COMPSCI 630: Randomized Algorithms February 26, 2018

Lecture 12

Lecturer: Rong Ge *Scribe: Ethan Holland*

1 Overview

In this lecture, we discuss Locality Sensitive Hashing, an application of dimension reduction and random projections. We begin by considering the Nearest Neighbor Problem, a common problem in Machine Learning. We then discuss relaxations of the Nearest Neighbor Problem and demonstrate that Locality Sensitive Hashing can solve the Approximate Nearest Neighbor in $o(n)$ time.

2 Nearest Neighbor Problem

2.1 Definition and Motivation

Definition 1 (Nearest Neighbor Problem (NN))**.** *Given points $x_1, ..., x_n \in \mathbb{R}^d$ and query point q, the Nearest Neighbor Problem asks for the point x_i which minimizes $||x_i - q||$*

Remark 1. *The distance metric in NN is often, but not necessarily, Euclidean distance*

The Nearest Neighbor Problem is a common problem in Machine Learning applications, most commonly in making recommendations. For example, if a website such as Netflix wants to recommend television show to users, they may find shows liked by users with similar taste. Often, a recommendation is made by taking a weighted average of the k nearest neighbors.

A naïve approach to solving NN might be to calculate $||x_i - q||$ for every i. This takes $O(nd)$ space and $O(nd)$ time. This is typically too slow, especially if the points are in high dimension, since calculating distance between two points takes time proportional to their dimension.

In order to reduce the run time, we can perform some prework which will then allow each query q to be resolved more quickly. For example, in one dimension we can sort the points and then perform a binary search to find the nearest neighbor for each query q. This takes $O(logn)$ for each query. There are other data structures which can be used in low dimension, but for sufficiently large dimension one cannot get performance much better than the naïve approach. However, we can achieve a reasonable running time by relaxing some of the constraints of the problem.

2.2 Approximate Nearest Neighbor (ANN)

Definition 2 (Approximate Nearest Neighbor Problem (multiplicative version))**.** *Given points $x_1, ..., x_n \in \mathbb{R}^d$, metric $||\cdot||$, and query point q, ANN asks for a point x_i such that $||x_i - q|| \leq c \min_j ||x_j - q||$*

Remark 2. *There are many forms of ANN. We examine only the multiplicative formulation*

Here we have relaxed the constraint that we must find the nearest neighbor, and instead require a point within distance cr where r is the distance to the nearest neighbor. One could apply the Johnson-Lindenstrauss Lemma to this problem to reduce the dimension of the points while preserving distances

within a multiplicative factor. This would reduce the run time from $O(nd)$ to $O(nd')$ where d' is the new dimension. However, this doesn't actually solve the problem since we still need to compare against every point and thus the run time is still linear in the number of points n. In order to achieve the desired run time we consider a simplified version of the Approximate Nearest Neighbor Problem.

Definition 3 (Simplified Approximate Nearest Neighbor Problem)**.** *Given points $x_1, ..., x_n \in \mathbb{R}^d$, metric $||\cdot||$, query point q, and the promise that there exists some x_j such that $||x_j - q|| \leq r$, the Simplified Approximate Nearest Neighbor Problem asks for a point x_i such that $||x_i - q|| \leq cr$*

Remark 3. *r is called the radius of interest. If we choose $r = \min_j ||x_j - q||$ then Simplified ANN is identical to ANN.*

Claim 1 (without proof)**.** *Solving Simplified ANN is equivalent to solving ANN*

proof sketch: Suppose we can solve Simplified ANN for any radius of interest r and have some bound D on the maximum possible distance between points. To solve ANN we can binary search on the minimum radius of interest r such that simplified ANN returns at least one point.

3 Locality Sensitive Hashing

3.1 Definition and Motivation

Locality Sensitive Hashing (LSH) is the most efficient technique that we know of for solving ANN for high dimensions.

Definition 4 (Locally Sensitive Hash Family)**.** *Given parameters p_1, p_2 with $1 > p_1 > p_2 > 0$, hash family F is a p_1, p_2 Locality Sensitive Hash Family if*

- *$\forall x, y$ such that $||x - y|| \leq r$, $\Pr_{f \in F}[f(x) = f(y)] \geq p_1$*
- *$\forall x, y$ such that $||x - y|| \geq cr$, $\Pr_{f \in F}[f(x) = f(y)] \leq p_2$*

Remark 4. *F can depend on r, c, p_1, p_2, and the metric $||\cdot||$.*

Remark 5. *The only randomness in the system is in the choice of f from F; every other variable is fixed.*

Intuitively, we can consider an LSH function to be a random partition of space. For example, the set of all hyperplanes in $\mathbb{R}^d$ is an LSH family.

