Data Science Through a Geometric Lens Fall 2019

Lecture 08 — October 28, 2019

Prof. Cyrus Rashtchian *Topics: LSH for Hamming Distance*

Overview. Today we talk about approximate near neighbor search (ANNS) for Hamming distance.

1 Locality Sensitive Hashing (LSH)

An influential line of work studies *approximate similarity search*, which both relaxes the distance threshold r and forgives a small percentage of false negatives. A main tool is Locality Sensitive Hashing (LSH), pioneered by Indyk and Motwani and Kushilevitz, Ostrovsky, and Rabani. An LSH family is a distribution over hash functions such that near points map to the same bucket with a higher probability than far points do. For Hamming space, an example LSH family chooses k random coordinates in $\{1, 2, \ldots, d\}$ and maps a vector to its k-bit projection on these coordinates. In any similarity join or clustering algorithm, LSH provides a way to reduce the running time through filtering. A modified algorithm would first build several hash tables based on independent hash functions from an LSH family. Then, the algorithm only compares points that have ever been hashed to the same bucket. The accuracy of this method can be improved by using more hash tables. In many cases, hashing the dataset some number of times is much faster than performing a large number of comparisons. A central motivation is Approximate Near Neighbor Search (ANNS).

Definition 1. *Let $S \subseteq \mathcal{X}$ be a dataset in a metric space $(\mathcal{X}, d_{\mathcal{X}})$. The* (c, r)-ANN problem *for $c > 1$ and $r \in \mathbb{R}_{\geq 0}$ is to efficiently pre-process S to quickly answer the following query. Given $q \in \mathcal{X}$, either return $x \in S$ with $d_{\mathcal{X}}(q, x) \leq cr$, or report that no such point exists.*

The algorithm knows the approximation c and the threshold r ahead of time. Therefore, we focus on data structures parameterized by c, r. A useful primitive is an LSH family.

Definition 2. *Let r and $c \geq 1$ and $p_1, p_2 \in (0, 1]$ with $p_1 > p_2$ be parameters. A hash family $\mathcal{H}$ in a metric space $(\mathcal{X}, d_{\mathcal{X}})$ is* (r, cr, p_1, p_2)-**sensitive** *if the following two conditions are satisfied:*

1. $\Pr[h(x) = h(y)] \geq p_1$ *for all $x, y \in \mathcal{X}$ with $d_{\mathcal{X}}(x, y) \leq r$,*
2. $\Pr[h(x) = h(y)] \leq p_2$ *for all $x, y \in \mathcal{X}$ with $d_{\mathcal{X}}(x, y) \geq cr$,*

where the probability is over sampling a uniformly random $h \in \mathcal{H}$.

2 Approximate Nearest Neighbor Search

Theorem 3. *For Hamming distance on $\{0, 1\}^d$, there exists a randomized (c, r)-ANN data structure and randomized query algorithm with the following three properties:*

1. *Space:* $O(n^{1+1/c} + nd)$
2. *Query Time:* $O(dn^{1/c})$
3. *Success Probability:* 0.99.

We can increase the success probability to anything large by running multiple independent copies of the data structure (analogous the ++ versions of the sketching algorithms). In practice, it's often best to test different parameters and see what works well for the dataset at hand.

2.1 The data structure

The data structure will store hash tables determined by hashing input vectors to subsets of coordinates. Let $S \subseteq \{1, 2, \ldots, d\}$ be a subset of coordinates. We will let k denote the size of S so that $|S| = k$, where we will set k later to trade-off accuracy and space.

For a vector $x \in \{0,1\}^d$, define the projection onto S as the vector $x|_S \in \{0,1\}^k$ consisting of k bits. We can use this to define a hash function $h_S : \{0,1\}^d \to \{0,1\}^k$, where $h_S(x) = x|_S$. By doing so, we can hash every vector into one of 2^k possible hash buckets.

On query $q \in \{0,1\}^d$, we will compare q to all points x in the dataset such that $h_S(q) = h_S(x)$. In other words, compare q to all elements that hash to the same bucket.

For a dataset of size n, the space usage is $O(nd)$, because we store each vector x in exactly one hash bucket. The query time will depend on the number of other points in the same bucket as q.

Lemma 4. *Let $\mathcal{E}$ be the event that a point x with $d_{\mathsf{H}}(q, x) \geq cr$ ends up in the same hash bucket as q does. Then,*

$$\Pr_S[\mathcal{E}] \leq \left(1 - \frac{cr}{d}\right)^k.$$

Proof. The event $\mathcal{E}$ occurs every coordinate $i \in S$ has the property that $x_i = q_i$. Since we have assumed that $d_{\mathsf{H}}(q, x) \geq cr$, we know that there are at least cr coordinates that are 'bad' in the sense that $x_i \neq q_i$ (among d total coordinates). Therefore, with probability at least $\frac{cr}{d}$, a coordinate in S will 'separate' q and x. On the other hand, with probability $1 - \frac{cr}{d}$, the coordinate will have the same value for q and x. We choose each of the k coordinates independently to form S, and therefore, the probability that $h_S(x) = h_S(q)$ is at most $\left(1 - \frac{cr}{d}\right)^k$. □

The expected size of q's hash bucket is at most $n\left(1 - \frac{cr}{d}\right)^k$. We will choose k to be the minimum integer such that $\left(1 - \frac{cr}{d}\right)^k \leq 1$.

