Fast random pair divisive construction of kNN graph using generic distance measures

Sami Sieranoja
School of Computing
University of Eastern Finland
sami.sieranoja@uef.fi

Pasi Fränti
School of Computing
University of Eastern Finland
pasi.franti@uef.fi

ABSTRACT

We introduce a fast hierarchical method algorithm to construct an approximate kNN-graph. The method is simple to implement and generic, as it works with any type of data for which a symmetric distance function can be provided. We tested the method with text data of 466,544 English words and an image dataset with one million features. Compared to the O(N^2) brute force, the method achieves 140:1 speed-up ratio, on average, and is about twice as fast as the previous state-of-the-art.

CCS Concepts

• Information systems applications → Data mining → Nearest neighbor searches • Theory of computation → Design and analysis of algorithms → Streaming, sublinear and near linear time algorithms → Nearest neighbor algorithms

Keywords

k-nearest neighbor graph; approximate k-NN.

1. INTRODUCTION

Given a data set of N data objects $X = \{x_1, x_2 ..., x_N\}$, *k-nearest neighbors* are defined as the k other objects that are the closest according to some distance function d. By collecting the k nearest neighbors for every data objects x, we can construct *kNN* graph. In this graph, every object (node) is connected by edges to its k nearest neighbors. The weights of the edges are the corresponding distances.

KNN graph can be used for efficient search structure and for modeling the data. Applications include fast nearest neighbor search [1], density estimation [2], outlier detection [3], and fast agglomerative clustering [4].

A brute force approach to construct *kNN* graph takes O(N^2) time because the need to calculate all pairwise distances. This can be acceptable for smaller datasets but it becomes a bottleneck in analyzing big data. Many exact [5] and approximation methods have therefore been proposed that compromise accuracy for speed [6, 7, 8, 9]. The best methods achieve speed-up factors of about 100:1 with 90% accuracy [9, 10].

Most of the existing methods operate by dividing the data into subsets and using brute force for each subset [6, 8, 9]. One approach is to calculate *principal component axis* (PCA), and to split the data iteratively along this projection [6, 9]. *Locality sensitive hashing* (LSH) with random linear projection was used in [8]. However, these approaches are not generic as the calculation of the projection axis requires the data to be numeric.

The only generic method we are aware of is the *neighborhood propagation* method called *NN-Descent* [7]. It takes a random graph as an input and improves by finding better neighbor candidates by exploring neighbors of the neighbors. It is generic because it can be applied to any type of data as long as a distance function can be provided. Also the LSH method in [8] can be potentially used for other types of data. However, it requires a hash function, which is specific to the data type.

In this paper, we propose a simple but generic approach to perform the splitting. Our method is similar to that of [9], but instead of using PCA, we select a random pair of points, and subdivide the data into groups based on which point is nearest. This process is the repeated several times to update the previous graph. The resulting solutions will also be improved using neighborhood propagation. Besides the speed, the main benefit of the method is that it does not require any numeric computation other than that required by the generic distance function. It is therefore directly applicable to other types of data. As a case study, we apply it for words using *edit distance* and *Dice coefficient*.

We show that the proposed method is competitive and outperforms compared methods. Because the method uses only distance calculations in the splitting stage, its behavior changes based on the distance measure, unlike e.g. methods that use PCA. Therefore, we expect it to adapt better to wider variety of distance measures.

2. RANDOM PAIR SPLITTING

We present next our new algorithm to construct an approximate kNN graph for generic distance measures. The algorithm has two parts. In the first part (Algorithm 1), we generate a crude approximation of the graph by subdividing the dataset until each subset reaches a given maximum size W. In the second part (Algorithm 2), this approximation is iteratively fine-tuned by combining Algorithm 1 with the NN-descent method.

2.1 Divide-and-conquer stage

The dividing is done simply by selecting two random points (a and b). The dataset is then divided into two subsets (A and B) based on the generic distance function. For every point x, we calculate its distance both to a and b, and assign it to the subset it is closer to. The dividing is recursively continued until the subset reaches a maximum allowed size W. In this paper, we use the

ICBDC '18, April 28–30, 2018, Shenzhen, China
© 2018 Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6426-3/18/04…$15.00
https://doi.org/10.1145/3220199.3220215

value W=2.5·k. Pseudo code is shown below, and an example given in Figure 1.

```
Algorithm 1: RandomPairDivision(X,kNN,Size)
{
IF size(X) < Size THEN
    BruteForce(X,kNN);
    return;
ELSE
    a = X[random(1,N)];
    b = X[random(1,N)];
    FOR i=1 TO N DO
        IF d(x,a) < d(x,b) THEN
            A = A ∪ x
        ELSE
            B = B ∪ x;
    RandomPairDivision(A);
    RandomPairDivision(B);
}
```