3.2 Application of LSH to ANN

In order to solve ANN in time less than $\theta(n)$ we must reduce the number of points we compare with the query q. In filtering points we have two goals:

1. Filter out points far from q
2. Preserve at least one point which is close to q

We will formalize these goals later on. Consider the following algorithm:

Algorithm 1

Sample $f_1, ..., f_k$ uniformly and independently from LSH family F
for x_i **do**
  map x_i to $(f_1(x_i), ..., f_k(x_i))$ and hash this value (this hash is not locality sensitive)
end for
for query q **do**
  calculate $(f_1(q), ..., f_k(q))$ and corresponding hash value
  for every point x_i in the same hash bucket as q **do**
    if $||x_i - q|| \leq cr$ **then**
      return x_i
    end if
  end for
end for

Claim 2. *For any q, the expected number of points in the same hash bucket as q that are at distance at least cr from q is at most* $n(p_2)^k$

Proof. By the definition of LSH $\forall q, x_i$ if $||q - x_i|| \geq cr$ then $\Pr_j[f_j(q) = f_j(x_i)] \leq p_2$. Since f_a, f_b are independent for $a \neq b$, the probability that $f_j(q) = f_j(x_i)$ for all j is at most $(p_2)^k$. Since each of the n points is in the same hash bucket as q with probability $(p_2)^k$ the expected number of far points with the same hash as q is $n(p_2)^k$. □

Remark 6. *By a parallel argument we can show that the expected number of points in the same hash bucket as q that are at distance at most r from q is at least* $(p_2)^k$. *This relies on the promise that there is at least one such point (see definition of Simplified ANN).*

We want to have only one far point with the same hash as q in expectation. To achieve this we set $k = -\frac{log(n)}{log(p_2)}$. Algorithm 1 satisfies goal 1 but not goal 2. Although $(p_1)^k > (p_2)^k$ it is possible that $(p_1)^k < n(p_2)^k$. Therefore, by our choice of k we may have reduced the expected number of close points in the bin to be less than 1. To increase this probability we repeat the above algorithm L times.

Algorithm 2 (below) takes $O(Ln)$ space to store all hash-values and hash-functions. The algorithm terminates as soon as it finds a point sufficiently close to q. By our choice of k, for every t in expectation we will have to check 1 point. Therefore, the expected run time is $O(L)$. As discussed in Remark 6, the expected number of close points found for every t is $(p_1)^k$. Therefore, the expected number of points across all iterations is $L(p_1)^k$. We want this expectation to be 1. Therefore, we set

$$L = (p_1)^{-k} = (p_1)^{\frac{log(n)}{log(p_2)}} = n^{\frac{log(p_1)}{log(p_2)}} = n^{\rho} \text{ where } \rho = \frac{log(p_1)}{log(p_2)}$$

Note that since $1 > p_1 > p_2 > 0$, we have $0 < \rho < 1$. Therefore, this algorithm takes space $O(n^{1+\rho})$ and runs in time $O(n^{\rho})$ for every query q. This run time excludes the prework of hashing all of the points.

Remark 7. *If* $p_1 = p_2$ *then* $\rho = 1$ *so space is* $O(n^2)$ *and run time* $O(n)$. *This is why we require that* p_1 *is strictly greater than* p_2.

Since Algorithm 2 runs in $o(n)$ time, we have shown that Locality Sensitive Hashing allows us to solve the Approximate Nearest Neighbor Problem efficiently for high dimension.

Algorithm 2

for $t = 1$ to L **do**
  Sample $f_1^t, ..., f_k^t$ uniformly and independently from LSH family F
  for x_i **do**
    map x_i to $(f_1^t(x_i), ..., f_k^t(x_i))$ and hash this value (this hash is not locality sensitive)
  end for
end for

for query q **do**
  for $t = 1$ to L **do**
    calculate $(f_1^t(q), ..., f_k^t(q))$ and corresponding hash value
    for every point x_i in the same hash bucket as q **do**
      if $||x_i - q|| \leq cr$ **then**
        return x_i
      end if
    end for
  end for
end for

4 Summary

In this lecture, we discussed the Nearest Neighbor Problem and the related Approximate Nearest Neighbor Problem. Furthermore, we introduced the concept of Locality Sensitive Hashing and showed that Locality Sensitive Hashing allows us to solve Approximate Nearest Neighbor efficiently for high dimensions.