Algorithms for Big Data Fall 2020

Lecture 12 — October 28, 2020

Prof. Cyrus Rashtchian *Topics: Nearest Neighbor Search Intro*

Overview. Today we talked about approximate nearest neighbor search.

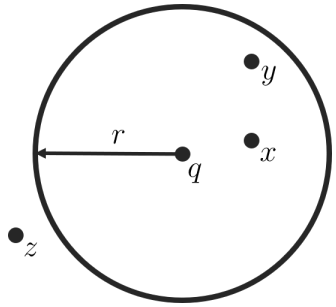

Figure 1: Query q has nearest neighbor x and r-near-neighbors x and y. The distance r similarity join of these four points is $\{\{q, x\}, \{x, y\}, \{y, q\}\}$.

1 Near Neighbor Search

We want to find the most similar element of a dataset to a query. What does most similar mean? In the context of a metric space, we can say that it is the point with the smallest distance.

We usually consider a dataset X of n points in d-dimensional "feature space" $X \subseteq \mathbb{R}^d$. Query $q \in \mathbb{R}^d$. Goal to return the "most similar" point in X to q under some norm (for example ℓ_1 or ℓ_2).

Where does this feature space come from? We assume that (i) complicated objects, such as images, products, or genomic sequences, have been represented by vectors, and (ii) similar vectors correspond to similar objects. This task often serves as an intermediate step when searching for related objects, such as images, products, or genomic sequences. When it is challenging to directly compare complicated objects, an effective workaround involves associating each object with a vector and comparing objects based on their corresponding vectors. For example, a binary vector may describe features of an image, or a string of ACGT characters may represent a strand of DNA. The hope is that similar vectors correspond to similar objects and vice versa.

We will design and analyze algorithms for solving the near(est) neighbor problem, approximately, for vectors. There are also extensions to other metric spaces, but these are more complicated. In general, there is not a perfect theory for when near neighbor search can be solved efficiently.

2 Review of Metric Spaces

We discussed this last time, but we will focus on some vectors (and review for anyone who missed it the first time).

Definition 1. *A* metric space *is a pair $(\mathcal{X}, d_{\mathcal{X}})$, where $\mathcal{X}$ is a set and $d_{\mathcal{X}} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$ is a nonnegative* distance function *on pairs in $\mathcal{X}$. The function $d_{\mathcal{X}}$ satisfies three properties:*

1. $d_{\mathcal{X}}(x, y) = d_{\mathcal{X}}(y, x)$,
2. $d_{\mathcal{X}}(x, y) = 0$ *if and only if* $x = y$, *and*
3. $d_{\mathcal{X}}(x, y) \leq d_{\mathcal{X}}(x, z) + d_{\mathcal{X}}(z, y)$ *for all* $x, y, z \in \mathcal{X}$.

The third property (known as the *triangle inequality*) differentiates metric spaces from arbitrary notions of distance, and it implies much useful structure.

Examples of Vector Metrics

- Hamming distance: $([k]^d, d_{\mathsf{H}})$, where k is a positive integer, $[k]$ denotes the set $\{1, 2, \ldots, k\}$, and $d_{\mathsf{H}}(x, y)$ counts the coordinates that differ in x and y. Usually we will be interested in $k = 2$ and denote the set of vectors by $\{0, 1\}^d$.
- Euclidean distance: $(\mathbb{R}^d, d_{\ell_2})$, where $d_{\ell_2}(x, y) = \|x - y\|_2 = \left(\sum_{i=1}^d (x_i - y_i)^2\right)^{1/2}$.
- Cosine distance: $(\mathcal{S}^{d-1}, d_{\ell_2})$ is the important special case of Euclidean distance that restricts to unit vectors on the d-dimensional sphere $\mathcal{S}^{d-1} = \{x \in \mathbb{R}^d : \sum_{i=1}^d x_i^2 = 1\}$.

Vectors may be very similar in one metric but very different in another. For example, the binary strings $0101 \cdots 0101$ and $1010 \cdots 1010$ in $\{0, 1\}^d$ have Hamming distance d but edit distance two. On the other hand, some distances may be expressed as others. For binary vectors, Euclidean distance is the square root of Hamming distance, which itself is the shortest path metric on the standard hypercube, that is, the graph with vertices $\{0, 1\}^d$ and edges connecting vectors with Hamming distance one.

