CS 493: Algorithms for Massive Data Sets	**Estimating Similarity**
04/04/2002	**Scribe: Ding Liu**

1 Locality Sensitive Hash Functions

In this lecture we will be talking about algorithms for estimating similarity. Fix a collection $\mathcal{C}$ of objects. We define a similarity function $sim(x,y)$ that maps pairs of objects $x, y \in \mathcal{C}$ to a number in $[0,1]$. $sim(x,y)$ measures the similarity between x and y. $sim(x,y) = 1$ means that x and y are identical; $sim(x,y) = 0$ means that x and y are very different. For example when $\mathcal{C}$ is a set of unit vectors in $\mathbb{R}^d$, we can define $sim(\vec{u},\vec{v})$ to be $1 - \theta(\vec{u},\vec{v})/\pi$ where $\theta(\vec{u},\vec{v})$ is the angle between $\vec{u}$ and $\vec{v}$.

Given $\mathcal{C}$ and $sim(x,y)$, a locality sensitive hash function family $\mathcal{F}$ operates on $\mathcal{C}$, such that for any $x, y \in \mathcal{C}$,

$$\text{Prob}_{h\in\mathcal{F}}[h(x) = h(y)] = sim(x,y)$$

Below are two examples of locality sensitive hash function family.

- $\mathcal{C}$ is a collection of sets. For two sets $A, B \in \mathcal{C}$, $sim(A,B) = |A \cap B|/|A \cup B|$. Broder *et al.* [1] introduced the notion of *min-wise independent permutations*, which can be used to construct locality sensitive hash functions for this similarity measure, known as the *Jaccard coefficient*.

- $\mathcal{C}$ is a collection of unit vectors. For two vectors $\vec{u},\vec{v}$, $sim(\vec{u},\vec{v}) = 1 - \theta(\vec{u},\vec{v})/\pi$ where $\theta(\vec{u},\vec{v})$ is the angle between $\vec{u}$ and $\vec{v}$. A locality sensitive hash function family is defined as follows. Let $S^{d-1} \subset \mathbb{R}^d$ denote the unit $(d-1)$-sphere $\{\vec{v} \in \mathbb{R}^d : \|\vec{v}\| = 1\}$. Let $\vec{r}$ be a vector drawn uniformly at random from S^{d-1}. Corresponding to this $\vec{r}$ we have a hash function $h_{\vec{r}}$ such that $h_{\vec{r}}(\vec{u}) = 1$ if $\vec{r} \cdot \vec{u} \geq 0$, and $h_{\vec{r}}(\vec{u}) = 0$ if $\vec{r} \cdot \vec{u} < 0$. The following fact was used by Goemans and Williamson [3] as a rounding step in their approximation algorithm for MAX-CUT:

$$\text{Prob}[h_{\vec{r}}(\vec{u}) = h_{\vec{r}}(\vec{v})] = 1 - \theta(\vec{u},\vec{v})/\pi$$

The second example above also provides a way of constructing locality sensitive hash functions for sets, but for some similarity measure different from the first example above. Given a set A whose elements are drawn from a base set U. We can represent A by a vector $\vec{v_A} \in \{0,1\}^{|U|}$ as follows: if the i-th element of U belongs to A then the i-th bit of $\vec{v_A}$ is 1, otherwise it is 0. For this set of vectors, we apply the hashing scheme of the second example above. Thus we obtain a hashing scheme for a collection of sets such that:

$$\text{Prob}[h(A) = h(B)] = 1 - \delta/\pi$$

where

$$\cos\delta = \frac{\vec{v_A} \cdot \vec{v_B}}{|\vec{v_A}||\vec{v_B}|} = \frac{|A \cap B|}{\sqrt{|A||B|}}$$

2 Existence of Locality Sensitive Hash Functions

The material of this section is from [2]. The first result is a necessary condition for the existence of locality sensitive hash function families for a given similarity measure.

