MIT 6.854/18.415: Advanced Algorithms Spring 2016

Lecture 6 – February 22, 2016

Prof. Ankur Moitra *Scribe: Rio LaVigne, Matthew Staib, Dimitris Tsipras*

1 Last Time - JL Lemma

In the previous lecture, we described the Johnson-Lindenstrauss Lemma, a way of embedding a set of points into a subspace of significantly smaller dimension, while *approximately* preserving their pairwise distances. At a high level, we constructed an unbiased estimator for the ℓ_2-norm of vectors via Gaussians, and then boosted the success probability by averaging independent repetitions. We therefore saw that ℓ_2-distance is a property that is preserved when embedding into low dimensional subspaces. We finished with an overview of variants/extensions of the JL Lemma that included faster and sparser transformations.

2 Nearest Neighbor Search

Today, we are going to discuss the *Nearest Neighbor (NN)* search problem, and a powerful tool for it, *Locality Sensitive Hashing (LSH)*. While in previous lectures we viewed hashing collisions as an unpleasant obstacle that we tried to avoid, we will now assume a different view and exploit collisions to come up with faster algorithms.

2.1 Setup for NN

First, let us formally introduce the NN search problem.

Problem (NN): Given a set P of points in $\mathbb{R}^d$, construct a data structure for the set P, that can answer NN queries on P, that is given any point q in $\mathbb{R}^d$, return the closest point to q in P, i.e. return p s.t. $p \in \arg\min_{p \in P} \|p - q\|$.

We will evaluate our algorithms on two aspects, space and query time. Usually we try to get our query time to be as low as possible, without requiring huge amounts of space. Note that we didn't specify the norm we are considering, since problem is equally interesting in most norms.

2.2 A Motivating Example

Before we proceed, let us briefly discuss a motivating example. Consider the task of spam classification – given a set of emails labeled as *spam* or *not spam*, determine whether a new email is spam. A common approach is that of the k-Nearest Neighbors: find the k labeled emails "closest" to the unknown email and return some notion of "majority label". The usual notion of distance is through the Bag-of-Words model, where each email is associated with a vector of occurrence counts. A distinguishing characteristic of this approach is that while the vectors are very sparse,

they lie in some huge space, the dimension of which equals the number of words in the dictionary. We will soon see that dimension is a huge barrier when designing fast algorithms for the problem.

2.3 Approach #1

A naive algorithm for the problem, requiring absolutely no preparation of the data, is simply storing the points in a list and searching linearly among them to answer every query. While very simple, the guarantees of this algorithm are poor:

$$\text{space: } O(nd) \qquad \text{query time: } O(nd)$$

The main question here is: can we improve the query time? At what cost?

2.4 Approach #2

Let's first consider the case $d = 1$. In that case, our points lie on a line, so the problem can be solved easily by constructing a binary search tree:

$$\text{space: } O(n) \qquad \text{query time: } O(\log n).$$

Intuitively, the points partition the space into n regions (Figure 1), depending on which point in P is the closest point.

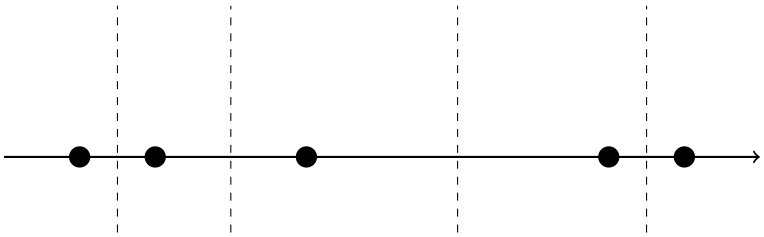

Figure 1: Line partitioning according to NN

Generalizing this notion for the case $d = 2$, we get a *Voronoi diagram* (Figure 2), a partition of the plane based on which query points map to a given point in P. It turns out that as the dimension grows, the description of such a partitioning becomes exponentially difficult. The multidimensional equivalent of binary search tree is the *k-d tree (k-dimensional tree)* described in [1].

