Fast Approximate Nearest Neighbor Search*

Amit Goyal and Hal Daumé III
Dept. of Computer Science, University of Maryland, College Park, MD 20742
{amit,hal}@umiacs.umd.edu

Many problems in Computer Vision [6, 5] and Natural Language Processing [12, 3] involves finding $l$ nearest neighbors to the query. However, finding exact $l$ nearest neighbors to the query can be time and memory intensive [10, 9]. Hence, in some applications [10, 5] it may be acceptable to return approximate $l$ nearest neighbors. In this work, we propose a novel fast approximate nearest neighbor search algorithm and apply it to finding $l$ similar words with respect to a query word.

1 Preprocessing for Fast Approximate Nearest Neighbor Search

First, for every word “z”, we assume that we are given a context vector $((c_1, v_1); (c_2, v_2) \ldots (c_d, v_d))$ of size $d$ where $c_d$ denotes the context and $v_d$ denotes the Pointwise Mutual Information (PMI) (strength of association) between the context $c_d$ and the word “z” (vocabulary of $Z$ words). The context can be lexical, predicate argument structure and dependency units that co-occur with the word “z”. For each word, we use hashing to project the context vectors onto $k$ directions. We use $k$ pairwise independent hash functions that maps each of the $d$ context $(c_d)$ dimensions onto $\beta_{d,k} \in \{-1, +1\}$; and compute inner product between $\beta_{d,k}$ and $v_d$. Next, $\forall k, \sum_{i} \beta_{d,k} v_i$ returns the $k$ random projections for each word “z”. We store the $k$ random projections for all words (mapped to integers) as a matrix $A$ of size of $k \times Z$.

The mechanism described above generates random projections by implicitly creating a random projection matrix from a set of $\{-1, +1\}$. This idea of creating implicit random projection matrix is motivated by the work on stable random projections [7] and online Locality Sensitive Hash [11]. The idea of generating random projections from the set $\{-1, +1\}$ was originally proposed by [1], then extended by [8].

For fast approximate search, we propose a novel approach, which involves two pre-processing steps:

First pre-processing step of fast approximate search is to create a binary matrix $B$ using matrix $A$ by taking sign of each of the entries of the matrix $A$. If $A(i,j) \geq 0$, then $B(i,j) = 1$; else $B(i,j) = 0$. This binarization creates Locality Sensitive Hash (LSH) function that preserves the cosine similarity between every pair of word vectors. This idea was first proposed by Charikar [2] and used in NLP for large-scale noun clustering [10]. However, in large-scale noun clustering work, they had to store the random projection matrix of size $D \times k$; where $D$ denotes the number of all unique contexts (which is generally large and $D >> Z$) and in this paper, we do not explicitly require to store a random projection matrix.

We then pre-process the matrix $A$. First for matrix $A$, we pair the words $1 \cdots Z$ and their random projection values as shown in first matrix in Fig. 1. Second, we sort the elements of each row of matrix $A$ by their random projection values from smallest to largest (shown in second matrix in Fig. 1). The sorting step takes $O(Z \log Z)$ time (We can assume $k$ to be a constant). The sorting operation puts all the nearest neighbor words (for each $k$ independent projections) next to each other. After sorting the matrix $A$, we throw away the projection values leaving only the words (third matrix in Fig. 1). To search a word in matrix $A$ in constant time, we create another matrix $C$ of size $(k \times Z)$ that is the fourth matrix from Fig. 1. Matrix $C$ maps the words $1 \cdots Z$ to their sorted position in the matrix $A$ (third matrix from Fig. 1) for each $k$.

2 Fast Approximate Search

After the pre-processing is done, fast approximate search is very simple and fast. To search a word “z”, first, we can look up matrix $C$ to locate the $k$ positions where “z” is stored in matrix $A$. This can be done in constant time (Again assuming $k$ to be a constant.). Once, we find “z” in each row, we can select $b$ (beam parameter) neighbors ($b/2$ neighbors from left and $b/2$ neighbors from right of the query word.) for all the $k$ rows. This can be done in

---

*Topic: other applications and Preference: oral
In their work, they used the search algorithm PLEB (Point Location in Equal Balls) first proposed by Indyk and Motwani [4] and further improved by Charikar [2]. The improved PLEB algorithm involves generating permutations of the binary matrix \( B \), and Ravichandran et al. [10] used \( p = 1000 \) random permutations in their work, which means storing \( p = 1000 \) copies of matrix \( B \). In our work, we only use one copy of \( B \) and along with that we store \( A \) and \( C \) to find potential nearest neighbors. To evaluate the quality of our approximate search algorithm, we fix parameters \( k = 3000 \) and \( b = 40 \). We are given a context vectors for 106,733 words of size \( d = 1000 \) and using our approximate algorithm, we find top 10 neighbors for each word. Table 1 shows the top 10 most similar words for some words found by algorithm.

### References