Theorem 2.1 *If similarity function $sim(x,y)$ admits a locality sensitive hash function family, then the distance function $1 - sim(x,y)$ satisfies the triangle inequality.*

Proof. Suppose there exists a locality sensitive hash function family $\mathcal{F}$ such that

$$sim(x,y) = \mathrm{Prob}_{h\in\mathcal{F}}[h(x) = h(y)]$$

Then

$$1 - sim(x,y) = \mathrm{Prob}_{h\in\mathcal{F}}[h(x) \neq h(y)]$$

Let $\Delta_h(x,y)$ be the indicator variable for the event $h(x) \neq h(y)$. That is, $\Delta_h(x,y) = 1$ if $h(x) \neq h(y)$, and $\Delta_h(x,y) = 0$ otherwise. We claim that $\Delta_h(x,y)$ satisfies the triangle inequality:

$$\Delta_h(x,y) + \Delta_h(y,z) \quad \geq \quad \Delta_h(x,z) \tag{1}$$

The reason is that the only possible way to violate inequality 1 is to let $\Delta_h(x,y) = \Delta_h(y,z) = 0$ and $\Delta_h(x,z) = 1$. But in this case $h(x) = h(y) = h(z)$, contradicting with $\Delta_h(x,z) = 1$. Take the expectation of inequality 1 over $h \in \mathcal{F}$, we get

$$\mathbf{E}_{h\in\mathcal{F}}[\Delta_h(x,y)] + \mathbf{E}_{h\in\mathcal{F}}[\Delta_h(y,z)] \quad \geq \quad \mathbf{E}_{h\in\mathcal{F}}[\Delta_h(x,z)] \tag{2}$$

To conclude the proof we observe that for any x, y

$$\mathbf{E}_{h\in\mathcal{F}}[\Delta_h(x,y)] = 1 - sim(x,y)$$

■

Theorem 2.1 could be used to prove that locality sensitive hash function families do not exist for certain set similarity measures. For example we can show that there is no locality sensitive hash function family for the Dice's coefficient defined as

$$sim_{Dice}(A,B) = \frac{|A \cap B|}{\frac{1}{2}(|A| + |B|)}$$

and for the Overlap coefficient defined as

$$sim_{Ovl}(A,B) = \frac{|A \cap B|}{\min(|A|, |B|)}$$

Consider the sets $A = \{a\}$, $B = \{b\}$, $C = \{a,b\}$. For the Dice's coefficient,

$$sim_{Dice}(A,C) = \frac{2}{3}, \quad sim_{Dice}(C,B) = \frac{2}{3}, \quad sim_{Dice}(A,B) = 0$$

$$(1 - sim_{Dice}(A, C)) + (1 - sim_{Dice}(C, B)) < (1 - sim_{Dice}(A, B))$$

Similarly for the Overlap coefficient:

$$sim_{Ovl}(A, C) = 1, \quad sim_{Ovl}(C, B) = 1, \quad sim_{Ovl}(A, B) = 0$$

$$(1 - sim_{Ovl}(A, C)) + (1 - sim_{Ovl}(C, B)) < (1 - sim_{Ovl}(A, B))$$

Sometimes it is desirable to have a hash function family that maps objects to 0 or 1. The following theorem says that we can always obtain such a binary hash function family, with some modification on the similarity measure.

Theorem 2.2 *Given a locality sensitive hash function family $\mathcal{F}$ for $sim(x, y)$, we can obtain another locality sensitive hash function family $\widehat{\mathcal{F}}$ that maps objects to $\{0, 1\}$ and corresponds to the similarity function $(1 + sim(x, y))/2$.*

Proof. Suppose we have a hash function family $\mathcal{F}$ such that

$$sim(x, y) = \mathrm{Prob}_{h \in \mathcal{F}}[h(x) = h(y)]$$

Let $\mathcal{B}$ be a pairwise independent family of hash functions that operate on the domain of the functions in $\mathcal{F}$ and map elements in the domain to $\{0, 1\}$. In other words for u, v in that domain, $\mathrm{Prob}_{b \in \mathcal{B}}[b(u) = b(v)] = 1/2$ if $u \neq v$, and $\mathrm{Prob}_{b \in \mathcal{B}}[b(u) = b(v)] = 1$ if $u = v$. We then consider the hash function family obtained by composing a hash function from $\mathcal{F}$ with one from $\mathcal{B}$. We denote it by $\widehat{\mathcal{F}}$ and show that it satisfies the requirements of this theorem.

