Data dependent LSH for ANN:

Eliezer Zimble Uni:ez2313

December 20, 2023

Abstract: Classic LSH [IM98], [HIM12] uses space partitioning to solve the ANN problem with hashing independent of the input dataset. In this paper I will survey the development of data-dependent LSH as a solution that makes use of the inherent structure of *any* arbitrary dataset. I will primarily focus on [ARS17]'s contribution of a practical algorithm that still outperforms classic LSH on adversarial queries. I will also discuss [ARS17] in the context of the work of [AR15] in establishing an optimal data-dependent LSH and [AB22] work towards adaptivity to "niceness" of the dataset.

1 Introduction

1.1 Background and Problem Statement:

Nearest Neighbor Search (NNS) is a fundamental problem in ML, information retrieval and computational geometry. In the NNS problem we are given a set P of n points in d dimensional metric space X, a distance/similarity metric D, and query point q. Our goal is to find the near or nearest neighbor $p \in P$. To solve the problem, we build a data structure to preprocess the dataset to allow us to efficiently answer all queries using as little space as possible. The exact solution is impractical for higher dimensional space as it suffers from "curse of dimensionality": either space or query time exponential is exponential in d. Research attention has therefore turned to finding efficient data structures for approximate solutions.

ANN Problem Statement: In the $(c, r)-$Approximate Near Neighbor Search problem (ANN), we are given a set P of n points in d dimensional metric space X, an approximation factor c and a given distance (or similarity measure) r. We preprocess the points with some data structure such that given a query point q, if a near neighbor p^* within distance r from q exists, then with high probability we can efficiently return an **approximate** near neighbor by returning any point $p \in P$ such that p is at most distance cr from q.

Definition of LSH family: Locality Sensitive Hashing intuitively aims to create randomized partition of the space such that "close" points have a high probability of hashing to the same bin, while "far" points have a low probability. Formally, a distribution $\mathcal{H}$ over hash functions h is a $(r, cr, p_1, p_2)-$sensitive LSH family if $\forall p \in P$ and $\forall q \in X$:

$$\text{If } D(p,q) \leq r \text{ then, } Pr_h\left[h(p) = h(q)\right] \geq p_1$$

$$\text{If } D(p,q) > cr \text{ then, } Pr_h\left[h(p) = h(q)\right] \leq p_2$$

We define the quality of an LSH family as:

$$\rho\left(\mathcal{H}\right) := \rho = log(1/p_1)/log(1/p_2)$$

Theorem 1: ([IM98], [HIM12]) *Given a $(r, cr, p_1, p_2)-$sensitive LSH family, there exists a $(c, r) - ANN$ data structure with query time $O\left(n^{\rho}\right)$ and extra space usage $O\left(n^{1+\rho}\right)$.*

Main idea[HIM12]: The hash functions can be composed together k times to amplify the difference between p_1 and p_2 by achieving $p_1' = p_1^k$ and $p_2' = p_2^k$. Applying the LSH family to the dataset produces a dictionary where we can lookup the bin for each point $p \in P$. Given a query point q, we use the dictionary to look up the bin and see if any of the contents $p \in h(q)$ are within distance cr from q. The query process has probability of success $Pr[success] \geq p_1^k$. Setting $k = \lceil log_{1/p_2}(n) \rceil$ gives us:

$$\begin{aligned} Pr[success] &\geq p_1^k \\ &\geq p_1^{log_{1/p_2}(n)+1} \\ &= p_1 \cdot n^{-\frac{log(1/p_1)}{log(1/p_2)}} \\ &= p_1 \cdot n^{-\rho} \end{aligned}$$

As the probability of success is relatively low, we boost the success probability by repeating the process for $O(n^\rho)$ number of independent dictionaries. We thereby achieve a constant probability of success.

1.2 Bit partitioning LSH

[IM98], [HIM12] conclude solution by showing that LSH families do exist, and in particular develop Bit partitioning LSH which [ARS17] use as a primitive. See [AIR18] section 2.3 for discussion of other LSH families.

For a Hamming space $\{0,1\}^d$, the bit partitioning LSH family [IM98], [HIM12] operates as follows. A coordinate $i \in [d]$ is randomly uniformly chosen. The partition is then dividing the points based on their value at coordinate i. Namely the two buckets of the partition are $\{p : p_i = 0\}$ and $\{p : p_i = 1\}$. Note: See section 5 and [AB22] ahead for discussion of potential improvement when the bits are selected according to a different distribution.

