6.897: CSCI8980 Algorithmic Techniques for Big Data September 26, 2013

Lecture 6

Dr. Barna Saha *Scribe: Huiqi Xu*

Overview

Last class we started to introduce nearest neighbor problem and talked about how to solve it when data point is high-dimensional. In today's class we will continue to discuss more details about approximating nearest neighbors problem.

1 Introduction to Nearest Neighbor Problem

Suppose a set of data points is V where data size is $|V| = N$ and the dimension of data point is D. The goal is given a distance function d, returning nearest neighbor of any query point q in V such that we can return a data point $x \in V$ whose d(x, q) is minimum.

For 2-dimensional data points, if we use some data structure to preprocess and store all points into this data structure, the space complexity to answer nearest neighbor is O(N) and query time complexity is O($\log N$). However, for D-dimensional data, the space cost is increased to O(N^D) while time cost is O($D \log N$). So space complexity for data structure to store all points is dramatically increased with the increasing of D, which is inappropriate for high-dimensional data. If we don't use any data structure to preprocess points in V, we can compute all distances between any point in V and query point, then get the minimum one. The space complexity is decreased to O(N*D), but the query time complexity is O(N*D). This is very inefficient when data size is large. In order to solve this problem with the constraint of space and time, we extend it to solve two variations of nearest neighbor problem. The variations is mainly to find data points within a radius of query point q because if we can find those points in a radius efficiently, we can always use binary search for radius to find a good one which only covers the nearest point of the query point.

R-Near Neighbor Problem. Given a set of data points V and the distance function d, when query point q comes, it returns data point x such d(x, q) $\leq$ R.

(C,R)-Near Neighbor Problem Given a set of data points V and the distance function d, when query point q comes, it returns data point x such d(x, q) $\leq$ CR.

To solve R-Near Neighbor and (C,R)-Near Neighbor problems, we use locality sensitive hashing to return the answer.

2 Locality Sensitive Hashing

Given a family of hash functions $\mathcal{H}$ and a distance function d, we call (C,R,P_1,P_2)-sensitive if

$$Prob_{h \sim \mathcal{H}}(h(x) = h(y) | d(x,y) \leq R) \geq P_1 \tag{1}$$

$$Prob_{h\sim\mathcal{H}}(h(x) = h(y)|d(x,y) > CR) \leq P_2 \tag{2}$$

where $P_1 > P_2$.

Suppose h is an elementary hash function, we pick L $\times$ K hash functions from $\mathcal{H}$ and get the following composite hash functions g:

$$\begin{aligned} g_1 &=< h_{1,1}, h_{1,2}, \ldots, h_{1,K} > \\ g_2 &=< h_{2,1}, h_{2,2}, \ldots, h_{2,K} > \\ &\vdots \\ g_L &=< h_{L,1}, h_{L,2}, \ldots, h_{L,K} > \end{aligned} \tag{3}$$

Hadmming Distance. The definition of hamming distance is the number of different digits between two points. For example, if two data points are s_1=00110100 and s_2=10010100, the distance between s_1 and s_2 is $d(s_1, s_2) = 2$. If we define hash functions $\mathcal{H} = \{h_1, h_2, \ldots, h_D\}$ and $h_i(x) = x_i$, we have

$$\begin{aligned} P_1 &= Prob(h(x) = h(y)|d(x,y) \leq R) \geq \frac{D-R}{R} = 1 - \frac{R}{D} \\ P_2 &= Prob(h(x) = h(y)|d(x,y) > CR) \leq \frac{D-CR}{D} = 1 - \frac{CR}{D} \end{aligned} \tag{4}$$

If we set K=3, L=2 and $g_1 =< h_1, h_3, h_5 >$, $g_2 =< h_2, h_6, h_8 >$, then $g_1(s_1) = 011$, $g_1(s_2) = 011$, $g_2(s_1) = 010$, $g_2(s_2) = 001$

3 Algoritm

The following is the algorithm procedures to compute (C,R)-Near neighbor problem. There are two steps. The first step is preprocessing step and it's reading all data points using every hash function g_j to map a data point into a bucket. The time complexity is O(NKL). Even though the time cost could be very large when data size is large, this procedure could be off-line and once it's done we don't need to linear scan all data points again. So the time cost is affordable. The second step is for query processing. It's finding every bucket[g_j(q)] for every j from 1 to L. Then we check if there exists a point x$\in$bucket[g_j(q)] such that d(x,q) $\leq$ CR, we return x.