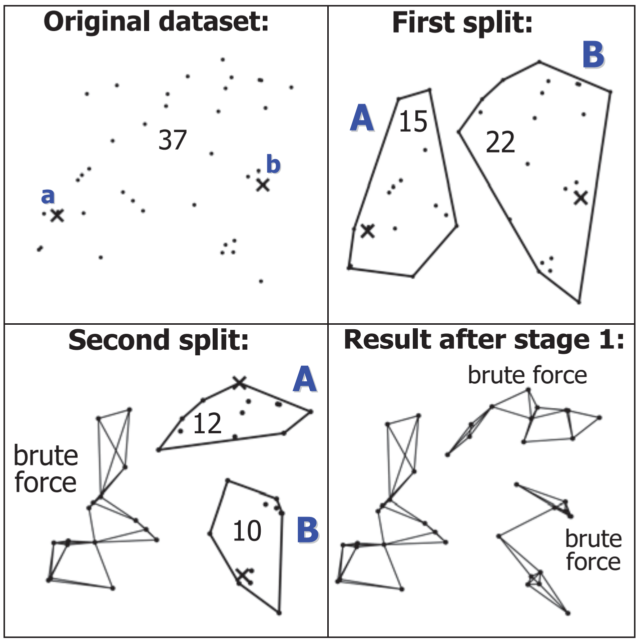

Figure 1. Example of the dividing stage.

2.2 Repeating Divide-and-conquer

This divisive approximation step is then tuned. We combine it with the NN-descent as follows. First, we repeat the divisive algorithm several times. However, instead of creating a new approximation from scratch, we update the previously obtained graph by taking the k closest neighbors for every object x, see Fig. 2 for an example. There are maximum 2k candidates: k from the previous solution, and k new ones. Some of them might be the same but it does not matter.

Time bottleneck of the algorithm is the brute force which requires O(W^2). Assuming that all subsets are exactly of size W, there will be N/W subsets. The total time complexity of single iteration of the algorithm is then O($N/W·W^2$) = O(NW). Using W=2.5·k, and assuming that the number of repeats (r) is a small constant, this leads to linear O(rkN) time algorithm.

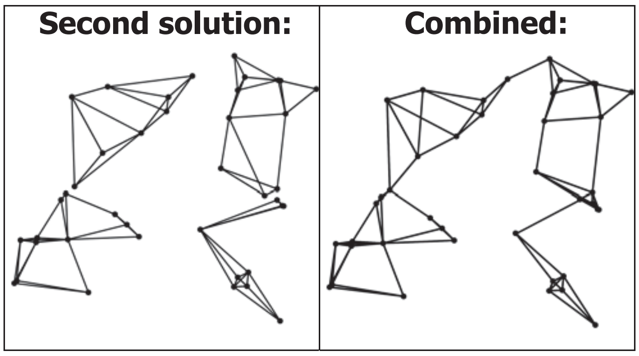

Figure 2. Effect of two repeats of the divisive algorithm.

2.3 Fine-tuning by NN-descent

In the second part, we improve the result of the divisive part by using neighborhood propagation [7]. In practice, we interleave the two parts so that, first, we iterate the divisive part (Algorithm 1) until the portion of changed edges in last iteration drops below 10%.

After that, we start to involve NN-Descent algorithm in the process. Basically the second part works the same as repeating the divisive algorithm so that the new approximation merely updates the previous solution. In other words, whenever closer neighbors are found, the graph is updated accordingly. The only difference is that the result is tuned by one iteration of NN-descent algorithm.

The overall process is summarized in Algorithm 2. In this paper, we fix the stopping criterion to 0.01 %. Source code (C++) can be found in: http://www.uef.fi/web/machine-learning/software

```
Algorithm 2: RandomPairNNDES(X,kNN,W)
{
REPEAT
    RandomPairDivision(X,kNN,W);
    delta = Changes(kNN);
UNTIL delta < 10%
REPEAT
    RandomPairDivision(X,kNN,W);
    NNDES(X,kNN);
    delta = Changes(kNN);
UNTIL delta < StopCriterion;
}
```

2.4 Distance function

Any distance function should be applicable with the method. In this paper, we consider three distance measures: *Euclidean*, *Edit distance* [11] and *Dice* [12]. Euclidean distance (L_2) is the most widely used for numerical data:

$$d(x,y) = \sqrt{\sum_{i=1}^{D} \|x_i - y_i\|^2} \quad (1)$$

Most commonly used measure for words is *edit distance*. It calculates the minimum number of edit operations needed to transform a string x to string y. The edit operations include insertion, deletion, and substitution. The best match can be found by dynamic programming [13].