Theorem 2*: [IM98], [HIM12] Bit partitioning LSH family achieves $p_1 = 1 - \frac{r}{d}$ and $p_2 = 1 - \frac{cr}{d}$ such that:*

$$\rho = \frac{log(1/p_1)}{log(1/p_2)} = \frac{log(1-\frac{r}{d})^{-1}}{log(1-\frac{cr}{d})^{-1}} \leq \frac{1}{c}$$

Proof. Let x and y be fixed close points such that $D(x,y) \leq r$. Close points x and y will be placed in different buckets only when the coordinate i selected for partitioning is one on which they differ such that $x_i \neq q_i$. As the coordinate for partitioning is selected uniformly with probability $\frac{1}{d}$ and close points differ on at most r coordinates, there is at most $\frac{r}{d}$ likelihood that x and y hash into different buckets. From the complement in probability there is therefore at least $1 - \frac{r}{d}$ probability that close points hash into the same bucket. Therefore $p_1 = 1 - \frac{r}{d}$.

By similar analysis, for far points x and y such that $D(x,y) > cr$, the points hash to different buckets when the coordinate selected is one of the $> cr$ on which they differ. As each coordinate has probability $\frac{1}{d}$ of being selected, the probability the points hash to different buckets is $> \frac{cr}{d}$. Therefore the probability far points hash to the same bucket is $\leq 1 - \frac{cr}{d}$. □

Per *Theorem 1*, Bit partitioning LSH solves $(c,r) - ANN$ for Hamming space with $\rho = \frac{1}{c}$.

1.3 Limitations of data-independent LSH

Research naturally focused on determining the best values for ρ that can be achieved using data-independent LSH families. The question was answered in establishing that for data-independent LSH the best quality for Hamming space is bounded by $\rho \geq \frac{1}{c} - o(1)$, and for Euclidean space is bounded by $\rho \geq \frac{1}{c^2} - o(1)$ [AIR18].

1.4 Data dependent LSH

Data dependent LSH allows the randomized hash family itself to depend upon the input dataset. As noted above, the algorithms we focus on do not assume any specific structure to the data. Rather, the incorporation of the input data applies to any arbitrary dataset. We will see that data-dependent LSH families can achieve better quality than classic LSH [AR15], [ARS17], [AB22]. The improvement demonstrates that *any* generic dataset has some structure that makes solutions to $(c, r) - ANN$ easier [AIR18].

1.5 Outline of paper:

In section 2 we establish the improvements and optimal results of data-dependent LSH from [AR15] but note that the result is not practical.

Our primary focus is to survey [ARS17] which offers a simple data-dependent LSH algorithm (though of lower quality ρ): section 3 outlines their algorithm, and section 4 proves its correctness.

Section 5 discusses [AB22] work which extends the algorithm of [ARS17]. [AB22] obtains the same worst-case guarantees but also adapts to allow for better quality if the dataset does exhibit "nice" structure. Section 5 concludes a discussion of the results of [ARS17] and future directions.

2 Improvement and limitations of Data dependent LSH:

A natural next question for research is whether Data dependent LSH can do better than classic LSH and if so what is the best ρ that can be achieved? [AR15] answered both of these questions in presenting an algorithm that achieves an improved $\rho = \frac{1}{2c^2-1}$ for Euclidean space with $c > 1$ and established that such a ρ is optimal.

The main idea of [AR15] is as follows: The "good case" for data-independent Ball-carving LSH in Euclidean space is when the points are randomly distributed on a unit sphere. We therefore want to reduce ANN on a generic dataset into ANN on a randomly distributed dataset. [AR15] performs the reduction by recursively decomposing the dataset into dense-clusters, and "pseudo-random" remainders for which almost all data points are $\approx \sqrt{2}$ apart (akin to random points on a unit sphere which are essentially $\sqrt{2}$-separated). See [AIR18] section 4.4 for a more detailed overview of the [AR15] algorithm.

The main drawback of the [AR15] algorithm is that the decomposition is complex and therefore not practical. For the majority of the survey we will focus on a simple algorithm for data-dependent LSH as presented in [ARS17].

