CPS290: Algorithmic Foundations of Data Science **February 16, 2017**

Lecture 7 : Locality Sensitive Hashing

Lecturer: Kamesh Munagala *Scribe: Kamesh Munagala*

It is conceivable that reducing dimension may be an overkill if our goal is simply to derive a faster algorithm for nearest neighbor search. We will now describe a general method that is remarkably similar to Bloom filters and Count-Min sketch; the main caveat is that it requires a distance-preserving hash function.

The Approximate Near-Neighbor Problem

Given a database D with n items, define the query $NN(q, r, c)$ as follows: Given query q and two parameters $r \geq 0$ and $c \geq 1$,

- If there exists $x \in D$ such that $D(q, x) \leq r$, then report some $y \in D$ such that $D(q, y) \leq cr$.
- If there is no $x \in D$ such that $D(q, x) \leq cr$, then report failure.
- Otherwise, the algorithm could either output a point at distance at most cr from q or report failure.

When $c = 1$, the above query is precise: If there is a point at distance at most r from q, the algorithm reports such a point; else it reports failure. We can now perform binary search over r and compute the nearest neighbor to an arbitrarily good approximation.

When c is much larger than 1, say $c = 3$, then the algorithm is good for distinguishing two cases: If all points in D are very far away from q – at distance at least $3r$ – the algorithm correctly reports failure. On the other hand, if there is a point at most distance r from q, the algorithm will report some point, but this could be a point further away – at distance at most $3r$ from q. It is easy to check that binary search over r yields a point y such that $D(y, q) \leq 3d^*$, where d^* is the distance of q to its nearest neighbor.

Locality Sensitive Hashing

Consider the hash function for Hamming distance that maps a d-dimensional bit vector to its value on a random coordinate. We showed that

$$\Pr[h(\vec{x}) = h(\vec{y})] = 1 - \frac{D(\vec{x}, \vec{y})}{d}$$

In other words, the hash functions are more likely to be equal if the distance between the points is smaller.

We generalize this definition as follows:

Definition 1. *For a parameter $c > 1$, probability values $p_1 > p_2$, and distance $r \geq 0$, A hash family $\mathcal{H}$ is said to be (r, cr, p_1, p_2)-Locality Sensitive (LSH) if $\forall q, x, y$, we have:*

- *If $D(x, q) < r$, then $\Pr[h(x) = h(q)] \geq p_1$, and*
- *If $D(y, q) > cr$, then $\Pr[h(y) = h(q)] \leq p_2$*

where the probability is over $h \in \mathcal{H}$ chosen at random.

LSH for Hamming Distance

In the case of Hamming distance, we will have

$$p_1 = 1 - \frac{r}{d} \qquad p_2 = 1 - \frac{cr}{d}$$

The running times of the algorithms we present will depend on the parameter

$$\rho = \frac{\ln(p_1)}{\ln(p_2)}$$

that in turn depends on the separation between p_1 and p_2. For instance, assuming $c \times r \ll d$ and $c > 1$, for the Hamming distance, the calculation is roughly:

$$\rho = \frac{\ln(p_1)}{\ln(p_2)} \approx \frac{r/d}{cr/d} = \frac{1}{c}$$

LSH for Jaccard Similarity

For Jaccard measure, choose a random permutation π on the universe U. For set $S \subseteq U$, the LSH for Jaccard measure is simply:

$$h(S) = \text{First element in } S \text{ according to permutation} \pi$$

Consider two sets S_1 and S_2. The Jaccard measure between them is

$$J(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

It is an easy exercise to check that

$$\Pr[h(S_1) = h(S_2)] = J(S_1, S_2)$$

where the probability is over the choice of permutation π. If we define distance as:

$$D(S_1, S_2) = 1 - J(S_1, S_2)$$

then it is easy to check that the hash family is LSH for any $r > 0$ and $c > 1$. In particular

$$D(S_1, S_2) < r \qquad \Rightarrow \qquad \Pr[h(S_1) = h(S_2)] \geq 1 - r$$

and

$$D(S_1, S_2) > cr \qquad \Rightarrow \qquad \Pr[h(S_1) = h(S_2)] \leq 1 - cr$$

The value $\rho \approx 1/c$ assuming $cr \ll 1$.

In practice, storing a permutation is difficult, so it is implemented by a universal hash function with range roughly equal to $|U|$.

LSH for Angular Similarity

The angular distance between two vectors $\vec{x}$ and $\vec{y}$ is the angle between them:

$$D(\vec{x}, \vec{y}) = \cos^{-1}\left(\frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\|\|\vec{y}\|}\right)$$

The LSH is simply the set of all random unit vectors:

$$h_{\vec{r}}(\vec{x}) = \text{Sign}\,(\vec{x} \cdot \vec{r})$$

In other words, the hash function maps $\vec{x}$ to 1 if $\vec{x}$ makes an acute angle with $\vec{r}$, and to -1 otherwise.

For two vectors $\vec{x}$ and $\vec{y}$, let θ denote the angle between them. A simple calculation shows

$$\Pr[h(\vec{x}) = h(\vec{y})] = 1 - \frac{\theta}{\pi} = 1 - D(\vec{x}, \vec{y})$$

Just as before, if we define distance as the angle between vectors scaled down by π, the above hash family is LSH for any $r \in [0, 1]$ and $c > 1$.