Algorithm 1 Preprocessing

```
for all i ∈ V do
  for $j = 1$ to $L$ do
    insert i → bucket[$g_j$(i)]
  end for
end for
```

Suppose F is the probability that x is hashed to the same bucket as q using some g_i, but d(x,q)>CR,

Algorithm 2 Query(q)

```
for $j = 1$ to $L$ do
  for all x $\in$ bucket[$g_j$(q)] do
    if d(x,q) $\leq$ CR then
      return x;
    end if
  end for
end for
```

then the query time is O(KL+NFL).

$$\begin{aligned}F &= Prob(g_i(x) = g_i(q)|d(x,q) > CR)\\ &= Prob(h_{i,1}(x) = h_{i,1}(q) \wedge h_{i,2}(x) = h_{i,2}(q) \wedge \dots h_{i,K}(x) = h_{i,K}(q)|d(x,q) > CR)\\ &= \prod_{j=1}^{K} Prob(h_{i,j}(x) = h_{i,j}(q)|d(x,q) > CR)\\ &\leq P_2^K\end{aligned} \tag{5}$$

Success Probability is

$$\begin{aligned}Prob(sucsess) &\geq Prob(\exists j, j \in [1,L], g_j(x) = g_j(q)|d(x,q) \leq R)\\ &= 1 - Prob(g_1(x) \neq g_1(q) \wedge g_2(x) \neq g_2(q) \dots g_L(x) \neq g_L(q)|d(x,q) \leq R)\\ &= 1 - \prod_{j=1}^{L} Prob(g_j(x) \neq g_j(q)|d(x,q) \leq R)\end{aligned} \tag{6}$$

Because

$$\begin{aligned}Prob(g_j(x) \neq g_j(q)|d(x,q) \leq R) &= Prob(\exists p, p \in [1,K], h_{j,p}(x) \neq h_{j,p}(q)|d(x,q) \leq R)\\ &= 1 - Prob(\forall p \in [1,K], h_{j,p}(x) = h_{j,p}(q)|d(x,q) \leq R)\\ &= 1 - \prod_{p=1}^{K} Prob(h_{p,j}(x) = h_{p,j}(q)|d(x,q) \leq R)\\ &\leq 1 - P_1^K\end{aligned} \tag{7}$$

So we can get

$$Prob(success) \geq 1 - [1 - P_1^K]^L \tag{8}$$

Because of the equation 5, the query running time is bounded by O(KL+NLP_2^K).

4 Analysis of Choosing K and L

If we set $P_1^K = \frac{1}{L}$, Prob(success) $\geq$ 1-$[1 - \frac{1}{L}]^L \simeq$ 1-$\frac{1}{e}$ which is satisfying for success probability. Then we can get $K = \frac{\log L}{\log \frac{1}{P_1}}$. So K is O(log L). Because N could be extremely large, we want the query cost O(KL+NLP_2^K) is dominated by KL but not related to N, NLP_2^k must be bounded by O(L). If NLP_2^k=L, we have

$$N = \frac{1}{P_2^K} \quad = (\frac{1}{P_1})^{k\frac{\log\frac{1}{P_2}}{\log\frac{1}{P_1}}} = L^{\frac{\log\frac{1}{P_2}}{\log\frac{1}{P_1}}}, because(\frac{1}{P_1})^K = L \tag{9}$$

So if we set $\frac{\log\frac{1}{P_1}}{\log\frac{1}{P_2}} = \rho$, we get $L = N^\rho$. Because R $<$ CR $\ll$ D, if the distance function is hamming distance, P_1=1-$\frac{R}{D} \simeq e^{\frac{-R}{D}}$, $P_2 = 1 - \frac{CR}{D} \simeq e^{\frac{-CR}{D}}$ and $\rho = \frac{\log\frac{1}{P_1}}{\log\frac{1}{P_2}} = \frac{RD}{CRD} = \frac{1}{C}$.

Considering the space complexity and time complexity, the space cost is $O(NKL) = \widetilde{O}(NN^\rho) = \widetilde{O}(N^{1+\frac{1}{C}})$. The query cost is O(KL+NKP_2^K=O(KL)=$\widetilde{O}(L)$=$\widetilde{O}(N^{\frac{1}{C}})$.

5 LSH in l_1 distance

When distance function d is l_1 instead of hamming distance, it is possible to obtain an isotropic embedding of the hamming space into l_1 space. The net effect is an increase in dimension, but once such embedding has been done, one can use the same LSH functions for hamming distance to compute near neighbor in l_1 distance. Recall that l_1-distance between two D-dimensional points in $\mathbb{R}^D$ $x = (x_1, x_2, \ldots, x_D)$ and $y = (y_1, y_2, \ldots, y_D)$ is defined as $l_1(x,y) = |x_1 - y_1| + |x_2 - y_2| + \ldots + |x_D - y_D|$. Assuming finite precision, by proper scaling and shifting it is possible to assume that each coordinate of the points in $V \in \mathbb{R}^d$ is at most $M \in \mathbb{N}$. We allocate M bits for each each coordinate and use unary representation for converting each l_1 coordinate into hamming space. For example, a coordinate with value A will be transformed into M bits with A leading bits as 1 and remaining $(M - A)$ tailing bits as 0. For example, two data points x=(2,4), y=(3,1), if we assume $M = 10$, x could be transformed to x=(2,4)=(1100000000,1111000000) and y could be transformed to y=(3,1)=(1110000000,1000000000). It is now straight-forward to verify that l_1 distance between the original points is same as the hamming distance between the transformed points.