Given two objects x, y. Fix $h \in \mathcal{F}$ and $b \in \mathcal{B}$. With probability $sim(x, y)$, $h(x) = h(y)$ and hence $b(h(x)) = b(h(y))$. With probability $1 - sim(x, y)$, $h(x) \neq h(y)$ and in this case, $\mathrm{Prob}_{b \in \mathcal{B}}[b(h(x)) = b(h(y))] = 1/2$. Thus,

$$\begin{aligned} \mathrm{Prob}_{h \in \mathcal{F}, b \in \mathcal{B}}[b(h(x)) = b(h(y))] &= sim(x, y) + (1 - sim(x, y))/2 \\ &= (1 + sim(x, y))/2 \end{aligned}$$

■

Theorem 2.2 can be used to prove a stronger condition for the existence of locality sensitive hash function families. We need a definition first.

Definition 2.3 *Given a collection X of objects. For any two objects $x, y \in X$ there is a distance $d(x, y)$. We say that $d(x, y)$ is isometrically embeddable in the d-dimensional Hamming cube if there exists a function $f : X \to \{0, 1\}^d$ such that for any x, y:*

$$|f(x), f(y)| = C \cdot d(x, y)$$

Here $|u \cdot v|$ is the Hamming distance between u and v and C is a constant.

Now we prove the following stronger condition for the existence of locality sensitive hash function families.

Theorem 2.4 *For any similarity function $sim(x, y)$ that admits a locality sensitive hash function family, the distance function $1 - sim(x, y)$ is isometrically embeddable in the d-dimensional Hamming cube for some integer d.*

Proof. We first apply Theorem 2.2 to construct a binary locality sensitive hash function family $\mathcal{F}$ for the similarity function $(1+sim(x, y))/2$. We arbitrarily order the hash functions of $\mathcal{F}$ and call them $h_1, \ldots, h_d$ (so $d = |\mathcal{F}|$). Given a object x, we map it to a binary vector $f(x) = (h_1(x), \ldots, h_d(x))$. Note that function f is an embedding of objects into the Hamming cube. We only need to show that it is an isometric embedding of the distance function $1 - sim(x, y)$.

Take any two objects x, y. $f(x) = (h_1(x), \ldots, h_d(x))$ and $f(y) = (h_1(y), \ldots, h_d(y))$. Recall that $\mathcal{F}$ has the following property (Theorem 2.2):

$$\mathrm{Prob}_{h \in \mathcal{F}}[h(x) \neq h(y)] = 1 - \frac{1 + sim(x, y)}{2}$$

This means that the number of index i such that $h_i(x) \neq h_i(y)$ is $d(1 - (1 + sim(x, y))/2)$, or equivalently,

$$|f(x), f(y)| = d/2(1 - sim(x, y))$$

In view of the Definition 2.3, this proves that $1 - sim(x, y)$ is isometrically embeddable in the Hamming cube. ■

We comment that Theorem 2.4 has a weak converse, that is, any isometric embedding of the distance function $1 - sim(x, y)$ in the Hamming cube yields a locality sensitive hash function family corresponding to the similarity measure $(\alpha + sim(x, y))/(\alpha + 1)$ for some α. Note that $(\alpha + sim(x, y))/(\alpha + 1)$ is in the range $[\alpha/(\alpha + 1), 1]$.

3 Estimating Similarity of Vectors

We consider the following problem: given a set V of n vectors in $\mathbb{R}^d$ and a query vector $\vec{q}$, decide if there exists a vector $\vec{v} \in V$ such that $sim(\vec{v}, \vec{q}) \geq \alpha$ where α is a constant (say 0.9). In other words, the algorithm answers "yes" if such a vector exists and reports it, and "no" otherwise. If we look at the probability of answering "yes" as a function of $\max_{\vec{v} \in V} sim(\vec{v}, \vec{q})$, then it is a threshold function as shown on the left part of Figure 1. In the following we show that by using locality sensitive hash functions we can approximate this threshold function. In fact we are presenting a randomized algorithm that solves this problem with constant probability.

Recall that in Section 1 we give a locality sensitive hash function family $\mathcal{H}$ for a collection of vectors in $\mathbb{R}^d$. We take k functions $h_1, \ldots, h_k$ from $\mathcal{H}$ uniformly at random, for some integer k. For a vector $\vec{v}$ we concatenate the k bits obtained by applying these k functions to $\vec{v}$. We use $g_1 \in \mathcal{H}^k$ to define this mapping, that is, $g_1(\vec{v}) = (h_1(\vec{v}), \ldots, h_k(\vec{v})) \in \{0, 1\}^k$. We apply g_1 to each $\vec{v} \in V$ and call the resulting n vectors a *group*.