The LSH Algorithm and Analysis

Suppose we are given a hash family $\mathcal{H}$ that is (r, c, p_1, p_2)-LSH. Our goal is to use it as a subroutine to design an algorithm for the $NN(q, r, c)$ problem.

Fix two parameters k, ℓ. Let $\mathcal{G}$ be the set of all k-dimensional vectors, each coordinate of which is a hash function h chosen *i.i.d* from $\mathcal{H}$. That is,

$$\mathcal{G} = \{\langle h_1, h_2, \ldots, h_k\rangle \mid h_j \sim \mathcal{H}\}$$

Choose $g_1, g_2, \ldots, g_\ell$ i.i.d. from $\mathcal{G}$.

Given $x \in D$, for each $i = 1, 2, \ldots, \ell$, store x in the bucket $g_i(x) = \langle h_{i1}(x), h_{i2}(x), \ldots, h_{ik}(x)\rangle$. Note that the bucket is defined by a vector, so that for each $i = 1, 2, \ldots, \ell$, the number of buckets is exponential in k. However, we only store the buckets which have at least one item mapping to it. (How will you do this?)

Given query q, compute $g_1(q), g_2(q), \ldots, g_\ell(q)$. Consider these buckets in order, and for each $x \in D$ in these buckets, compute the distance from x to q. If any of these is at most $c \times r$, report this item and stop. Otherwise continue till 4ℓ items have been checked. At this point, stop and declare failure.

Analysis

Suppose there is a point $x^* \in D$ such that $d(x^*, q) \leq r$. The analysis hinges on showing that two properties hold with constant probability:

1. For some $i = 1, 2, \ldots, \ell$, we have $g_i(x^*) = g_i(q)$; and
2. There are at most 4ℓ items $x \in D$ with $D(x, q) > cr$, such that for some i, $g_i(x) = g_i(q)$.

Assuming these properties hold, the second property implies that there are at most 4ℓ false positives, meaning items that hash to the same location as q but that have distance more than cr from q. This means the algorithm will not encounter the case where it has to stop because it found too many false positives. The first property then implies that in the worst case, the algorithm encounters x^* for some value of i and will return it. (It could return some other item if it encounters it earlier, but the property guarantees it returns something.) This shows the algorithm will be correct with constant probability – it will return an item whose distance from q is at most cr, assuming there is one item whose distance from q is at most r. Though this probability is only a constant, we can boost it by replicating the above data structure independently $O(\log n)$ times, and running the same algorithm on all these copies.

Running Time and Space Requirement. Before we prove that the properties hold with constant probability, what is the running time and space required? The number of false positives allowed is 4ℓ, so this is the worst case number of distance computations the algorithm performs. We need to make this number much smaller than n. We will show below that $\ell = n^\rho$, which is roughly $n^{1/c}$ for the hash functions discussed above, including for Hamming distance, and Jaccard and Cosine similarities. Assuming $c = 2$ for instance gives a running time of $O(\sqrt{n})$ distance computations – far better than brute force! This is the magic of LSH. Note however, that each data item is stored ℓ times, so that the space needed is $O(n\ell) = O(n^{1+1/c})$. So the space blows up, but is sub-quadratic. Note further that each hash function g is a concatenation of k hash functions h. Therefore, the number of evaluations of h is $4k\ell$. We will choose $k = \frac{\log n}{\log(1/p_2)} \approx \frac{1}{cr}\log n$ for Jaccard and angular. This means the number of evaluations is roughly $O\left(\frac{\ell}{cr}\log n\right)$, which is sub-linear in n, assuming cr is not too small.

Theorem 1. *Let $\rho = \frac{\ln(p_1)}{\ln(p_2)}$. Let $\ell = n^\rho$, and $k = \frac{\log n}{\log(1/p_2)}$. Then both properties (1) and (2) hold with probability at least $1/3$.*

Proof. Consider the second property. For some $x' \in D$ with $D(x', q) > cr$, the LSH property implies that for any i,

$$g_i(x') = g_i(q) \leq p_2^k$$

Setting this to $1/n$ yields $k = \frac{\log n}{\log(1/p_2)}$. This means that for fixed i, the expected number of x' that map to the same bucket as q is at most $n \times 1/n = 1$. Applying linearity of expectation over all ℓ buckets, the expected number of false positives is at most ℓ. By Markov's inequality, the probability there are more than 4ℓ false positives is therefore at most $1/4$.

For the first property, note that for any i,

$$\Pr[g_i(x^*) \neq g_i(q)] \leq 1 - p_1^k = 1 - n^{-\rho} = 1 - \frac{1}{n^\rho}$$

where we used $k = \frac{\log n}{\log(1/p_2)}$ and $\rho = \frac{\ln(p_1)}{\ln(p_2)}$. Since $\ell = n^\rho$, we have

$$\Pr[g_i(x^*) \neq g_i(q)\ \forall i] \leq \left(1 - \frac{1}{n^\rho}\right)^{n^\rho} \leq \frac{1}{e}$$

This means the first property holds with probability at least $1 - 1/e$.

By union bounds, the probability that both the first and second properties hold with probability at least $1 - 1/4 - 1/e \geq 0.382 \geq 1/3$. □