3 [ARS17] Algorithm: construction of data structure and query algorithm

3.1 Overview of [ARS17]

[ARS17] offers a simpler data-dependent LSH algorithm for d dimensional Hamming space. The exponent ρ is better than data-independent LSH but is suboptimal for data-dependent LSH (as established by [AR15]). The algorithm is performs a random partition of the dataset to build a decision tree, augmenting each node with a constant number k of "pivot" points. The query is checked against each set of pivot points while descending down the tree. To increase the success probability, the process is repeated $O(n^\rho)$ number of times. The algorithm achieves the suboptimal exponent $\rho \approx \frac{1}{ln(4)\cdot c}$, and results in query time $O(n^\rho \cdot d^2 \cdot k)$ and space is $O(n^{1+\rho}k + nd)$ for $k \geq 1$.

In particular, we can express this in pseudocode as:

BuildForest(P)
1 Build an LSH tree using Bit partitioning.
2 For each node v in the tree, select and store carefully chosen constant number of "pivot" points.
3 Repeat steps (1) and (2) for n^ρ independent number of times and output the result as an LSH forest F.

The $QueryForest(F, q)$ algorithm, given a query point q, iterates through each tree of the LSH forest F, and upon reaching a node checks whether any of its associated "pivots" contains a near neighbor p such that $D(p, q) \leq cr$. If such a p is found, then the query algorithm terminates and returns p.

Note: For the purpose of the analysis, the paper assumes that for query q there is only a single near neighbor $p^* \in P$ such that $D(p^*, q) \leq r$. The assumption can be relaxed by including an additional $\Theta\left(log(n)\right)$ uniformly random pivots in addition to the pivots stored at each node [ARS17].

Outline of section 3: In the following subsections we will formalize steps (1) and (2) of the algorithm as subroutines. In section 3.2 we will formalize the construction of each LSH tree using bit partitioning, and in section 3.3 formalize the selection of the "pivots" for each node as subroutines of the algorithm. In section 3.4 we will use the subroutines to formally present the full algorithm.

3.2 LSH tree construction using bit partitioning

The LSH forest construction recursively uses the LSH Bit partitioning routine from above to create a forest of n^ρ decision trees in the following manner.

1. Choose a random permutation of the coordinates $\pi : [d] \to [d]$ and apply the permutation to each point $p \in P$.
2. Use Bit partitioning LSH recursively to build a decision tree on the permuted points until obtaining leaves with only one point.
3. The process is repeated independently n^ρ number of times.

Note: The Bit partitioning LSH is independent of the dataset as the coordinate chosen for partitioning is uniformly selected. In contrast, the LSH forest is data dependent: the recursive calls depend upon the distribution of points in the dataset.

Letting BuildTree(S) be the subroutine that constructs an LSH decision tree, we can formalize step (2) in the following way. Let v be a node of the tree with associated set $S \subseteq P$. Let $\mathcal{R}$ denote a random partition on metric space X, and $\mathcal{R}(p)$ denote the part of $\mathcal{R}$ that the point p lies in. AddPivots(v, k) is a subroutine defined ahead in section 3.3. We then have:

```
BuildTree(S)
1  Create node v with associated set S.
   #Create pivots and store in v
2  T_v=AddPivots(v, k) and store in v.
3  if |S| = 1
4     then return v
   #Use Bit partitioning LSH to create children nodes.
5  Else: Uniformly select a coordinate i ∈ [d] and create partition R_v = {p : p_i = 0} ∪ {p : p_i = 0}.
6  for U ∈ R_v:
7        do : if |S ∩ U| ≠ 0
8                then add BuildTree(S ∩ U) as a child of v.
```

3.3 Pivot selection and augmentation

While constructing each Bit partitioning LSH tree, the algorithm adds a constant number of pivots to each node. For a subset $S \subseteq P$, the pivots are chosen by computing the mean of S and sorting each $p \in S$ according to ℓ_1 distance from the mean. The first k points that are sufficiently far from each other, namely, that are pairwise $(c-1)r$ apart from each other are stored as the pivots [ARS17].