Lemma 5. *The expected size of q's hash bucket is at most one, and therefore, the expected query time is $O(d)$.*

Lemma 6. *Let $\mathcal{E}'$ be the event that a point x with $d_{\mathsf{H}}(q, x) \leq r$ ends up in the same hash bucket as q does. Then,*

$$\Pr_S[\mathcal{E}'] \geq \left(1 - \frac{r}{d}\right)^k.$$

Proof. The event $\mathcal{E}'$ occurs if every coordinate $i \in S$ has the property that $x_i = q_i$. Since we have assumed that $d_{\mathsf{H}}(q, x) \leq r$, we know that there are at least r coordinates that are 'bad' in the sense that $x_i \neq q_i$ (among d total coordinates). Therefore, with probability at least $\frac{r}{d}$, a coordinate in S will 'separate' q and x. On the other hand, with probability $1 - \frac{r}{d}$, the coordinate will have the same value for q and x. We choose each of the k coordinates independently to form S, and therefore, the probability that $h_S(x) = h_S(q)$ is at most $\left(1 - \frac{r}{d}\right)^k$. □

Lemmas 4 and 6 show is that this is an LSH family with $p_1^k = (1 - r/d)^k$ and $p_2^k = (1 - cr/d)^k$.

2.2 The overall ANNS algorithm for Hamming distance

We will build our data structure using a bunch of independent hash tables. We first describe one such table. Let $k < d$ be a as set in the above lemma (roughly $k = \frac{r}{cd} \log n$).

We choose a set of k indices (with replacement, for a simple analysis). Denote them as $S = \{i_1, i_2, \ldots, i_k\}$. Then we project each vector $x \in S$ onto these k coordinates.

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

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 $h(x) = h(y)$.

2.3 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

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

We set $k = \lceil \frac{d}{2r} \cdot \ln n \rceil$. Let's ignore the ceiling for simplicity.

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

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

And for far collisions it is

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

We will repeat the hashing process $L = 1/p_1^k = n^{1/c}$ times. There will be L hash tables (each storing all n points). So the total storage is $O(nd \cdot L) = O(n^{3/2}d)$.

Summarizing, we have the following:

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 $i = 1, 2, \ldots, L$, 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.

2.4 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 7. *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. We know from Lemma 5 that there is 1 expected far point in each bucket as q in the hash table. We are considering L different hash tables, and thus, there are L far points in the same hash bucket as q in one of the tables (by linearity of expectation, over the L hash tables). By Markov's inequality, the probability that we see $100L$ far points is at most $1/100 = 0.01$.

The algorithm stops after comparing against $100L$ points, so we never have more comparisons than this. But with probability 0.99, we have fewer than $100L$ far points to compare against anyway.

From Lemma 6, the probability that we see a close point in one hash table is $p_1^k = n^{1/c}$. We can set $L = 10/p_1^k$ to be the number of hash tables. The probability of failure is the probability that we don't see a close point in any of the hash tables. We choose the hash functions independently, so the probability of failure is

$$(1 - p_1^k)^L \approx e^{-L \cdot p_1^k} = e^{-10}.$$

Therefore, with probability $1 - e^{-10}$, which is very close to one, we will see a close point to q in one of the hash tables, and therefore we will succeed. □

Recap. The constants in the above proof can be optimized in many ways. We set k and L for convenience, but a tighter analysis with better constants is possible. But overall, we have showed that the number of comparisons is bounded by $O(L) = O(1/p_1^k) = O(n^{1/c})$ as desired (each takes d time). Similarly, the space is bounded by $O(nL + nd) = O(n^{1+1/c} + nd)$, also as claimed.

We discuss in the next section how this proof can be generalized. In particular, we only used properties of Hamming distance to design the hash functions. Other than that, the parameters were set in terms of p_1 and p_2, and the same data structure and analysis holds verbatim for other metrics and LSH families as well. This is one of the powerful aspects of the LSH definition.

We also note that an improvement for Hamming distance is known if we allow the ANNS algorithm to depend on the data when building the data structure [3]. A data dependent improvement for Euclidean distance is also known [2].

3 Other distance metrics

To solve the (c, r)-ANN problem, it actually suffices to find (r, cr, p_1, p_2)-sensitive hash families. Then, the above analysis extends analogously.

If the dataset has n points, then any efficiently-computable family $\mathcal{H}$ leads to a data structure for the (c, r)-ANN problem with space $n^{1+\rho+o(1)}$ and query time $n^{\rho+o(1)}$ where

$$\rho = \frac{\log(1/p_1)}{\log(1/p_2)}.$$

For Hamming distance, (r, cr, p_1, p_2)-sensitive families are known achieving $\rho = 1/c$. For Euclidean distance, the value is $\rho = 1/c^2$. The time and space bounds come from building n^ρ independent hash tables, each having n points.[1]

For details, we refer readers to the well-written references on LSH and applications [1, 4].

Next time, we will see LSH for Euclidean distance with worse parameters than the best possible (we will only achieve $\rho = 1/c$). There are also LSH families for certain ℓ_p norms, for earth movers distance, and for set similarity functions (e.g., Jaccard). It's a major open question to find a good LSH for edit distance and other more complicated metrics. Metric embeddings, such as Bourgain's embedding, can be used to solve ANNS for general metric spaces but with a worse approximation factor. For example, using Bourgain's embedding, we can solve ANNS for any metric space with an $O(\log n)$ approximation factor by first embedding in Euclidean distance.

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.

[1] The $o(1)$ factors correspond to the space overhead for the points (which is assumed to be $n^{o(1)}$ per point) and the time for computing a hash function (which is also assumed to be $n^{o(1)}$).