11.1 (c,R) - near neighbor problem

Notations:
\( d(.,.) \rightarrow \) distance metric (may not always be given explicitly). For eg., for \( L_2 \), we know what \( d \) is.
\( U \) is the universe of all elements which again may not be given explicitly.
\( S \subseteq U \) is the database of elements which is explicitly given.

Query(x):
If \( \exists y \in S \) such that \( d(x,y) \leq R \), then with probability at least \( 1 - \delta \),
Output \( z' \) such that \( d(x,z) \leq cR \)
Never return \( z' \) such that \( d(x,z) > cR \)

11.2 LSH family:

Family of hash functions \( H \) is \((c,R,p_1,p_2) - LSH\) for \( U \) if
i) \( \Pr_{h \sim H}(h(x) = h(y)) \geq p_1 \) when \( d(x,y) \leq R \)
ii) \( \Pr_{h \sim H}(h(x) = h(y)) \leq p_2 \) when \( d(x,y) > cR \)

Usually, we want \( p_2 < p_1 \)

Eg: If \( U \) is the set of numbers on a straight line, what is a reasonable hash family ?
First attempt: \( h(x) = \lceil x \rceil \)
One can easily verify that \( R = 0.25, c = 2, x = 1.9, y = 2.1 \) results in a violation of our requirements.

Observe that the hash function we used above is a specific hash function as opposed to a hash family. Consider the family of hash functions \( H = [0,1) \). We parametrize every hash function belonging to the family with a random number \( z' \). So, we have \( h_z(x) = \lfloor x - z \rfloor \)
This essentially amounts to translating the coordinate system by a random number \( z' \) and then taking floor.
\( H = \{ h_z : z \in [0,1) \} \)

11.3 How to use LSH ?

We introduce two parameters \( L \) and \( R \) the motivation for which will become evident subsequently.
Consider hash functions \( g_1, g_2 \ldots g_L \) such that \( g_j(x) = h_j.1(x), h_j.2(x), \ldots h_j.K(x) > \) where \( h_{j,i} \) s are chosen independently and uniformly at random from \( H \).
\( \Pr[g_j(x) = g_j(y)] \geq (p_1)^K \) when \( d(x,y) \leq R \)
Pr[\(g_j(x) = g_j(y)\)] \leq (\(p_2\))^K \text{ when } d(x, y) > cR

The intuition behind \(g_j\) is that it amplifies the difference between \(p_1\) and \(p_2\) and in the process, it also drives down both the values. Note that we cannot choose \(K\) to be too large because otherwise, even points that are very close to each other may not fall in the same hash bucket of \(g_j\). As one can see, \(L\) is just used to repeat the whole experiment many times.

11.4 Preprocessing:
\(\forall x \in S\), compute \(<g_1(x), g_2(x) \ldots g_L(x)>\). We have \(O(KL)\) hash function computations per element \(x\).
Total time = \(O(NKL)\) where \(N = |S|\)
Total space (memory) = \(O(NKL)\)

11.5 Query(y):
Compute \(g_1(y), g_2(y) \ldots g_L(y)\)
\(\forall \in S\), \(\forall 1 \leq j \leq L\), if \(g_j(x) = g_j(y)\), then mark \(x\) as a candidate.
Find the distance from \(y\) to all candidates and return the first \(x^*\) such that \(d(x^*, y) \leq cR\).
Given \(y\),

\begin{verbatim}
for j = 1 to L do
  foreach x s.t g_j(x) = g_j(y) do
    if d(x, y) \leq cR then
      return x [i.e exit]
  end
end
return “NO CLOSE POINT”
\end{verbatim}

Algorithm 1: (c,R)-near neighbor

How can we do the inner “foreach” loop efficiently ?
During the preprocessing phase, store an inverted index i.e, aggregate all 'x' s for a given \(g_j(x)\) value. Following is the corresponding map function that would enable us to do it on map-reduce.

MAP(x):
\(\forall j, 1 \leq j \leq L\)
Emit \(<j, g_j(x)>, x)\)

Issues:
i) Inefficiency: Many far away points hashing to the same bucket \(g_j\) for some \(j\). So the inner “foreach” loop is inefficient.
ii) Correctness: A close point that does not map to the same bucket for any \(j\).
Addressing the issues:

i) Correctness:
We want to make sure that if \( x \) and \( y \) are close, i.e, \( d(x, y) \leq R \), then \( g_j(x) = g_j(y) \) for some \( j \).

\[
\Pr[g_j(x) \neq g_j(y)] = 1 - p_1^K
\]

\[
\Pr[\forall j, g_j(x) \neq g_j(y)] = (1 - p_1^K)^L
\]

If we ensure \( p_1^K = \frac{1}{L} \),

then \( \Pr[\forall j, g_j(x) \neq g_j(y)] = \frac{1}{e} \)

\( \Rightarrow \) with probability \( 1 - \frac{1}{e} \), \( g_j(x) = g_j(y) \) for some \( j \).

ii) Efficiency:

**Expected query time:** \( = O(L + NLF) \)

where \( N = |S| \), \( L \) is the \# outerloop iterations, \( F \) is the false positive probability. The additive \( L \) term is required because even if \( F \) is very very small, we still need to do the outer loop. In other words, expected query time is also \( \Omega(L) \).

\[
F = \Pr[g_j(x) = g_j(y)] \text{ when } d(x, y) > cR.
\]

**Goal:** Set \( F = \frac{1}{N} \)

\( \Rightarrow p_2^K = \frac{1}{N} \)

Solving for \( N \) and \( L \), we have

\[
p_2^K = (p_1^K)^{\frac{\log p_2}{\log p_1}}
\]

\[
\frac{1}{N} = \frac{1}{L}^{\frac{\log p_2}{\log p_1}}
\]

\( \Rightarrow L = N^{\frac{\log p_1}{\log p_2}} \)

\( = N^{\log \frac{1}{p_2}} \)

(1)

Call the exponent of \( N \) as \( \rho \) which is a property of the hash function family.

So \( L = N^\rho < 1 \)

Substituting for \( K \), we have

\[
K = \frac{\log L}{\log \frac{1}{p_1}}
\]

\( K \) is small, around \( \log N \) whereas \( L \) is \( \text{poly}(N) \) but still sublinear since \( \rho < 1 \).