CS294-165 Final Project Instructor: Jelani Nelson

Approximate Distance Estimation

Emaan Hariri, Nagaganesh Jaladanki, Noah Kingdon December 2020

Abstract

In this survey, we examine various works related to the approximate distance estimation problem (ADE) and its related variants as well as the applications of sketching to the problem. We first discuss the bounds on the asymptotic space complexity of the problem as well as the derivation and implications of such bounds as mentioned in [AK17]. We also discuss from [IW17] a metric embedding technique that shows how space complexity for distance estimation can go below the lower bounds established in for [JL84] outside the dimensionality reduction regime. Lastly, we explore ADE in the context of *adaptive* or *adversarial* queries as discussed in [CN20], which has become relevant in the broader context of machine learning and other data analysis applications.

1 Introduction

The problem of *approximate distance estimation* (ADE) is a fundamental problem related to many other algorithms in data analysis, such as nearest neighbors search. The problem, as we consider it, is defined as follows:

Problem 1. *For a known norm $||\cdot||$, we are given a set of vectors $X = \{x_i\}_{i=1}^n \subset \mathbb{R}^d$ and an accuracy parameter $\varepsilon \in (0,1)$ and must produce data structure $\mathcal{D}$. Later, given only $\mathcal{D}$ stored in memory with no direct access to X, we must respond to queries that specify $q \in \mathbb{R}^d$ by reporting distance estimates $\widetilde{d}_1, \ldots, \widetilde{d}_n$, satisfying:*

$$\forall i \in [n], (1-\varepsilon)\|q - x_i\| \leq \widetilde{d}_i \leq (1+\varepsilon)\|q - x_i\|$$

The seminal work [JL84] introduced the well-known algorithm for approximate distance estimation which formally states:

Theorem 1.1. *For any $\varepsilon \in (0,1)$ and any $X \subset \mathbb{R}^d$ for $|X| = n$ finite, there exists an embedding $f : X \to \mathbb{R}^m$ for $m = O(\varepsilon^{-2}\log n)$ such that:*

$$\forall x, y \in X, (1-\varepsilon)||x-y||_2^2 \leq ||f(x)-f(y)||_2^2 \leq (1+\varepsilon)||x-y||_2^2$$

The best-known proof of this theorem projects the points of X onto a random subspace of small dimension and utilizes concentration of measure to show that the squared Euclidean distances between the points have not changed by a relative error greater than ε.

The original statement of [JL84] was shown without an optimality statement; it was shown to be optimal only over 30 years later in [LN17] for dimensionality reduction embeddings. Further work in this field since has delved into proving bounds, reducing space complexity, or additional robustness in an adversarial setting. In this survey, we look through additional work in approximate distance estimation since [JL84] in these various settings.

2 Bound on Space Complexity of ε-distance Sketch

We now consider the lower bound on a ε-distance sketch as presented by Alon and Klartag in [AK17], where we are concerned with an additive rather than a relative error. A ε-distance sketch is a data structure $\mathcal{D}$ such that given $X \subset \mathbb{R}^k$ with $|X| = n$ and $\forall x \in X$ $\|x\| \leq 1$, forall $x, y \in X$ we can provide an estimate $\widehat{\|x-y\|}_2^2$ such that

$$\left| \widehat{\|x-y\|}_2^2 - \|x-y\|_2^2 \right| \leq \varepsilon$$

In other words, all distances can be estimated up to an *additive* error of ε. For $n \geq k \geq 1$ and $\varepsilon \geq 1/n^{0.49}$, [AK17] provides a lower bound on $f(n, k, \varepsilon)$, the minimum number of bits of $\mathcal{D}$. They find that when $1/n^{0.49} \leq \varepsilon \leq 0.1$ we have

$$f(n,k,\varepsilon) = \begin{cases} \Theta\left(\frac{n \log n}{\varepsilon^2}\right) & \frac{\log n}{\varepsilon^2} \leq k \leq n \\ \Theta\left(nk \log\left(2 + \frac{\log n}{\varepsilon^2 k}\right)\right) & \log n \leq k \leq \frac{\log n}{\varepsilon^2} \\ \Theta\left(nk \log\left(\frac{1}{\varepsilon}\right)\right) & 1 \leq k \leq \log n \end{cases}.$$

In [KOR00] we are provided a $O(\frac{n \log n}{\varepsilon^2})$ sketch for the closely related approximate nearted neighbor search in the unit ball, which solves our above problem, which we can see by the above is the optimal such sketch for $k \geq \log n / \varepsilon^2$.