However, measuring similarity of short text like document titles, character-level measures might not capture all the essential information [14]. Word-level measures have been used in this case. We consider one such approach known as *Dice coefficient* [12], which works as follows. Each string is represented as a set of bigrams. For example, word *string* would become {*st*, *tr*, *ri*, *in*, *ng*}. Similarity is measured as the size of the intersection divided by the average cardinality of the two sets. We convert the similarity to distance for convenience:

$$d_{Dice}(x,y) = 1 - \frac{2 \cdot |x \cap y|}{|x| + |y|} \quad (3)$$

An example of kNN graph for words is illustrated in Figure 3 using edit distance. Here only the neighbors of the three words are shown. All distances in the graph are 2, except those marked by number 1.

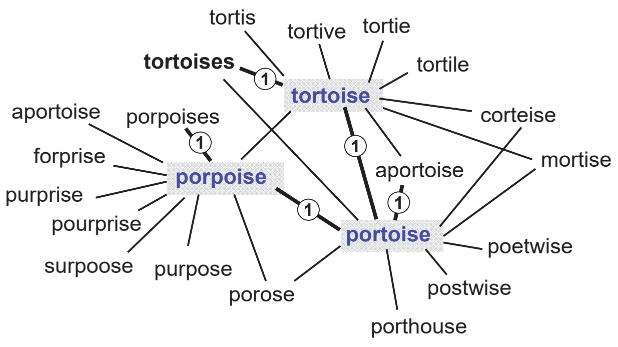

Figure 3. Example of nearest neighbors for the words *porpoise*, *tortoise* and *portoise*. Here *k*=10 is used.

3. EXPERIMENTS

3.1 Datasets

We test the proposed algorithm with the four datasets summarized in Table 1. Sift1M dataset contains 1 million Scale-invariant feature transform (SIFT) image feature vectors of dimensionality 128. Words dataset contains 466,544 English words. Their lengths vary from 1 to 45; average string length is 9.4 characters. A Birkbeck corpus contains 36,133 misspellings of 6,136 words, which constitute 39,030 unique strings. The string lengths vary from 1 to 18; and is 7.8 characters on average. We use Euclidean distance for Sift1M, and edit distance and Dice for Words and Birkbeck. The number of neighbors we fix to *k*=20 in all tests.

Table 1. Datasets used in the experiments.

Data:	Size:	Dim:	Distance:	Type:
Sift1M[1]	1M	128	Euclidean	Image
Birch2[2]	100,000	2	Euclidean	Synthetic
Words[3]	466,544	9.4 char	{Edit., Dice}	Strings
Birkbeck[4]	39,030	7.8 char	{Edit., Dice}	Strings

3.2 Measurements

To evaluate the goodness of the resulting graph we calculate the sum of the distances (total weight) in the graph. We measure the quality of the approximation graph as the ratio of its total weight relative to the total weight of the exact kNN graph. We call this measure as *gap*:

$$Gap = \frac{\sum_i |kNN(i)|}{\sum_i |kNN_{opt}(i)|} - 1$$

where *kNN* refers to the resulting graph, and kNN_{opt} to the exact (optimal) graph.

Another measure commonly used is *Recall* [6] but it is not suitable for string data because there are often many strings within the same edit distance. For example, when constructing graph with *k*=20 neighbors, the 20th nearest may have edit distance of 5, but there can be 30 other strings with the same distance. So, even brute force can generate two different *kNN* graphs depending on the order of distance calculations.

3.3 Results

The results show that the proposed RP-div provides speed-up factors varying from 19:1 to 323:1, compared to standard $O(N^2)$ time brute force. With all data sets, it is also roughly 50% faster than the state-of-the-art.

Besides the faster speed, the method also provided better (or similar to) quality than the NNDES method. In the case of Words data set and Dice similarity, NNDES converges to 2% in 223 seconds whereas RP-Div is able to reach 1% in 100 seconds and 0.1% in 247 seconds.

The effect of the data size is demonstrated further in Figure 4. The results show that the bigger the data, the more significant the speed-up is. For example, with *N*=10000 RP-Div obtains speed-up of 63:1 but with full size *N*=100,000 already 888:1 compared to brute force. The benefit compared to NN-Descent also increases with *N* reaching 5:1 when *N*=100,000.

Table 2. Processing time (s) to reach quality level of gap 1%.