Formally, let v be a node of the tree, and S_v be the associated points $p \in P$ stored in the partition $\mathcal{R}_v$. Namely $S_v = P \cap R_v$. Set constant k as the number of pivots per node. T_v is a set that will store the pivot points for v. We can then express the pivot augmentation as the following subroutine:

```
AddPivots(v, k)
1  S_v ← the associated points p ∈ P in R_v.
2  Initialize T_v = ∅
3  Compute the mean of S_v and store as p^c.
4  for p ∈ S_v in terms of increasing ℓ_1 distance from p^c:
         do
5           if the ℓ_1 distance between p and T_v is at least (c − 1)r
6              then T_v ← T_v ∪ {p}
7                   if |T_v| = k :
8                      then return T_v
9  return T_v
```

3.4 Formal algorithm

We can now formally express the construction algorithm for the LSH forest as:

```
BuildForest(P)
1  Initialize an empty forest F.
2  for i = 1 to n^ρ:
3        do Choose a random permutation of the coordinates π : [d] → [d],
             and apply the permutation to each point p ∈ P. Let P' be the permuted dataset.
4            Add T_i=BuildTree(P') to F.
5  return F
```

We can formally express the query algorithm as:

```
QueryForest(F, q)
1  For each tree T ∈ F with root r_i:
2        do QueryTree(r, q)
```

where $\text{QueryTree}(r, v)$ is defined for tree node v and query q as:

```
QueryTree(v,q)
1  if ∃p ∈ T_v within distance cr from q:
2     then
3          Return: p
   #Check the child node in the same bucket of the partition as q.
4  if there is a child v' of v that is associated with R_v(q)
5     then
6          return QueryTree(v', q)
7     else
8          return "There is no near neighbor of q".
```

4 Proof of correctness

We will now analyze and now prove the correctness of the [ARS17] algorithm as presented in section 3.4.

4.1 Overview of analysis

The crux of the analysis is a proof by induction that hinges upon the following inequality:

$$(*)\ Pr_{\mathcal{R}}\left[\mathcal{R}(q) = \mathcal{R}(p^*)\right] \times \mathbb{E}_{\mathcal{R}}\left[\left(\frac{|S \cap \mathcal{R}(q)|}{|S|}\right)^{-\rho} \middle| \mathcal{R}(q) = \mathcal{R}(p^*)\right] \geq 1$$

where we are given a dataset P of n points, q is a query point and $p^* \in P$ is a near neighbor of q. Let $0 < \rho \leq 1$ be the smallest real number such that the equation holds for every subset $S \subseteq P$ where $p^* \in S$.

Intuitively, the inequality $(*)$ gives us a bound for the decrease factor in the bucket size containing the near neighbor throughout the operation of **any** LSH algorithm. An algorithm satisfying the

inequality returns a near neighbor within distance cr from q with probability $\geq n^{-\rho}$. As above, we can therefore repeat the process for $O(n^{\rho})$ independent trees to achieve constant probability (say 90%).

The paper uses combinatoric proofs to demonstrate that inequality $(*)$ is satisfied for $\rho \approx \frac{1}{ln(4)\cdot c}$ for Hamming space when the Bit Partitioning based LSH Forest algorithm from section 3.4.

In section 4.2 we use proof by induction to formally prove the main theorem of the paper. In section 4.3 we present a combinatoric lemma necessary for the analysis, and in section 4.4. prove that the algorithm satisfies inequality $(*)$. In section 4.5 we conclude the results by showing that the algorithm achieves the desired ρ value, query time and space usage.

4.2 Main theorem and Proof by induction

Theorem 3: *The main theorem of [ARS17] is: for dataset P of n points, query point $q \in X$ and $p^* \in P$ is a near neighbor of q such that $D(p^*, q) \leq r$. Let $0 < \rho \leq 1$ be the smallest real number such that the equation holds for every subset $S \subseteq P$ where $p^* \in S$.*

- *either q is within distance cr from a pivot* BuildForest *computes for S.*
- *or inequality $(*)$ is satisfied.*

then BuildTree *returns a near neighbor of q with probability at least $n^{-\rho}$. Therefore* BuildForest *which samples $O(n^{\rho})$ independent trees has constant success probability (of say .9).*

Proof. We will prove the theorem using a proof by induction. If any pivot point is a near neighbor of q then we are successful with probability 1. Likewise in the **base case** the theorem holds as when $|S| = 1$ the success probability is 1 since the set of pivots is non-empty and by assumption $p^* \in S$.

We therefore need to prove the **inductive step** of the theorem applies when inequality $(*)$ holds.