We take $t = \frac{\log(2+\varepsilon^2 n)}{\varepsilon^2}$ for $\varepsilon \geq \frac{2}{n}$ and $\ell \leq c \frac{\log n}{\varepsilon^2}$ for some small positive constant c. We have that for $k \leq t$ there exists some $S \subset \mathbb{R}^k$, $|S| = n$, $\forall s \in S$, $\|s\| \leq 1$ such that the embedding of S into $\mathbb{R}^\ell$ distorts the distance between some pair of points by more than ε. This extends the proof of optimality of the Johnson-Lindenstrauss Lemma from [LN17] to the regime $\varepsilon \in (2/\sqrt{n}, \log^{0.5001} n/\sqrt{k})$.

As noted by [LN17] the derivation of $f(n, k, \varepsilon)$ itself provides an optimality proof for the JL Lemma. We see that $f(n, n, 2\varepsilon) \gg f(n, \ell, \varepsilon)$ for $\ell = c \frac{\log n}{\varepsilon^2}$ for small positive constant $c > 0$. However, if we wish to reduce to dimension m, we require that $f(n, n, 2\varepsilon) \leq f(n, m, \varepsilon)$. Thus we find that our bound of $m = \Omega(\varepsilon^{-1} \log n)$ is tight (note that there is increasing monotonicity of $f(n, k, \varepsilon)$ in k).

Additionally, [AK17] provides an improvement to the JL Lemma that allows reduction to dimension $O(t)$ in the *bipartite variant* of the JL Lemma which is as follows.

Theorem 2.1. *For any $2n$ vectors $a_1, \ldots, a_n, b_1, \ldots, b_n \in \mathbb{R}^n$, $\forall i \in [n]$, $\|a_i\| \leq 1$ and $\|b_i\| \leq 1$, we can find $x_1, \ldots, x_n, y_1, \ldots, y_n \in \mathbb{R}^t$ for $t = O(\varepsilon^{-2} \log(2 + \varepsilon^2 n))$ such that $\forall i, j \in [n]$*

$$|\langle x_i, y_j \rangle - \langle a_i, b_j \rangle| \leq \varepsilon.$$

2.1 Gram Matrix Argument

We present the argument made by [AK17] that uses Gram matrices to produce the appropriate upper bounds of $f(n, k, \varepsilon)$. The Gram matrix for vectors $w_1, \ldots, w_n$, or $G(w_1, \ldots, w_n) \in \mathbb{R}^{n \times n}$, is defined by $G_{ij} = \langle w_i, w_j \rangle$. Two Gram matrices G and G' are ε-separated if $\exists i \neq j$ such that $|G_{ij} - G'_{ij}| > \varepsilon$.

We consider the set $\mathcal{G}$, which is the maximal set of ε-separated Gram matrices of ordered sets of n vectors $w_1, \ldots, w_n \in \mathbb{R}^m$ such that $\|w_i\| \leq 2$ (we choose 2 instead of 1 because if we wish to apply the JL Lemma our original vector norms will be slightly distorted). Observe that since $\mathcal{G}$ is *maximal*, for any Gram matrix M of vectors $v_1, \ldots, v_n \in \mathbb{R}^m$, $\|v_i\| \leq 2$ there must be an element $G^* \in \mathcal{G}$ such that $\forall i \neq j \in [n]$, $|M_{ij} - G^*_{ij}| \leq \varepsilon$ (otherwise $\mathcal{G}$ would not be maximal). Thus, it suffices to store the index of G^* which require $\log |\mathcal{G}|$ bits.

In effect, we have formed an ε-net across all relevant $n \times n$ Gram matrices (under an entrywise norm), and are concerned with point in the net which is ε-close to the Gram matrix of our point set $G(w_1, \ldots w_n)$. The proof bounding $f(n,k,\varepsilon)$ is completed by now upper bounding the cardinality of $\mathcal{G}$ (which can done via a probabilistic argument).

2.2 Algorithmic Proof Sketch

We further expound on the algorithmic proof of theorem of the upper bound of $f(n,k,\varepsilon)$ as presented in [AK17] which presents an algorithm for the explicit construction of a sketch of size $O(f(n,k,\varepsilon)/n)$ per vector; specifically, we look at the sketch which allows for the recovery of approximate inner products between vectors.

In the case of $k \leq \log n$, it suffices to use an $\varepsilon/2$-net of size $(1/\varepsilon)^{O(k)}$ for the unit ball, and round each vector to one that is ε-close. We concern ourselves with $k \geq \frac{40 \log n}{\varepsilon^2}$, where by [JL84] we project our vectors onto $\mathbb{R}^m$ for $m \geq \frac{40 \log n}{\varepsilon^2}$ and scale as appropriate so that each vector lies on the unit ball in $\mathbb{R}^m$ and norms and squared distances will be distorted by less than 3ε.