All these approaches share a well-known problem in computational geometry, the infamous "curse of dimensionality" (a term coined by Bellman). Every known approach for NN is exponential in the dimension in terms of space, time or both.

3 Approximate Nearest Neighbor Search

In order to come up with practical algorithms for NN search in high dimensional spaces, we will have to relax our algorithm guarantees. Roughly speaking, we will allow for the query to return

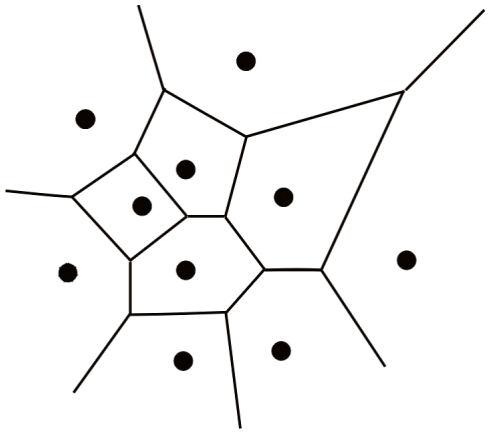

Figure 2: Voronoi diagram

points that are "close enough" instead of the closest. This problem is known as *Approximate Nearest Neighbors (ANN).*

Problem (c-ANN): Given a set P of points in $\mathbb{R}^d$, construct a data structure, that given any point q in $\mathbb{R}^d$, returns a point $p \in P$ such that $d(p,q) \leq c \min_{p' \in P} d(p',q)$.

3.1 The PLEB primitive

In order to make steps towards solving c-ANN, we will start by solving an easier problem –which we will later use as a primitive– namely *Point Location in Equal Balls (PLEB).*

Problem ((r_1, r_2)-PLEB): Given a set P of points in $\mathbb{R}^d$, and radii r_1, r_2, construct a data structure, that given some point q:

1. if there is a $p \in P$ such that $d(p,q) \leq r_1$, returns **YES** and any point $p' \in P$ with $d(p',q) \leq r_2$.
2. if there is no point $p \in P$ such that $d(p,q) \leq r_2$, returns **NO**.

Note that we do not care what happens in the case where there is a point within radius r_2 but not within radius r_1. This "gap" is where the approximate nature of our algorithms comes from.

Lemma 1. *If for all radius r, there exists a data structure with space S and query time T that solves $(r,(1+\varepsilon)r)$-*PLEB*, then there exists an algorithm for $(1+\varepsilon)^2$-*ANN *with space $O\left(S \log_{1+\varepsilon} \frac{D_{\max}}{D_{\min}}\right)$ and query time $O\left(T \log\log_{1+\varepsilon} \frac{D_{\max}}{D_{\min}}\right)$, where $D_{\min}, D_{\max}$ denote the minimum and maximum distance between points respectively.*

Proof. We will use our data structure for $(r,(1+\varepsilon)r) -$ PLEB to search over the following radii:

$$\frac{D_{min}}{2}, (1+\varepsilon)\frac{D_{min}}{2}, (1+\varepsilon)^2\frac{D_{min}}{2}, \cdots, \approx D_{max}.$$

We use a binary search to find the minimum radius r so that our algorithm returns **YES** for a single query point q. Let r^* be that minimum radius and let p be the returned point. The following are true:

1. $d(p, q) \leq (1 + \varepsilon)r^*$ from the definition of r^*.
2. For all $p' \in P$, $d(p', q) \geq \frac{r^*}{1+\varepsilon}$ because for the previous smallest radius, $(r^*/(1 + \varepsilon), r^*)$, our algorithm returned **NO**.

To conclude, let p^* be the nearest neighbor to q. We know that $d(p^*, q) \geq \frac{r^*}{1+\varepsilon}$. So, $d(p, q) \leq (1 + \varepsilon)^2 d(p^*, q)$. Therefore, p satisfies the requirements for being a $(1 + \varepsilon)^2$-approximate nearest-neighbor to q. □

It is possible to remove the extra $(1 + \varepsilon)$ factor with more efficient reductions. Indyk-Motwani [3] and [2] both show how to reduce $(1 + \varepsilon)$-ANN to $(r, (1 + \varepsilon)r)$-PLEB.