For a vector $x \in \mathbb{R}^d$, the ℓ_p norm of x is defined for $p \in [1, \infty)$ as

$$\|x\|_p = \left(\sum_{i=1}^d |x_i|^p\right)^{1/p}.$$

Although one can extend the definition above to $0 < p < 1$, these functionals do not in general define norms, since they do not always satisfy the triangle inequality. On the other hand, for $p = 0$ and $p = \infty$, there are suitable generalizations. By convention $\|x\|_0$ is defined as the sparsity of x, that is, the number of nonzero elements, and $\|x\|_\infty$ is defined as the maximum magnitude of a coordinate; that is,

$$\|x\|_\infty = \max_{i \in [d]} |x_i|.$$

In other words, we have analogous spaces ℓ_0 and ℓ_∞. These norms induce a metric d_{ℓ_p} through

$$d_{\ell_p}(x, y) = \|x - y\|_p.$$

Most importantly for us will be $p \in \{1, 2, \infty\}$. Notice that d_{ℓ_1} reduces to Hamming distance on binary vectors, and d_{ℓ_2} is Euclidean distance.

3 Warm-up: Finding close points in Hamming distance

Before getting into formalities, let's work through a special case. Say we have a dataset of bit-vectors $S \subseteq \{0, 1\}^d$ in some high-dimensional space (so d is large). Say that $n = |S|$. We want to pre-process

S in an efficient way to support near neighbor queries. That is, given a new point $q \in \{0,1\}^d$, quickly find a point $x \in S$ with Hamming distance at most r from q, for a given distance r.

One way to do this, naively, is to perform a linear scan. We compare q to each point in S, and output the closest point $x \in S$. There are $n = |S|$ comparisons, and each takes time d, for a total of $O(nd)$ time.

If n is large, we want to do better by using a randomized approximation algorithm. Specifically, we will find a point with $d(x,y) \leq 2r$ if there exists one in S with distance at most r. So our "close" point might not be the closest, but it'll be close enough up to this factor of two (which we can improve to a smaller constant).

We show how to do this using roughly $O(n^{3/2})$ space and $O(\sqrt{n})$ query time, so we will trade-off using more space to answer queries faster. this is beneficial if we are answering a large number of near neighbor queries.

3.1 The hash tables

We will build our data structure using a bunch of independent hash tables. We first describe one such table. Let $k < d$ be a parameter that we will set shortly. We define a random hash function by choosing a set of k indices (with replacement, for a simple analysis). Denote them as $I = \{i_1, i_2, \ldots, i_k\}$. Then we project each vector $x \in S$ onto these k coordinates.

$$h(x) = x|_I = (x_{i_1} x_{i_2} \cdots x_{i_k}).$$

This gives us a new vector $h(x)$ that is k bits long.

We build a hash table based on these k bits. There are at most 2^k buckets, some of which may be empty. And two vectors x and y are in the same bucket if $h(x) = h(y)$.

Intuition. We claim that similar vectors are more likely to hash to the same bucket than dissimilar vectors. Specifically,

Lemma 2. *For the hash function defined above, we have*

- *If* $d(x,y) \leq r$, *then* $\Pr[h(x) = h(y)] \geq (1 - \frac{r}{d})^k$.
- *If* $d(x,y) \geq 2r$, *then* $\Pr[h(x) = h(y)] \leq (1 - \frac{2r}{d})^k$.

Proof. We choose each coordinate i independently from the d coordinates. For each, we have the following observations.

If $d(x,y) \leq r$, then x and y disagree in at most r positions (agree in at least $d-r$). So the probability that $x_i = y_i$ over a random i is at least $\frac{d-r}{d} = 1 - \frac{r}{d}$. Repeating this k times independently gives $\Pr[h(x) = h(y)] \geq (1 - \frac{r}{d})^k$.

If $d(x,y) \geq 2r$, then x and y agree on at most $2r$ positions and disagree in at least $d - 2r$. For a random coordinate, we have $x_i = y_i$ with probability at most $1 - \frac{2r}{d}$.

□

3.2 Building a data structure

We will use the above hash tables (multiple copies) to design an efficient near neighbor data structure.

First we will denote

- $p_1 = 1 - \frac{r}{d} \approx e^{-r/d}$
- $p_2 = 1 - \frac{2r}{d} \approx e^{-2r/d}$

We set $k = \frac{d}{2r} \cdot \ln n$.

Therefore, the probability of close collisions (hashing to the same bucket with x and y are close) is

$$p_1^k \approx e^{-rk/d} = 1/\sqrt{n}.$$

And for far collisions it is

$$p_2^k \approx e^{-2rk/d} = 1/n.$$

We will repeat the hashing process $L = O(\sqrt{n})$ times. Therefore, there will be L hash tables (each storing all n points). So the total storage is $O(nd \cdot L) = O(n^{3/2}d)$.

Now we will analyze this scheme formally and show that it works. We will also replace the factor of 2 with any factor $1 + \varepsilon$ with $\varepsilon > 0$.