The sketch now involves taking the rounded vector V_i in place of our original vectors w_i. The vectors V_i are constructed by taking each coordinate of w_i and independently and rounding randomly to one of its nearest integral multiples of $\frac{1}{\sqrt{m}}$. In other words, the j^{th} coordinate of V_i is

$$V_i(j) = \begin{cases} \frac{\lceil w_i(j) \cdot \sqrt{m} \rceil}{\sqrt{m}} & \text{w.p. } (w_i(j) \cdot \sqrt{m} - \lfloor w_i(j) \cdot \sqrt{m} \rfloor) \\ \frac{\lfloor w_i(j) \cdot \sqrt{m} \rfloor}{\sqrt{m}} & \text{w.p. } (\lceil w_i(j) \cdot \sqrt{m} \rceil - w_i(j) \cdot \sqrt{m}) \end{cases}$$

where the probabilities are chosen such that $\mathbb{E}[V_i(j)] = w_i(j)$ and so $\mathbb{E}[V_i - w_i] = \vec{0}$.

We now consider the inner product $\langle V_i - w_i, w_j \rangle$ and observe that we can represent this as $\langle V_i - w_i, w_j \rangle = \sum_{l=1}^k X_l$ for some random variable $X_l \in [a_l, b_l]$ such that $\mathbb{E}[X_l] = 0$ and $\sum_{l=1}^k (b_l - a_l)^2 \leq \|w_j\|^2/m$. Since $\|w_j\| \leq 1$, we know $\sum_{l=1}^k (b_l - a_l)^2 \leq 1/m$ and by Hoeffding's inequality [Hoe63] we have that $\mathbb{P}(|\langle V_i - w_i, w_j \rangle| \geq \varepsilon/2) \leq 2\mathrm{e}^{-\varepsilon^2 m/8} \leq 1/n^5$ for our values of ε. We follow similar reasoning for $\langle V_i, V_j - w_j \rangle$ and since we have that $|\langle V_i, V_j \rangle - \langle w_i, w_j \rangle| \leq |\langle V_i, V_j - w_j \rangle| + |\langle V_i - w_i, w_j \rangle|$ taking a union bound over $\forall i \neq j$ we have that

$$\mathbb{P}\left(\forall i \neq j \in [n] : |\langle V_i, V_j \rangle - \langle w_i, w_j \rangle| \leq \varepsilon\right) \geq 1 - \frac{2}{n^3}$$

as desired. Separately, we have our sketch for approximate norms with $O(\log(1/\varepsilon))$ bits per vector which, in conjunction with the approximate inner product sketch, will allow us to approximate distances.

To complete the algorithm for the regime of $k = \frac{40\delta^2 \log n}{\varepsilon^2}$ for $\delta \in [\varepsilon, 1/2]$, we no longer perform a projection onto $\mathbb{R}^m$ and now round randomly in a similar fashion to one of the nearest integral multiples of $\delta/\sqrt{k}$ (so it is still the case $\mathbb{E}[V_i(j)] = w_i(j)$). The above provides an algorithm that can for $g(n,k,\varepsilon) = O(f(n,k,\varepsilon)/n)$ (across all values of k) bits per vector preserve the distances between vectors to within an additive error of ε, as desired.

3 Metric Compression

It is known that there is no better dimensionality reduction than that given in [JL84]. However, [IW17] gave an improved algorithm using metric compression with a lower space complexity.

Specifically, the metric approximate distance estimation is identical to the one defined in Problem 1 with the caveat that the points have ℓ_p pairwise distances in the range $[1, \Phi]$. Given a preprocessed sketch $\mathcal{D}$, the algorithm must output all pairwise distance between the input vectors with an accuracy parameter of ε.

In this setting, [JL84] requires $O(\varepsilon^{-2} \log n \log \Phi)$ bits per point using dimensionality reduction, but [IW17] shows a bound of $O(\varepsilon^{-2} \log n + \log\log \Phi)$ bits per point using a metric compression algorithm. Specifically, they have:

Theorem 3.1. *For n points on an ℓ_2 metric, approximating pairwise distances that lie in the range $[1, \Phi]$ has a tight bound of $\Theta(\varepsilon^{-2} n \log n + n \log\log \Phi)$ bits of space complexity.*

The given randomized sketching algorithm below has preprocessing time $O(n^{1+\alpha}) \log \Phi + nd \log d + \varepsilon^{-2} n \log^3 n)$ where d is the dimension of the input metric and α is an arbitrarily small constant. The estimation algorithm runs in time $O(\varepsilon^{-2} \log n \log(\varepsilon^{-1} \Phi \log n))$.

