms Fall 2014

Lecture 14 — October 16, 2014

Prof. Jelani Nelson *Scribe: Jao-ke Chin-Lee*

1 Overview

In the last lecture we covered learning topic models with guest lecturer Rong Ge.

In this lecture we cover *nearest neighbor search* with guest lecturer Piotr Indyk.

2 Introduction

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 by some metric.*

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 q by some metric.*

Note we can consider r-near neighbor search to be a decision problem formulation of nearest neighbor search.

Nearest neighbor search, especially high-dimensional instances, often appear in machine learning (consider the problems of determining image similarity or analyzing handwriting, in which given classified image data, we wish to determine the "class" of some query image as determined by closeness; the dimension here is the number of pixels); another example is near duplicate retrieval, as discussed in the last lecture, in which the dimension is the number of words.

Example ($d = 2$)**.** 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.

When we consider higher dimensions, i.e. $d > 2$, however, the Voronoi diagram has size $n^{\lceil d/2 \rceil}$, which is prohibitively large. Also note we can simply perform a linear scan in $O(dn)$ time.

Therefore, we instead consider approximations.

3 Approximate Nearest Neighbor

We 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.

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 on 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, we can bring our probability of failure arbitrarily close to 0.

Some algorithms and their bounds appear below.

We will focus on *locality-sensitive hashing* as presented by Indyk and Motwani [IM98], Gionis, Indyk and Motwani [GIM99], and Charikar [Cha02], and touch on a few others that have appeared in recent work.

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.*

Example (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

We use functions of the form $g(p) = \langle h_1(p), h_2(p), \ldots, h_k(p) \rangle$.

In preprocessing, we build up our data structure: 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
- 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. Define

- p point s.t. $\|p - q\| \leq r$;
- $\mathrm{FAR}(q) = \{p' \in P : \|p' - q\| > cr\}$ 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

and

- $E_1 : g_i(p) = g_i(q)$ 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 $3L$.

We will show that both E_1 and E_2 occur with nonzero probability.

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

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

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. (Final project ideas?)

6.1 Random Projection LSH for L^2

This covers work by [DIIM04].

We define $h_{X,b}(p) = \lceil (p \cdot X + b)/w \rceil$, where $w \approx r$, $X = (X_1, \ldots, X_d$ are iid Gaussian random variables, and b is a scalar chosen uniformly at random from $[0, w]$. Conceptually, then, our fash function takes p, projects it on to X, shifts it by some amount b, then determines the interval of length w containing it.

This only has a small improvement over the $1/c$ bound, however (please refer to slides for more details).

Therefore we consider alternative ways of projection.

6.2 Ball Lattice Hashing

This covers work by [AI06].

Instead of projecting onto $\mathbb{R}^1$, we project onto $\mathbb{R}^t$ for constant t. We quantize with a lattice of balls—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.

For more details and analysis, as well as other LSH schemes (including *data dependent hashing*, in which we cluster related data until it is "random," which presents better bounds) and schemes (including the Jaccard coefficient we explored in pset 2), please refer to the slides.

References

[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.

[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.