4 Locality Sensitive Hashing

So far, we have been using hash functions to perform tasks like load balancing, where it is better to avoid collisions – we do not want to assign two jobs to the same machine. However, collisions are not always bad! We will show how to exploit collisions to solve PLEB.

Definition 2. *A locality sensitive hash (LSH) [3] is a hash family where similar items are more likely to collide. Formally, a hash family* $\mathcal{H} = \{h : \mathcal{U} \to S\}$ *is called* (r_1, r_2, p_1, p_2)-locality sensitive *if for all points* $p, p' \in \mathcal{U}$*,*

1. *if* $d(p, p') \leq r_1$*, then* $\mathbb{P}[h(p) = h(p')] \geq p_1$*.*
2. *if* $d(p, p') > r_2$*, then* $\mathbb{P}[h(p) = h(p')] \leq p_2$*.*

Note that this definition only makes sense if $r_1 < r_2$ and $p_1 > p_2$.

4.1 Example of a Locality Sensitive Hash

Let $\mathbb{H}^d = \{0, 1\}^d$ be the d-dimensional binary cube, where our distance metric is hamming distance: $d(p, p') = \sum_{i=1}^{d} p_i \oplus p'_i$. Then, $\mathcal{H} = \{h_i : h_i(b_1, b_2, \cdots, b_d) = b_i \text{ for } i \in [d]\}$ is $(r, cr, 1 - \frac{r}{d}, 1 - \frac{cr}{d})$-locality sensitive.

This is because if $d(p, p') \leq r$, they have $d - r$ bits in common. So, the chance that a hash function h_i is pointing to a bit they have in common is $\frac{d-r}{d} = 1 - \frac{r}{d}$. Similarly, if $d(p, p') \geq cr$, they have $d - cr$ bits in common, so the chance that h_i maps them to the same bucket is $1 - \frac{cr}{d}$.

5 Solving PLEB

We will now show how to apply LSH, in order to solve PLEB.

Theorem 3 (Locality-Sensitive Hashing for PLEB [3])**.** *Suppose there is some* (r_1, r_2, p_1, p_2)*-LSH* $\mathcal{H} = \{h : U \to S\}$*. Then, there is an algorithm for* (r_1, r_2)*-PLEB which uses*

- $O(dn + n^{1+\rho})$ *space, and*
- $O(n^{\rho})$ *query time, measured in hash evaluations,*

where $\rho = \frac{\ln 1/p_1}{\ln 1/p_2}$. *This algorithm succeeds with constant probability.*

Over the rest of the section, we present an algorithm and then prove that it has these properties.

5.1 The algorithm

Let k and ℓ be parameters (which we will set later). Define a new hash family $\mathcal{G} = \{g : U \to S^k\}$. Each hash function $g \in \mathcal{G}$ takes the form $g(p) = (h_1(p), h_2(p), \ldots, h_k(p))$, where each hash function h_i is in $\mathcal{H}$.

To completely specify our algorithm, we need to specify both the preprocessing phase and how we respond to a query point q. First, preprocessing:

1. Choose hash functions $g_1, \ldots, g_\ell$ from $\mathcal{G}$.
2. For each p in the given set of points P, store p in each of the buckets specified by $g_1(p), \ldots, g_\ell(p)$.
3. Discard all empty buckets.

Now, when queried with the point q, we search through the buckets $g_1(q), \ldots, g_\ell(q)$, and stop after the first 2ℓ points. If any of these points p has $d(p, q) \leq r_2$, return p with YES; otherwise, return NO.

5.2 Analysis of the algorithm

For the algorithm to work, we need the following to hold (with constant probability):

1. If there is a $p \in P$ with $d(p, q) \leq r_1$, then there should exist j with $g_j(p) = g_j(q)$.
2. There are at most $2\ell - 1$ "bad" points $p \in P$ with have $d(p, q) > r_2$ but still do collide with q, with $g_j(p) = g_j(q)$ for some j.