3.1 Technical Overview

Intuitively, the overview of the algorithm is as follows: we define a "surrogate" point $s^*(x) \in \mathbb{R}^d$ for all points $x \in X$. We approximate the pairwise distance $||x_i - x_j||$ between points as $||s^*(x_i) - s^*(x_j)||$ instead.

The strategy to of the surrogate points is to store a point with respect to its relative location to a nearby point. Specifically, we choose an ingress point $ins(x) \in X$ near x and define the surrogate $s^*(x)$ inductively as:

$$s^*(x) = s^*(ins(x)) + [x - s^*(ins(x))]_\gamma$$

where we use $[y]_\gamma$ to imply rounding to a γ-net with appropriate precision to guarantee our relative error ε. One way to think of this estimation is that we are "snapping" all points onto a grid of pixels that we then store the relationships between.

Instead of storing approximations for individual points, we construct a hierarchical clustering tree. We define "surrogate" and "ingress" points along for all elements along the tree to aid in reconstructing the distance between points. This tree clumps together nearby points into a "center" point that is sufficiently close such that the distances from points that are further away are not greatly affected. Then, returning the distance between computed surrogates is enough to satisfy the relative error of ε.

The intuition above gives an algorithm for ℓ_p metrics, but fails on ℓ_2 metric for technical reasons. The solution to this is to to use randomized rounding to a grid as a Ω-net instead of the γ-net we used earlier for the ℓ_p metric. This paper will skip over this and focus on a presentation of the hierarchical clustering tree instead.

3.2 Grid Nets

The best known net are γ-nets, which are formed by a collection of $(\frac{c}{\gamma})^d$ points along the p-norm unit ball $\mathcal{B}_p^d \subset \mathbb{R}^d$ in dimension d such that for all $x \in \mathcal{B}_p^d$, x is no less than γ distance away from a point in the γ-net.

We use a specific construction involves the intersection of a ball with a scaled grid. Specifically, let $\mathcal{G}^d[p] \subset \mathbb{R}^d$ denote the d-dimensional grid with cell side length ρ. For this construction, we use the net $\mathcal{N}_\gamma = 2 \cdot \mathcal{B}_p^d \cap \mathcal{G}^d[\gamma/d^{1/p}]$, where $2 \cdot \mathcal{B}_p^d$ is just the p-norm ball of radius 2.

It is easily verified that $\mathcal{N}_\gamma$ is indeed a γ-net with an optimal cardinality $(c/\gamma)^d$. This is since given some $x \in \mathcal{B}_p^d$, we can find a $y \in \mathcal{N}_\gamma$ that is less than γ distance away by dividing each coordinate by $\gamma/d^{1/p}$, rounding it to the smallest integer, and multiplying it by $\gamma/d^{1/p}$. The output of this procedure must output a value in $\mathcal{N}_\gamma$ since the output has all coordinates an integer multiple of $\gamma/d^{1/p}$, which must lie on our grid as specified before. Note that this procedure takes time $O(d)$ and storage $O(d\log(1/\gamma))$.

3.3 The Relative Location Tree

This paper introduces the *relative location tree* data structure. We outline the construction of this data structure before going into an explanation of the annotations of this tree corresponding to the surrogate and ingress points.

3.3.1 Hierarchial Clustering Tree Construction

We construct a hierarchical clustering tree T^* level-by-level over our input points X as follows. The bottom level (level 0) starts off with leaves that is just a singleton cluster $\{x_i\}$. We construct level $l > 0$ from the sets in level $l-1$ by merging together clusters that are less than 2^l distance apart. Notice that by level Φ, the pointset will have been merged into a single complete set.

We denote $C(v) \subset X$ to be the cluster associated with a particular tree node $v \in T^*$ and $\Delta(v)$ to be its cluster diameter. With this construction, we have a partition for the input points X at every level based on distance. This construction takes $O(n^2 \log \Phi)$ time. Because of the cluster construction, we have the following:

$$||x_i - x_j|| \geq 2^l$$

As an optimization, we introduce "path compression" in order to compress long paths in T^* that do not have branches. Essentially, for a downward path $v_0, v_1, ..., v_k$ where $v_1, ..., v_k$ are degree 1 tree nodes, we can compress all the degree 1 nodes into a single node, removing the excess nodes to save space if the path is long enough. Specifically, we replace a path if its length $k > \log(2^{-\ell(v_k)}\Delta(v_k)/\varepsilon)$. Call the path compressed tree T, and note that this path compression step requires $O(n \log \Phi)$ time, since it is linear in the size of the tree.