3.3 Algorithm

To recap, the overall algorithm is the following. We state it generally for a approximation parameter $c \geq 1$, but above we have only considered $c = 2$ being fixed.

LSH-based ANNS for Hamming distance

- **Input:** Dataset $X \subseteq \{0,1\}^d$, and distance threshold $r \in [d]$, and approximation parameter c.
- **Parameters:** Number of bits k per hash, and number of hash tables L.
- **Preprocess:**
  1. Choose L hash functions $h_1, \ldots, h_L$ by choosing k bit positions independently for each (with replacement).
  2. For each $x \in X$, hash x based on the L hash functions $h_1(x), \ldots, h_L(x)$.
  3. Store X and store the L hash tables (which are partitions of X based on h_i)
- **Query:**
  1. On query point $q \in \{0,1\}^d$, hash q based on the L hash functions $h_1(q), \ldots, h_L(q)$.
  2. For each $i = 1, 2, \ldots, L$, compare q against all $x \in X$ that have the same hash $h_i(x) = h_i(q)$.
  3. As soon as you find a point $y \in X$ with $d(q, y) \leq cr$, then output y as the near neighbor for q.
  4. If you compare against $100L$ input points, and find that none of them are close enough to q, then return "no close pair to q in X" and terminate.

3.4 Overall space and time

Space: There are $n = |X|$ input points, and each take d bits. So the space is $O(nd)$ plus what is needed for the hash tables. There are L hash tables, so the space is $O(nd + Ln \log n)$, because we need to store the index of the hash buckets (the hash value $h(x)$) and also a pointer to the actual input point (each takes roughly $O(\log n)$ bits). Note that each hash table is a partition of X, so the total number of points in each hash table is exactly n.

Time: We need to be a bit more careful about the time, because the hashing is random. What we will show is that we can stop after looking at $100L$ elements, and if none of them have distance at most cr, then we can conclude that no close pair exists in X.

Lemma 3. *If there is some $y \in X$ with $d(q, y) \leq cr$, then we will find it with probability 0.99 after comparing q against at most $100L$ points (for L and k set as above).*

Proof. Exercise (use Markov's inequality). □

4 Other Flavors of Similarity Search

Computational problems fall into two broad categories. In the first, the goal is to support similarity-based queries after preprocessing a set of input points. As a use case, we might want to find a small set of advertisements, restaurants, or job listings that are relevant for a given user. The second category concerns *all-pairs* problems, where the goal is to quickly find all close pairs in a dataset. Clustering is an all-pairs task, where we want to partition a dataset into smaller groups with mutually similar points.

Definition 4 (Near(est) Neighbor Search)**.** *Fix a metric space $(\mathcal{X}, d_{\mathcal{X}})$, a threshold r, and a dataset $S \subseteq \mathcal{X}$. After preprocessing S, the* r-near-neighbor *problem for a query $q \in \mathcal{X}$ is to output all $x \in S$ with $d_{\mathcal{X}}(q, x) \leq r$. The* nearest-neighbor *problem returns $\mathsf{argmin}_{x \in S}\ d_{\mathcal{X}}(q, x)$. For a parameter $k \geq 1$, the* k-nearest-neighbor *problem for a query $q \in \mathcal{X}$ returns a set $T \subseteq S$ with $|T| = k$ such that $d_{\mathcal{X}}(q, x) \leq d_{\mathcal{X}}(q, y)$ for any $x \in T$ and $y \in S \setminus T$.*

Definition 5 (Similarity Join)**.** *The* similarity join *at distance r of a set S in a metric space $(\mathcal{X}, d_{\mathcal{X}})$ is the set of pairs in S with distance at most r, that is,*

$$\{\{x, y\} \subseteq S \mid d_{\mathcal{X}}(x, y) \leq r, x \neq y\}.$$

Figure 1 depicts nearest neighbor, r-near-neighbor, and similarity join for a small example. Approximate versions of these problems allow pairs at distance cr for $c \geq 1$.

References

[1] A. Andoni, P. Indyk, I. Razenshteyn, Approximate Nearest Neighbor Search in High Dimensions. Proceedings of ICM, 2018 `https://arxiv.org/abs/1806.09823`

[2] A, Andoni, T. Laarhoven, I. Razenshteyn, and E. Waingarten Optimal hashing-based time-space trade-offs for approximate near neighbors. SODA 2017

[3] A. Andoni, I. Razenshteyn, N. Shekel Nosatzki LSH Forest: Practical Algorithms Made Theoretical SODA, 2017

[4] Har-Peled, Sariel, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. Theory of computing 8.1 (2012): 321-350.