A Simple, Coarse Vector Nearest Neighbors Algorithm

S. Muthukrishnan
muthu@cs.rutgers.edu

1 Introduction

This note presents a simple algorithm for finding the nearest neighbors (NNs) in vectors; this algorithm is particularly simple to implement, but the guarantees it provides are somewhat coarser than best known NN algorithms that tend to be computationally more expensive. In addition, this algorithm is dynamic and so it may be applicable to online tasks in mining massive data streams.

The problem is as follows. We are given a set S of n binary vectors of length ℓ each. We are allowed to preprocess S. Given a query vector q, the problem is to return $t \in S$ such that $d(q,t) \leq d(q,s)\ \forall s \in S$. Here, $d(x,y)$ is some distance function between two vectors. We call t the vector nearest neighbor (NN) of query q.

This problem faces what is known as the "dimensionality curse", i.e., best known algorithms for this problem are caught between two extremes in the worst case: using data structure polynomial in $n\ell$ with query time exponential in ℓ, or using data structure exponential in ℓ with query time near-linear in ℓ. In IR tasks, the dimension ℓ is very large, and the curse is fundamentally limiting.

Progress on this problem has come from the observation that (a) in many applications – IR applications fall in this category – approximate solution is just as useful as the exact optimum, and (b) approximate NNs are easier to compute using dimensionality reduction techniques.

Major results known in computing approximate NNs are as follows:

1. Kleinberg presented subexponential preprocessing time algorithm with near-linear query time, for large ℓ. The NN is a $1+\epsilon$ approximation to the true NN, provably. [2] This was the first-known, significant progress for large ℓ. The result applied to Euclidean distance only.

2. Ostrovski et al, and Indyk et al [4, 3] presented truly polynomial preprocessing with near-linear query time for Hamming as well as Euclidean distances. These results subsume the one above.

3. A new approximate NN algorithm for the Hamming distance by Muthukrishnan et al [1], which is simple, but not as accurate as the result above. It is also dynamic, and hence, can deal with online situations.

Here, we describe the third result above. We focus on $H(x,y)$, the Hamming distance, ie., the number of positions where x and y differ. Our discussion below applies directly to square of the Euclidean distance, or weighted version where the weight is assigned per vector dimension, by unary encoding of the $t[k]$'s and the weights either exactly or to some suitable discretization.

2 The Algorithm

The algorithm consists of a preprocessing step, followed by query processing.

The preprocessing algorithm is as follows, and proceeds in two steps. All logs are to base 2 unless specified otherwise.

- For any $t \in S$, for $i = 1, \ldots, \log^2 n$, do the following. For $j = 1, \ldots, \log_\beta \ell$, pick β^j locations $r_1, \ldots, r_{\beta^j}$ in $1, \ldots, n$ uniformly, independently and randomly. We compute

$$m_{i,j}(t) = XOR_{k=1,\ldots,\beta^j} t[r_k]$$

  Define $M_i(t) =< m_{i,1}(t) \cdots m_{i,\log_\beta \ell}(t) >$.[1]

- Construct a *trie* of the vectors $M_i(t)$ for a given i and all t's. Denote this by T_i. Construct all such T_i's, $i = 1, \ldots, \log^2 n$.

Query processing proceeds in two steps.

- Given q, we compute $m_{i,j}(q)$ for all i and j. Consider $M_i(q) =< m_{i,1}(q) \cdots m_{i,\log_\beta \ell}(q) >$. Walk down the trie T_i using $M_i(q)$ until no path exists, and pick any leaf t in the subtree where search ends. We will denote this as p_i. (This will represent the longest common prefix p_i between $M_i(q)$ and $M_i(t)$ for all $t \in S$.)

- Compute the Hamming distance $H(q, p_i)$ for $i = 1, \ldots, \log^2 n$ and pick any p_i with the smallest such H. This distance can be calculated brute force by comparing each item in q with corresponding item in p_i or by estimating from the M_i's or by other means by, say, sampling. The bruteforce solution is recommended.

The solution that is returned above is provably approximate. See [1] for details.

The algorithm above is simple because it involves (a) mapping vectors to small dimensions using XOR function, (b) building tries and (c) finding prefixes in the tries. It is easy to make *dynamic*, ie., insert new vectors into S or delete vectors from S. This will lead to inserting and deleting M_i's into various T_i's which is straightforward. This may be of use in the online scenario's.

References

[1] Simple and practical nearest neighbors with block operations. S. Muthukrishnan and S. Sahinalp. CPM 2002.

[2] J. Kleinberg. Two Algorithms for Nearest-Neighbor Search in High Dimensions. STOC 1997: 599-608.

[3] P. Indyk and R. Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. STOC 1998: 604-613.

[4] E. Kushilevitz, R. Ostrovsky, Y. Rabani. Efficient Search for Approximate Nearest Neighbor in High Dimensional Spaces. SIAM J. Comput. 30(2): 457-474 (2000).

[1] This is the dimensionality reduction step, and there are similarities with Ostrovsky et al work.