3.3.2 Tree Annotations: Centers, Ingresses, and Surrogates

Next, we annotate each tree node with a center, ingress, and surrogate. The idea is that when computing the norm between two given points, we can traverse T and compute a representation of the distance between points through these additive distance separations.

Center: Each tree node v corresponds to a subset of original given points, $A_v \subset X$. Set the center point to be the point with the smallest index in A_v:

$$center(v) = x_{\min_{i:x_i \in A_v} i}$$

Ingresses: Intuitively, the ingress point for a tree node is a point in the subtree rooted at that tree node that is close to the center of the tree. The purpose is to store the center by its location relative to the ingress point.

For a point in our tree $v \in T$, let $u_1, ..., u_k$ be the children of v. Then, construct the graph H with vertex set $u_1, ..., u_k$ and edge set $\{(u_i, u_j) : dist(C(u_i), C(u_j)) \le 2^{\ell(v)}\}$, where $\ell(v)$ is the level of node v. We know that H must be connected because we connected the node v at level $\ell(v)$ because the nodes were within $2^{\ell(v)}$ distance away.

Set the ingress of each node u_i to be the node in its parent's cluster (on the spanning tree) for which it is the closest do. Given this, we have that the distance between a point u and its ingress point $in(u)$ will be at most $3 \cdot 2^{\ell(u)} + \Delta(u)$.

Finally, we place two "rounding" values to each node: $\gamma(v)$ and $\eta^*(v)$ defined as follows.

$$\gamma(v) = \left(5 + \lceil \frac{\Delta(v)}{2^{\ell(v)}} \rceil \right)^{-1}$$

$$\eta^*(v) = \frac{\gamma(v)}{2^{\ell(v)}} \cdot (center(x) - s^*(in(x)))$$

We also define surrogates, but we do not include them in our sketch – we only place them here for the purpose of using them later. Define the surrogate of a node v to be $s^*(v)$ as being inductive of the ingress of v, giving:

$$s^*(v) = s^*(in(v)) + \frac{2^{\ell(v)}}{\gamma(v)} \cdot \eta(v)$$

Note that $\eta(v)$ is just $\eta^*(v)$ but rounded to the closest value on the net $\mathcal{N}_{\gamma(v)}$. We define leaf surrogates to be the same, but rounded to the closest point on the $\mathcal{N}_{\gamma(v) \cdot \varepsilon}$ net as defined earlier.

Claim 3.2. *For every $v \in T$, we have: $||center(v) - s^*(v)|| \le 2^{\ell(v)}$*

A similar corollary holds for leaves, except the RHS is $2^{\ell(v)} \cdot \varepsilon$.

3.3.3 Overall tree

The overall data structure consists of the actual tree T (including path-compressed long paths) including some auxiliary information. For each node, we store: the point $center(v)$, the ingress $in(v)$, the precision $\gamma(v)$, and the element $\eta(v)$ (belonging to the original grid net). We show a series of lemmas that bounds the size of the total data structure.

Lemma 3.3. *The number of nodes in the tree T is $O(n \log(1/\varepsilon))$.*

Proof. Now, remember that path compression removes nodes from the tree if the path is above a certain length. So, we can iterative over the nodes left that have degree $deg(v) > 1$ and upper bound the length of the 1-path they hav. If we iterate over those nodes and add up the number of nodes they replaced, we can get the number of actual nodes in the tree.

Specifically, we have that the number of nodes in the tree is $\sum_{v:deg(v)\neq 1} k(v)$ where $k(v)$ is the number of nodes in the path that was replaced. We know that $k(v) \leq \log(2^{-\ell(v)}\Delta(v)) + \log(1/\varepsilon)$ or it would have replaced in the path compression step. Combining this with the fact that there are $2n$ nodes in the tree and with the lemma proved in the Appendix A, we get that:

$$\begin{aligned}|T| &= \sum_{v:deg(v)\neq 1} k(v) \\ &= \sum_{v:deg(v)\neq 1} (\log(2^{-\ell(v)}\Delta(v)) + \log(1/\varepsilon)) \\ &= 2n(2 + \log(1+\varepsilon))\end{aligned}$$

□

Lemma 3.4. *The total sketch size is $O(n(d + \log n)\log(1/\varepsilon) + n \log\log\Phi)$.*

Proof. The size of the tree is $O(n\log(1/\varepsilon))$, so we need those many bits to encode the structure of the tree. We know that there are $\log\Phi$ levels in the tree, since the max distance between elements is Φ. Representing this level will take $\log\log\Phi$ bits for each node, so it will take $O(n\log\log\Phi)$ bits to represent the depth for all nodes.