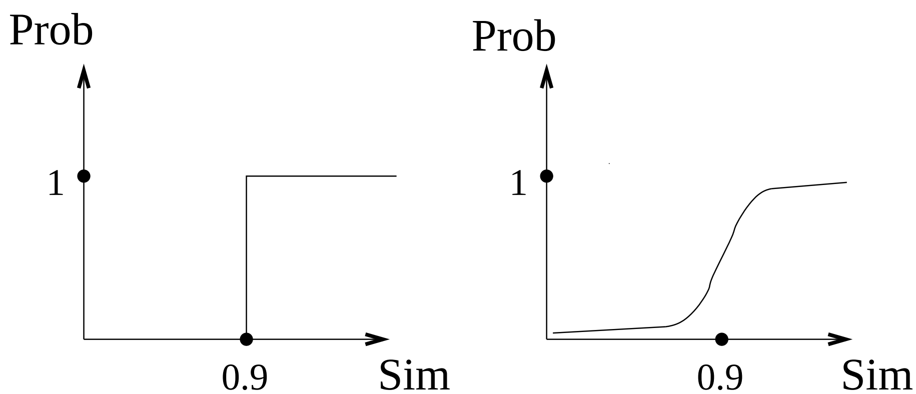

Figure 1: *Threshold function and its approximation.*

For some integer l, we form l groups independently. They are defined by l functions $g_1, \ldots, g_l$. During preprocessing we store $\vec{v}$ with each $g_j(\vec{v})$, for $1 \leq j \leq l$.

To answer a query $\vec{q}$, we compute $g_1(\vec{q}), \ldots, g_l(\vec{q})$. For each $g_j(\vec{q})$ we identify those $\vec{v}$ such that $g_j(\vec{v}) = g_j(\vec{q})$. Then for each $\vec{v}$ identified we test if $sim(\vec{v}, \vec{q}) \geq \alpha$. If this is the case then we answer "yes" and report $\vec{v}$. Otherwise we answer "no" after testing all the identified vectors.

Let $\vec{v_0}$ be the vector of V that maximizes $sim_{\vec{v} \in V}(\vec{v}, \vec{q})$, and let $p_0 = sim(\vec{v_0}, \vec{q})$. It is easy to see that the above algorithm is correct when $\vec{v_0}$ is one of those vectors identified. What is the probability for this to happen? We first look at g_1 and its k functions $h_1, \ldots, h_k$. For each h_i, $\mathrm{Prob}[h_i(\vec{v_0}) = h_i(\vec{q})] = p_0$. So $\mathrm{Prob}[g_1(\vec{v_0}) = g_1(\vec{q})] = p_0^k$. If $g_1(\vec{v_0}) = g_1(\vec{q})$ then we say that $\vec{v_0}$ and $\vec{q}$ agree on group g_1. So the probability that $\vec{v_0}$ and $\vec{q}$ *do not* agree on all l groups is $(1 - p_0^k)^l$. Our algorithm is correct when $\vec{v_0}$ and $\vec{q}$ agree on at least one group, which happens with probability $1 - (1 - p_0^k)^l$. If we plot this probability as a function of p_0 (with fixed k and l), we get something similar to the right part of Figure 1.

If we modify the above algorithm such that we test vectors that agree with the query on at least two groups, then the probability function becomes $1 - (1 - p_0^k)^l - lp_0^k(1 - p_0^k)^{l-1}$. This function looks similar to $1 - (1 - p_0^k)^l$, but "sharper" than the latter. That is, this new function moves from its low function value range (close to 0) to its high function value range (close to 1) more quickly than the previous one.

4 Approximate Nearest Neighbor Searching

In this section we consider the approximate nearest neighbor searching problem. The problem is as follows. Given a collection $\mathcal{C}$ of n objects from a base set $\mathcal{U}$. Given a distance function $D(x, y)$ that operates on any two objects $x, y \in \mathcal{U}$. We want to preprocess $\mathcal{C}$ such that given any on-line query $q \in \mathcal{U}$, we can quickly find an $(1 + \epsilon)$-approximate nearest neighbor of q in $\mathcal{C}$. We call p^* an $(1 + \epsilon)$-approximate nearest neighbor of q if for any $p \in \mathcal{C}$, $D(p^*, q) \leq$

$(1+\epsilon)D(p,q)$. This is a general form of the problem. For example, we can let $\mathcal{U}$ be $\mathbb{R}^d$, $\mathcal{C}$ be a set of n points, and $D(x,y)$ be the Euclidean distance between points x and y.