In order to ensure (2) holds, we set $k = \log_{1/p_2} n$. Then, the expected number of points satisfying (2) is at most

$$(n \text{ points in } P) \cdot (p_2^k \text{ probability all } k \text{ hashes collide}) = n p_2^{\log_{1/p_2} n} = 1. \tag{1}$$

Then, by Markov's inequality, the probability that (2) does not hold is at most

$$\mathbb{P}[\#\text{ bad points} \geq 2\ell] \leq \frac{1}{2\ell} \leq \frac{1}{2}. \tag{2}$$

Now we must ensure (1) holds. Assume there is a $p \in P$ with $d(p,q) \leq r_1$. Fix an index j. The probability that $g_j(p) = g_j(q)$ is the probability that all k hashes h_i agree, which is bounded below by

$$p_1^k = p_1^{\log_{1/p_2} n} = \left(\left(\frac{1}{p_2} \right)^{\log_{1/p_2} p_1} \right)^{\log_{1/p_2} n} = n^{-\frac{\log 1/p_1}{\log 1/p_2}} = n^{-\rho}. \tag{3}$$

By considering all ℓ hashes g_j, and setting the number of hashes $\ell = n^\rho$, we see that (1) holds with probability at least

$$1 - (1 - n^{-\rho})^\ell = 1 - (1 - n^{-\rho})^{n^\rho} \geq 1 - \frac{1}{e} \geq \frac{1}{2}. \tag{4}$$

Hence, with probability at least $\frac{1}{4}$, (1) and (2) both hold, meaning this algorithm works with constant probability.

Observation 4. *Theorem 3 means that in order to solve PLEB (and consequently approximate nearest neighbors) on some set U, all we need to do is find a good locality-sensitive hash for U. Moreover, finding better and better locality-sensitive hashes automatically yields better and better algorithms for PLEB.*

5.3 Applications of the link between LSH and PLEB

Recall the example from Section 4.1, where $\mathbb{H}^d = \{0,1\}^d$ is the d-dimensional hypercube, and we refere to any point $x \in \mathbb{H}^d$ by its bits $(b_1, b_2, \ldots, b_d)$. The hash family

$$\mathcal{H} = \{h_i : h_i(b_1, b_2, \ldots, b_d) = b_i \text{ for } i = 1, 2, \ldots, d\} \tag{5}$$

is $(r, cr, 1 - \frac{r}{d}, 1 - \frac{cr}{d})$ for the Hamming distance.

Corollary 5. *There is an algorithm for (r, cr)-PLEB in $\mathbb{H}^d$ that uses space $O(dn + n^{1+\frac{1}{c}})$, and for each query needs $O(n^{\frac{1}{c}})$ evaluations of the hash function, each of which takes $O(d)$ time. In the language of Theorem 3, for the Hamming distance, the constant ρ satisfies $\rho \leq \frac{1}{c}$.*

Theorem 6 ([4])**.** *For Euclidean distance, we can find an even smaller ρ. Namely, $\rho \leq \frac{1}{c^2}$.*

Theorem 7 ([5])**.** *The above bounds for ρ in both the Hamming and Euclidean distance are tight.*

References

[1] Bentley, Jon Louis. Multidimensional binary search trees used for associative searching. *Communications of the ACM* 18.9 (1975): 509-517

[2] S. Har-Peled. 2001. A replacement for Voronoi diagrams of near linear size. In: *Proceedings of 42nd Annual symposium on Foundations of Computer Science, 2001.* p.94.

[3] Indyk, P. and Motwani, R., 1998, May. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing* (pp. 604-613). ACM.

[4] Andoni, A. and Indyk, P., 2006, October. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on* (pp. 459-468). IEEE.

[5] O'Donnell, R., Wu, Y. and Zhou, Y., 2014. Optimal lower bounds for locality-sensitive hashing (except when q is tiny). *ACM Transactions on Computation Theory (TOCT)*, 6(1), p.5.