We store the center and ingress for each node, which is just a label to another node, which takes $\log n$ bits times the number of nodes in the tree, for a total of $O(|T|\log n)$ bits.

Finally, each node also contains $\eta(v)$, which belongs to the original grid net. Each element requires $O(d\log(1/\gamma(v)))$.

Putting this all together, we get that the total size of the sketch it $O(n(d + \log n)\log(1/\varepsilon) + n\log\log\Phi)$. □

3.4 Distance Estimation

With the preprocessing stage complete, the algorithm can use the relative location tree in order to approximate $||x_i - x_j||$ for all x_i, x_j. The main intuition is that we use the tree to recover the surrogates of the points, which will contain an upper bound in how far they deviate from the actual point pair.

Remember that we defined surrogates earlier but did not store them with the rest of the tree. We aim to recreate the value of the surrogate using the information stored with the tree during the distance estimation. Define the shifted leaf surrogate the same way as we defined the original surrogate, which we can do since we have labels for the ingress node, $\gamma(v)$, and $\eta(v)$. Denote the shifted surrogate as $s(v)$. We then have that

$$s(v) = s^*(v) - center(x)$$

This makes sense since we cannot store the full center point for a node (if we could do this, we might as well store every point in its full description anyway).

Finally, to find the approximate distance between a given $x_i, x_j \in X$. Let u_{ij} be the lowest common ancestor of x_i and x_j. Let v_i be the direct child of u_{ij} whose subtree contains x_i, and similarly for x_j. Given this, we have the approximation as:

Lemma 3.5. $||s_\varepsilon(v_i) - s_\varepsilon(v_j)|| = (1 \pm 4\varepsilon) \cdot ||x_i - x_j||$

We know that $||s_\varepsilon(v_i) - s_\varepsilon(v_j)|| = ||s^*_\varepsilon(v_i) - s^*_\varepsilon(v_j)||$, so we lose no accuracy there even though we did not store the center point. Using the triangle inequality, we have:

$$||s^*_\varepsilon(v_i) - s^*_\varepsilon(v_j)|| = ||x_i - x_j|| \pm (||x_i - s^*_\varepsilon(v_i)|| + ||x_j - s^*_\varepsilon(v_j)||)$$

The intuition is using the bounds developed earlier to upper bound the deviation by a factor of 4ε.

4 Adaptive Queries

One setting in which previous methods for ADE can fail is that of *adaptive* queries. Suppose our data structure processes a sequence of queries, each of which may depend on the output of the queries that came before. We explore how to handle queries in this context.

4.1 Generalized Adaptive Context

4.1.1 Adaptive versus Non-adaptive guarantees

Generalizing from the context of ADE, suppose we have a space of queries $\mathcal{Q}$ that we wish to answer using a data structure $\mathcal{D}$. In the ordinary non-adaptive setting, we wish to guarantee that

$$\forall q \in \mathcal{Q} : \ \mathbb{P}_{\text{pre}} (\mathcal{D} \text{ answers } q \text{ correctly}) > 1 - \delta \tag{4.1}$$

where the probability is taken over the randomness in constructing the data structure, i.e. the pre-processing. This means that for each fixed query, we can guarantee with high probability that the data structure we construct correctly answers that query. However, it is still possible that for every potential data structure we construct, there is some query it always answers incorrectly. An adversarial sequence of queries could potentially use the outputs of previous queries to find this pathological query, reliably causing the data structure to fail. To protect against this, we would hope to provide the following modified guarantee

$$\mathbb{P}_{\text{pre}} (\forall q \in \mathcal{Q} : \ \mathbb{P}_{\text{query}} (\mathcal{D} \text{ answers } q \text{ correctly}) > 1 - \delta) > 1 - \delta \tag{4.2}$$

where the outer probability is taken over the the pre-processing and the inner probability is taken over the query process. Having a randomized query process will actually set apart the adaptive approach we discuss, as all non-adaptive methods we've discussed have had deterministic query algorithms.