Data:	Method:			Speed-up factor:
	RP-div	NNDES	Brute force	
Sift1M (L_2)	351	726	113,701	323:1
Birch2 (L_2)	2	6	290	145:1
Words (edit)	808	1288	65,837	81:1
Birkbeck (edit)	30	61	573	19:1
Words (Dice)	100	-	9444	94:1
Birkbeck (Dice)	5	10	79	15:1

[1] http://corpus-texmex.irisa.fr/

[2] http://cs.uef.fi/sipu/datasets/

[3] https://github.com/dwyl/english-words

[4] https://www.dcs.bbk.ac.uk/~ROGER/corpora.html

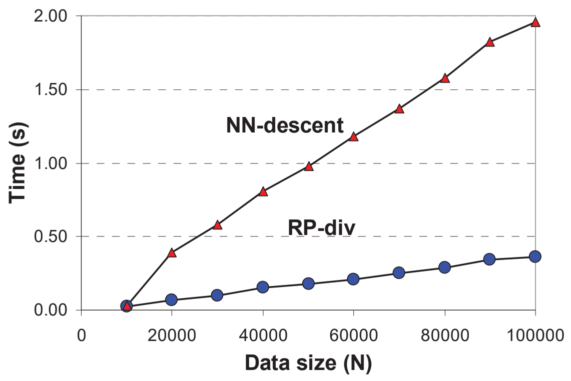

Figure 4. Effect of the data size (subsets of Birch2). Results are to reach quality level of gap 10%.

4. CONCLUSIONS

We proposed a fast approximate *kNN* graph construction method called RP-div. Its main benefit is that it can operate with any symmetric distance measures. Our experiments show that it reaches reasonably accurate graphs (1% gap) even for very large (~0.5M) string data. The method provides from 19:1 to 323:1 speed-up factors compared to standard O(N^2) time brute force, and outperforms the previous state-of-the-art (NNDES) method by about 50%.

We intend to continue this work by using the graph for agglomerative clustering [4] of string data, to allow O(N log(N)) clustering of strings. The method would also allow estimating density using kNN graph and using this information for fast density based clustering of strings.

5. REFERENCES

[1] Hajebi K., Abbasi-Yadkori Y., Shahbazi H., and Zhang H. 2011. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2, 1312-1317.

[2] Yu J., and Kim S.B. 2016. A density-based noisy graph partitioning algorithm. *Neurocomputing*, 175 473–491.

[3] Hautamäki V., Kärkkäinen I., and Fränti P. 2004. Outlier detection using k-nearest neighbour graph. In *IAPR Int. Conf. on Pattern Recognition (ICPR'04)*, Vol. 3, 430–433.

[4] Fränti P., Virmajoki O., and Hautamäki V. 2006. Fast agglomerative clustering using a k-nearest neighbor graph, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 28, 1875–1881.

[5] Connor M. and Kumar P. 2010. Fast construction of k-nearest neighbor graphs for point clouds. *IEEE Trans. on Visualization and Computer Graphics*, 16, 599–608.

[6] Chen J., Fang H-r., and Saad Y. 2009. Fast approximate k NN graph construction for high dimensional data via recursive Lanczos bisection. *The Journal of Machine Learning Research*. 10, 1989–2012.

[7] Dong W., Moses C., Li K. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In *ACM International Conference on World wide w*eb. ACM, 577–586.

[8] Zhang Y-M., Huang K., Geng G., and Liu C-L. 2013. Fast kNN graph construction with locality sensitive hashing. In *Machine Learning and Knowledge Discovery in Databases*. Springer, 660–674.

[9] Wang J., Zeng G., Tu Z., Gan R., and Li S. 2012. Scalable k-NN graph construction for visual descriptors. In *IEEE Conf. on Computer Vision and Pattern Recognition* (CVPR), 1106–1113.

[10] Sieranoja S. and Fränti P. 2018. High-dimensional kNN-graph construction using z-order curve. *ACM Journal of Experimental Algorithmics*. (in review)

[11] Levenshtein V.I. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, 10 (8), 707-710.

[12] Brew C., McKelvie, D. 1996. Word-pair extraction for lexicography. In *International Conference on New Methods in Language Processing*, 45-55.

[13] Jimenez S., Becerra C., Gelbukh A. and Gonzalez F. 2009. Generalized mongue-elkan method for approximate text string comparison. In *Int. Conf. on Intelligent Text Processing and Computational Linguistics*, 559-570.

[14] Gali N., Mariescu-Istodor R., Fränti P. 2016. Similarity measures for title matching. In *IAPR Int. Conf. on Pattern Recognition (ICPR'04)*, 1548-1553.