1. Assume inequality $(*)$ holds and the inductive hypothesis holds for $S' \subseteq P$ where $|S'| < |S|$

2. By definition:
$$Pr[\text{success}] = Pr_{\mathcal{R}}[\text{success for } S \cap \mathcal{R}(q)]$$

3. By the law of total probability:
$$\geq Pr_{\mathcal{R}}[\text{success for } S \cap \mathcal{R}(q), \mathcal{R}(p^*) = \mathcal{R}(q)]$$

4. By Bayes rule:
$$\geq Pr_{\mathcal{R}}[\mathcal{R}(p^*) = \mathcal{R}(q)] \cdot Pr_{\mathcal{R}}[\text{success for } S \cap \mathcal{R}(q) | \mathcal{R}(p^*) = \mathcal{R}(q)]$$

5. By (1) the induction assumption for $|S'| < |S|$ we have:
$$\geq Pr_{\mathcal{R}}[\mathcal{R}(p^*) = \mathcal{R}(q)] \cdot \mathbb{E}_{\mathcal{R}}\left[|S \cap \mathcal{R}(q)|^{-\rho} \middle| \mathcal{R}(q) = \mathcal{R}(p^*)\right]$$

6. By linearity of expectation:
$$= Pr_{\mathcal{R}}[\mathcal{R}(p^*) = \mathcal{R}(q)] \cdot \mathbb{E}_{\mathcal{R}}\left[\left(\frac{|S \cap \mathcal{R}(q)|}{|S|}\right)^{-\rho} \middle| \mathcal{R}(q) = \mathcal{R}(p^*)\right] \cdot |S|^{-\rho}$$

7. By (1) that inequality $(*)$ holds we then have:

$$\geq |S|^{-\rho}$$

Applying the theorem to the root of $S = P$ gives the desired success probability for a single LSH tree as $|P|^{-\rho} = n^{-\rho}$. The algorithm from 3.4 therefore has constant success probability and solves $(c, r)-$ANN with probability at least .9. □

4.3 Combinatoric Lemma

The authors of [ARS17] use a number of combinatoric lemmas during the analysis. We condense them and present (without proof) the following:
Combinatoric Lemma: Let $x \in \{0,1\}^d, s \in \mathbb{N}$ and let $\mathcal{U}$ be a family of k subsets of coordinates $[d]$ such that:

- *For each $U \in \mathcal{U}, |U| \geq s$*
- *For each $U \in \mathcal{U}$, we have $\sum_{i \in U} x_i \geq \frac{|U|}{2} \cdot \left(1 - O\left(\frac{1}{c}\right)\right)$*
- *For every distinct pair $U_1, U_2 \in \mathcal{U}$, we have $|U_1 \setminus U_2| \geq s$*
- *$\sum_{i \in [d]} x_i \geq s$*

Then:

$$\prod_{i \in [d]} (1 - x_i) \leq exp\left(-s \cdot \frac{ln(4)k + 1}{k + 1} + O\left(s/c\right)\right)$$

4.4 Proof that the algorithm satisfies inequality $(*)$

In the main theorem, if q is within distance cr from any pivot point then the algorithm for a single LSH tree successfully finds the near neighbor with probability 1. We therefore need to prove that the algorithm satisfies inequality $(*)$ in the case when $D(p, q) > cr$ for all pivot points p.

For $i \in [d]$ let w_i denote the fraction of points $p \in S \subseteq P$ for which $p_i \neq q_i$. As we use the Bit partitioning LSH we then have:

$$\frac{|S \cap \mathcal{R}_i(p^*)|}{|S|} = 1 - w_i$$

where $\mathcal{R}_i$ denotes the partition corresponding to splitting the space according to bit i. We therefore have:

$$\mathbb{E}_{\mathcal{R}}\left[\left(\frac{|S \cap \mathcal{R}(q)|}{|S|}\right)^{-\rho} \middle| \mathcal{R}(q) = \mathcal{R}(p^*)\right] = \mathbb{E}_{i \in [d]}\left[(1 - w_i)^{-\rho} \middle| \mathcal{R}(q) = \mathcal{R}(p^*)\right]$$

We now use the fact that all pivot points are far from q and the Combinatoric Lemma to bound above expression.

Intuition: Without any further information, we would have that on average each w_i is $\frac{cr}{d}$ as the points in S are at distance cr far from q. As each pivot point is selected in order of closeness to the mean of S and sufficiently far from each other, we have that the mass is more highly concentrated.