4.1.2 Constructing an Adaptive Sketch from a Non-adaptive Sketch

Cherapanamjeri and Nelson [CN20] present the following procedure for constructing a data structure to handle adaptive queries, given a data structure which handles non-adaptive queries. Suppose we have an algorithm to construct a data structure $\mathcal{D}$ which satisfies the non-adaptive guarantee (4.1). Furthermore, suppose that for some l, if we take l independent instances $\mathcal{D}_1, \dots, \mathcal{D}_l$ of $\mathcal{D}$, then with probability at least $1-\delta$, they satisfy the condition

$$\forall q \in \mathcal{Q} : \sum_{i=1}^{l} \mathbf{1}\{\mathcal{D}_i \text{ answers } q \text{ correctly}\} \geq 0.9l \tag{4.3}$$

Under this assumption, if we take a random sample of the l data structures, then in expectation, 90% of them will answer any given query correctly. Therefore, by the Chernoff bound, in order to answer an approximate numerical query correctly with probability at least $1-\delta$, it suffices to take a sample of the l data structures of size $\Theta\left(\log \frac{1}{\delta}\right)$ and report the median. This provides a data structure which satisfies (4.2), and hence correctly answers adaptive queries with probability at least $1-2\delta$. When applying this technique to a specific query problem for which a non-adaptive sketch is known, the operative becomes finding such an l to satisfy the above condition (4.3).

4.2 Adaptive ADE for ℓ_p Norms

4.2.1 Overview

We now assume we're back in the context of Approximate Distance Estimation, where we're given a set of vectors $X \subset \mathbb{R}^d$, and we want to construct a data structure which can take in a query point $q \in \mathbb{R}^d$ and quickly provide approximate distances from q to every vector in X with respect to a specific norm. Applying resulting from [Ind06], we can construct a linear sketch which solves ADE for ℓ_p norms in the non-adaptive case by using a projection matrix $\Pi \in \mathbb{R}^{m \times d}$ whose entries are drawn i.i.d. from a p-stable distribution. Using the general technique from the previous section, if we take l copies of this linear sketch for some l satisfying guarantee (4.3), we can get a data structure which correctly answers adaptive ADE queries for ℓ_p norms. By bounding the Frobenius norm of Π and leveraging results from [Ind06] along with a net argument, [CN20] finds that it is sufficient to take

$$l = O\left(\left(d + \log \frac{1}{\delta}\right) \log \frac{d}{\varepsilon}\right)$$

copies of the non-adaptive data structure to achieve this guarantee.

4.2.2 Data Structure

As input for ADE on the ℓ_p norm, we take in the set of vectors $X = \{x_i \in \mathbb{R}^d\}_{i=1}^n$, an accuracy parameter $\varepsilon \in (0,1)$ and a failure probability $\delta \in (0,1)$. The non-adaptive data structure from [Ind06] stores $\{\Pi, \{\Pi x_i\}_{i=1}^n\}$ where $\Pi \in \mathbb{R}^{m \times d}$ is randomly chosen via $\Pi_{ij} \sim \mathrm{Stab}(p)$, with $m = O\left(1/\varepsilon^2\right)$. Therefore, the adaptive data structure for ℓ_p norm ADE, as described in [CN20], stores l copies of this, i.e.

$$\{\Pi_j, \{\Pi_j x_i\}_{i=1}^n\}_{j=1}^l$$

Each matrix $\Pi_j \in \mathbb{R}^{m \times d}$ takes up space $O(md)$, and each set of vectors $\{\Pi_j x_i\}_{i=1}^n$ takes up space $O(mn)$, so the space complexity of l copies of these pairs is given by

$$O(ml(d+n)) = O\left(\frac{1}{\varepsilon^2}(d+n)\left(d+\log\frac{1}{\delta}\right)\log\frac{d}{\varepsilon}\right)$$

which is slightly worse than the $O(nd)$ space it would take to store all of X explicitly.

Observe that computing the matrix vector products $\{\{\Pi_j x_i\}_{i=1}^n\}_{j=1}^l$ is equivalent to computing the single matrix product

$$\begin{bmatrix} \Pi_1 \\ \vdots \\ \Pi_l \end{bmatrix} \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix} = \begin{bmatrix} \Pi_1 x_1 & \cdots & \Pi_1 x_n \\ \vdots & \ddots & \vdots \\ \Pi_l x_1 & \cdots & \Pi_l x_n \end{bmatrix}$$

Therefore, if the runtime of multiplying an $a \times b$ matrix with a $b \times c$ matrix is $\mathrm{MM}(a,b,c)$, the pre-processing time to construct the data structure is dominated by

$$\mathrm{MM}(ml,d,n) = \mathrm{MM}\left(\frac{1}{\varepsilon^2}\left(d+\log\frac{1}{\delta}\right)\log\frac{d}{\varepsilon}, d, n\right).$$