The approximate nearest neighbor searching problem (and its variants) has a considerable amount of literature. Here we list a couple of recent papers: Kleinberg [5], Indyk and Motwani [4], Kushilevitz, Ostrovsky, and Rabani [6].

The approximate nearest neighbor searching problem considered in this Lecture (and the next) is in the Hamming space. That is, $\mathcal{U}$ is H^d, each point is a 0-1 vector in dimension d, and $D(x,y)$ is $H(x,y)$ (the Hamming distance). We define the similarity between x and y as: $sim(x,y) = 1 - H(x,y)/d$. We also observe that locality sensitive hash function families exist for this similarity measure. For example the following d functions $h_1, \ldots, h_d$ constitute such a family: h_i maps $x \in \{0,1\}^d$ into its i-th bit.

We will be using the algorithm of Section 3. There we developped techniques for estimating similarity between vectors, but it is easy to see that the same strategy applies to other similarity measures as well. As a warm-up, we observe that by calling the algorithm of Section 3 $\log d$ times we can get a 2-approximate nearest neighbor (with certain probability). In fact, it suffices to ask the following sequence of questions: is there a point whose Hamming distance from the query is at most 1? at most 2? at most 4? and so on $\cdots$ In general the i-th question is: is there a point whose distance from the query is at most 2^i. To answer this question we set α (the similarity threshold) to $1 - 2^i/d$. The correctness of this 2-approximate nearest neighbor algorithm is obvious.

To solve the $(1+\epsilon)$-approximate nearest neighbor problem we solve the following (r_1, r_2)-neighbor problem first, with $r_1 < r_2$. The basic setting of this problem is the same as the nearest neighbor problem. But in the (r_1, r_2)-neighbor problem we want to distinguish between the following two cases:

- Determine whether there exists a point p whose distance from the query is at most r_1. If yes, then return a point p' whose distance from the query is at most r_2.
- Determine whether all points are at distance more than r_2 from the query.

We show how to solve this (r_1, r_2)-neighbor problem (with constant probability) in sublinear time. In the next lecture we will use it to solve the $(1+\epsilon)$-approximate nearest neighbor problem.

We use terminology from Section 3. For k and l specified later, we form l groups with each group defined by k locality sensitive hash functions. As what we did in Section 3, we compute $g_1(q), \ldots, g_l(q)$ and test those p such that p agrees with q in at least one group. In order to obtain sublinear running time (we will see this later) we interrupt the search after finding $2l$ points, including duplicates. Let $p_1, \ldots, p_t$ ($t \le 2l$) be the points we encountered. For each p_j if it is within distance r_2 from q then we answer "yes" and return p_j; we answer "no" if all the points are at distance more than r_2 from q. It is easy to see that the algorithm is correct if the following two properties hold with constant probability:

1. If there exists p such that $D(p,q) \le r_1$ then $g_j(p) = g_j(q)$ for some $j = 1, \ldots, l$.

2. The total number of p such that $D(p,q) \geq r_2$ and yet $g_j(p) = g_j(q)$ for some $j = 1, \ldots, l$ is less than $2l$.

In the next lecture we will show that by picking appropriate values for k and l we can: (1) make the desired probability lower bounded by a constant; (2) use this algorithm as a sub-routine in approximate nearest neighbor searching to achieve sublinear query time.

References

[1] Broder, A.Z., Charikar, M., Frieze, A., Mitzenmacher, M. *Min-wise independent permutations*, Proc. ACM STOC (1998), pp. 327 – 336.

[2] Charikar, M. *Similarity estimation techniques from rounding algorithms*, Proc. ACM STOC (2002), To appear.

[3] Goemans, M.X., Williamson, D.P. *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, J. ACM 42 (1995), 1115–1145.

[4] Indyk, P., Motwani, R. *Approximate nearest neighbors: Towards removing the curse of dimensionality*, Proc. ACM STOC (1998), 604–613.

[5] Kleinberg, J.M. *Two algorithms for nearest-neighbor search in high dimensions*, Proc. ACM STOC (1997), 599–608.

[6] Kushilevitz, E., Ostrovsky, R., Rabani, Y. *Efficient search for approximate nearest neighbor in high dimensional spaces*, SIAM J. COMPUT. 30 (2000), 457–474.