Consider a case of a single pivot (i.e. k=1). Without loss of generality let $p^* = 0^d$. As in our case each pivot point is "far" from the query, and p^* is at distance at most r, we have that $||u - p^*||_1 > cr - r \approx cr$. Without loss of generality we can set $u = 1^{cr}0^{d-cr}$. Since u is closer to the mean than p^*, we have that at least for the cr non-zero coordinates of u that $w_i \geq \frac{1}{2}$ (which is much improved over $\frac{cr}{d}$).

The usage of k pivots at a distance $\approx cr$ from each other gives us an increased number of large weights. Therefore if k is large, the mass of w_i's is almost entirely concentrated on $\approx 2cr$ coordinates. This corresponds to a random instance for which (classic) Bit Sampling LSH yields $\rho \approx \frac{0.73}{c}$. Our algorithm can therefore be conceived as a reduction from the worse-case generic dataset to a random instance.

Let $\{p^1, \cdots, p^k\} \subseteq S$ be the set of k pivot points that are cr far from q. As above let p^c denote the mean of S. We then have:

Initial constraints:

1. Each pivot is closer to the mean of S than p^*. Formally: for each $i \in [k]$, we have $||p^i - p^c||_1 \leq ||p^* - p^c||_1$.
2. Each pair of pivots is $(c-1)r$ distant (from the selection of the pivots). Formally: for every distinct $i, j \in [k] : ||p^i - p^j||_1 \geq (c-1)r$.

Updated constraints:
For $i \in [k]$, let $U_i \subseteq [d]$ denote the set of coordinates on which p^i differ from p^*.
For each $j \in U_i$ we have: $p^*_j - p^c_j = w_j$ and $p^i_j - p^c_j = 1 - w_j$. Plugging to the initial constraints and summing over each U_i gives us:

1. For each $U_i : \sum_{j \in U_i} w_j \geq \sum_{j \in U_i} 1 - w_j$ such that $\sum_{j \in U_i} w_j \geq |U_i|/2$.
2. For every distinct $U_i, U_j : |U_i \setminus U_j| \geq (c-1)r$

Final constraints:
We now condition on the fact that q and p^* collide in the same bucket. Let $i \in I \subseteq [d]$ denote the set of coordinates for which $p^*_i = q_i$. We restrict ourselves to the set I and update the constraints to obtain:

1. For each $U_i : \sum_{j \in U_i \cap I} w_j \geq \frac{|U_i \cap I| - r}{2} = \frac{|U_i \cap I|}{2} \cdot \left(1 - O\left(\frac{1}{c}\right)\right)$
2. For every distinct $U_i, U_j : |(U_i \setminus U_j) \cap I| \geq (c-1)r - r = (c-2)r$

We introduce a new constraint:

3. For each $U_i : |U_i - I| \geq (c-1)r$ (since p^i is at least cr distant from q).

We can now finally use the Combinatoric Lemma with vector $x = w$ over bits in I and $s = (c-2)r$. We obtain:

$$(**) \prod_{i \in [d]} (1 - w_i) \leq exp\left(-(c-2)r \cdot \frac{ln(4)k + 1}{k+1} + O(r)\right)$$

We can now put all the pieces together for the final proof:

Proof. 1. As we sample uniformly we have:

$$\mathbb{E}_{\mathcal{R}}\left[\left(\frac{|S\cap\mathcal{R}(q)|}{|S|}\right)^{-\rho}\middle|\mathcal{R}(q)=\mathcal{R}(p^*)\right]=\mathbb{E}_{\mathcal{R}}\left[\left(\frac{|S\cap\mathcal{R}(p^*)|}{|S|}\right)^{-\rho}\middle|\mathcal{R}(q)=\mathcal{R}(p^*)\right]$$

$$=\mathbb{E}_{i\in I}\left[(1-w_i)^{-\rho}\right]$$

By the AM-GM inequality:

$$\geq\left(\prod_{i\in I}(1-w_i)^{-\rho}\right)^{1/|I|}$$

$$=\left(\prod_{i\in I}(1-w_i)\right)^{-\rho/|I|}$$

Plugging in $(**)$:

$$\geq exp\left(\frac{(c-2)r\cdot\rho\cdot(ln(4)k+1)}{|I|\cdot(k+1)}-O\left(\frac{\rho\cdot r}{|I|}\right)\right)$$