4.2.3 Query Algorithm

As input for an ADE query, we're given a vector $q \in \mathbb{R}^d$. For each $i \in [n]$, the non-adaptive data structure from [Ind06] estimates the distance $\|q - x_i\|_p$ as $\frac{\mathrm{Median}(|\Pi q - \Pi x_i|)}{\mathrm{Med}_p}$, where $\mathrm{Med}_p = \mathrm{Median}(|Z|)$ for $Z \sim \mathrm{Stab}(p)$. For the adaptive data structure described in [CN20], we sample $j_1, \ldots, j_r$ with replacement from $[l]$, where $r = O(\log n/\delta)$. For each $i \in [n]$, we then report the median of responses for that i from the r sampled data structures. The result of our adaptive query is therefore

$$\left\{\mathrm{Median}\left(\left\{\frac{\mathrm{Median}\left(|\Pi_{j_k} q - \Pi_{j_k} x_i|\right)}{\mathrm{Med}_p}\right\}_{k=1}^r\right)\right\}_{i=1}^n.$$

For each query, we first compute $\Pi_{j_k} q$ for each $k \in [r]$, which takes $O(mdr)$ time. Computing the above quantities afterwords then takes $O(mrn)$ time, so the overall query time is given by

$$O(m(d+n)r) = O\left(\frac{1}{\varepsilon^2}(d+n)\log\frac{n}{\delta}\right).$$

References

[AK17] Noga Alon and Bo'az Klartag. Optimal compression of approximate inner products and dimension reduction. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 639–650. IEEE, 2017.

[CN20] Yeshwanth Cherapanamjeri and Jelani Nelson. On adaptive distance estimation, 2020.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[Ind06] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of the ACM (JACM)*, 53(3):307–323, 2006.

[IW17] Piotr Indyk and Tal Wagner. Near-optimal (euclidean) metric compression. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 710–723. SIAM, 2017.

[JL84] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

[KOR00] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal on Computing*, 30(2):457–474, 2000.

[LN17] Kasper Green Larsen and Jelani Nelson. Optimality of the johnson-lindenstrauss lemma. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 633–638. IEEE, 2017.

Appendix

Lemma 4.1. *For the clustering tree in [IW17], we have:*

$$\sum_{v \in T} \log(2^{-\ell(v)}\Delta(v)) \leq 4n$$

Proof. For an edge between parent node u and child node v, call uv to be a "1-edge" if $deg(u) = 1$. Set the weight of an edge to be 0 if it is a "1-edge", and $2^{\ell(v)}$ otherwise. Denote the weight of a vertex to be the weight of all edges in the subtree rooted at that vertex.

Now, we show that $\Delta(v) \leq wt(v)$, or that the diameter of a cluster at node v is less than the weight at that node. Use induction. At the base case, we have $\Delta(v) = wt(v) = 0$. There are two cases for the inductive step: if $deg(v) = 1$ or otherwise. In the first case, $wt(u) = wt(v)$ because the 1-edge does not change the total weight.

In the second case, we have that that the cluster at node v is partitioned among the different nodes that are children of v, which are $u_1, ..., u_k$. We have that the diameter $\Delta(v)$ is upper bounded by the sum of the diameters of the child clusters plus the distance between them, which gives us:

$$\Delta(v) \leq \sum_{i=1}^{k} \left(\Delta(u_i) + dist(C(u_i), C(v) \setminus C(u_i))\right)$$

We know from induction that $\Delta(u_i) \leq wt(u_i)$, and from how we constructed the tree that $dist(C(u_i), C(v) \setminus C(u_i)) \leq 2^{\ell(v)} = wt(vu_i)$.

Putting this together, we get:

$$\Delta(v) \leq \sum_{i=1}^{k}(wt(vu_i) + wt(u_i)) = wt(v)$$

Now, we show that $\sum_{v \in T} 2^{-\ell(v)} wt(v) \leq 4n$. First, note that we only need to add the contributions from non-1-edges (since 1-edges contribute 0 to the cost). Notice that a non-1-edge contributes its weight to all vertices along its path to the root, which we denote $u_1, ..., u_k$.

Then, the contributions of this non-1-edge will be $\sum_{i\geq 0} 2^{-\ell(u_i)} \cdot 2^{\ell(u)}$. Since u is at level $\ell(u)$ and it only increases from there, we get that this is sum is the same as $\sum_{i\geq 0} 2^{-i} \leq 2$. Iterating over all $2n$ nodes, we get $\sum_{v\in T} 2^{-\ell(v)} wt(v) \leq 4n$.

Adding a log factor to the statement above only makes it smaller, which gives us what we want. $\square$