Setting $\rho=\frac{k+1}{(ln(4)k+1)\cdot(c-2)}+O\left(\frac{1}{c^2}\right)$, gives us:

$$\geq e^{r/|I|}$$

2. The probability that p^* and q collide is given by:

$$Pr_{\mathcal{R}}\left[\mathcal{R}(p^*)=\mathcal{R}(q)\right]\geq\frac{|I|}{|I|+r}$$

$$\geq e^{-r/|I|}$$

3. Multiplying (1) and (2) gives us:

$$Pr_{\mathcal{R}}\left[\mathcal{R}(q)=\mathcal{R}(p^*)\right]\cdot\mathbb{E}_{\mathcal{R}}\left[\left(\frac{|S\cap\mathcal{R}(q)|}{|S|}\right)^{-\rho}\middle|\mathcal{R}(q)=\mathcal{R}(p^*)\right]\geq e^{r/|I|}\cdot e^{-r/|I|}$$

$$=1$$

such that BuildTree satisfies inequality $(*)$. Per *Theorem 3*, the overall BuildForest algorithm from section 3.4 is therefore correct with constant probability.

□

4.5 Quality, Query time and Space usage:

As $\rho=\frac{k+1}{(ln(4)k+1)\cdot(c-2)}+O\left(\frac{1}{c^2}\right)<\frac{1}{ln(4)\cdot(c-2)}\cdot\left(1+\frac{2}{7k+5}\right)+O\left(\frac{1}{c^2}\right)$, we have: $\rho\approx\frac{1}{ln(4)\cdot c}$ as desired.

The overall BuildForest algorithm from section 3.4 samples $O(n^\rho)$ trees with each node storing k pivot points. The algorithm therefore has space usage $O(n^{1+\rho}k+nd)$. As the algorithm on receiving a query checks all pivots in all trees, the query time is $O(n^\rho\cdot d^2\cdot k)$.

5 Extensions and Discussion

A goal for ANN research is to determine whether an algorithm can adapt optimally to the structure of the dataset, while still maintaining worse-case guarantees. [AB22] makes progress towards this goal by extending the algorithm from [ARS17] to obtain the same worst-case guarantees but also adapts to the dataset to obtain success probability if the dataset does exhibit "nice" structure.

In particular, [AB22] modifies BuildTree to optimize for the best possible distribution over the random hash function for partitioning, instead of the uniform distribution used in [ARS17]. The ANN problem is recast as a two-player zero-sum game between the "hash player" and "query player" for Hamming space [AB22]. Results from game theory allow an approximately optimal equilibrium to be practically achieved. The authors demonstrate that in certain conditions the algorithm of [AB22] outperforms the uniform distribution used in [ARS17]. The results of [AB22] are limited; however, in that there is no improvement in ρ value for worse-case datasets.

Discussion:
[AB22] demonstrates a potential proof strategy for possible improvement to [ARS17] by finding algorithms that satisfy inequality $(*)$.

Both [ARS17] and [AR15] essentially perform reductions from the worst-case dataset (i.e. any generic dataset) to a random dataset for Hamming and Euclidean spaces respectively and exploit the good performance of classic LSH on random datasets. Further progress in this direction would require improvement to the random dataset case which is a formidable challenge that seemingly would require new techniques [AR15].

It remains an open question whether a simple algorithm akin to [ARS17] can achieve (or nearly so) the optimal ρ value established by [AR15].

Finally, [ARS17] is an example of taking a popular heuristic and proving a theoretical guarantee. While seemingly difficult, a potential direction for further improvement would be finding theoretical guarantees for other ANN heuristics that empirically perform well.

References

[AB22] Alexandr Andoni and Daniel Beaglehole Learning to Hash Robustly, Guaranteed. arXiv 2108.05433, 2022

[AIR18] Alexandr Andoni, Piotr Indyk and Ilya Razenshteyn. Approximate Nearest Neighbor Search in High Dimensions. arXiv 1806.09823, 2018.

[AR15] Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In Proceedings of the Symposium on Theory of Computing (STOC), 2015.

[ARS17] Alexandr Andoni, Ilya Razenshteyn, and Negev Shekel Nosatzki. Lsh forest: Practical algorithms made theoretical. In Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA), 2017.

[HIM12] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. Theory of computing, 8(1):321–350, 2012.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In Proceedings of the thirtieth annual ACM symposium on Theory of computing, pages 604–613. ACM, 1998.