ESANN 2017 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 26-28 April 2017, i6doc.com publ., ISBN 978-287587039-1. Available from http://www.i6doc.com/en/.

Scalable approximate k-NN Graph construction based on Locality Sensitive Hashing

Carlos Eiras-Franco [1], Leslie Kanthan[2], Amparo Alonso-Betanzos[1] and David Martínez-Rego[2] *

1- Department of Computer Science - University of Corunna.
2- Department of Maths and Computer Science - University College London.

Abstract. Nearest neighbours graphs are a pervasive basic construct in areas such as Data mining, Machine Learning and Information Retrieval. Among them, the k Nearest Neighbours Graph (kNNG), is probably the most studied of all. Unfortunately, its naïve construction is in $\mathcal{O}(n^2)$ for n data points, which becomes a quagmire when scaling to Big Data. However sub-quadratic construction of kNNG remains an open question. This paper explores an adaptive algorithm based on Locality Sensitive Hashing which presents good performance on distributed architectures.

1 Introduction

Many problems in areas such as Information Retrieval, Data Mining, and Machine Learning rely on the construction of graphs that encode a notion of similarity between elements in the problem under study [1, 2]. These similarity graphs fuel further tasks such as classification, clustering, etc. K-Nearest Neighbour Graphs (kNNG) are among the most commonly used, due to their (a) usefulness and robustness in real practice and (b) intuitive meaning. A kNNG is a directed graph where for n data points, $X = \mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_n$, edges $(\mathbf{x}_i, \mathbf{x}_j)$ indicate that $\mathbf{x}_j$ is amongst the k most similar elements to the point $\mathbf{x}_i$ under a specified similarity measure $S(\mathbf{x}_i, \mathbf{x}_j)$. The simplicity of this idea conflicts with its computational cost. Obtaining the kNNG requires $n(n-1)/2$ comparisons, and thus implies $\mathcal{O}(n^2)$ time complexity. Algorithms that can build an exact solution when the dimensionality of the input space is small have been proposed in the literature [3]. In addition, some efficient metric specific algorithms have also been studied [4]. Nevertheless, in order to scale to a reasonable input dimension, reduce complexity and cope with more general metrics, only approximate solutions can be built. Approximate kNNG algorithms attempt to balance computational complexity with accuracy of the built approximate graph w.r.t its exact counterpart. Such algorithms have been proposed using different design principles from divide-and-conquer [5, 6] to local search [7, 8]. In [9] a generic strategy for approximating kNNG under any similarity is devised.

Locality Sensitive Hashing (LSH) [10] is a technique originally devised to answer individual similarity queries for any point in sublinear time by constructing a data structure with the base dataset to be queried. The optimal method to

*This research has been supported in part by the Spanish Ministerio de Economía y Competitividad (project TIN 2015-65069-C2-1-R), partially funded by FEDER funds of the EU and by the Consellería de Industria of the Xunta de Galicia (project GRC2014/035).

exploit this structure and the principles of LSH for kNNG remain an open problem. A strategy for kNNG construction on the LSH data structure would be a good fit for parallel implementation, and would induce the scaling of various algorithms that are currently limited by kNNG. LSH would first partition data into buckets containing points that have a high probability of being similar, then partial graphs would be computed by brute force for each bucket and finally merged in order to produce the final approximate kNNG. So far, and to the authors knowledge, only one work [9] has explored this possibility. Our algorithm follows an alternative strategy which, instead of focusing on obtaining equally sized groups, reduces the number of irrelevant comparisons while maintaining accuracy. Additionally, both sub-graph construction and merging occur in parallel for the different buckets thanks to the implementation in the Spark [11] platform. Experimental results using the real datasets validate our proposal.

2 kNNG using Locality Sentitive Hashing

We start by introducing the principles of LSH that help build the proposed divide and conquer strategy and then detail the steps of the kNNG construction proposed in this work. LSH [10] is based on the idea that if two points are similar, then after a projection, they will still share that same notion of similarity. It is defined, in our case, by the usage of a distance function. A locality sensitive hashing function $h(x)$ guarantees with a fixed probability that similar items will hash to the same hash value. The LSH algorithm uses such hash functions to group similar data together. In order to achieve that, it firstly generates a hash code for each point in the dataset. The points are then grouped into buckets according to their hash value. When we need to query a point, in order to find its k nearest neighbours, the algorithm converts the query point into a hash code using that same hash function and similar points are subsequently searched inside the colliding buckets. The LSH algorithm itself depends upon the existence of locality sensitive hashing functions for a given distance. Such a family of hash functions H is called $(r; cr; P_1; P_2)$-sensitive if for any $\mathbf{p}, \mathbf{q} \in \Re^d$:

$$\|\mathbf{p} - \mathbf{q}\| \le r \longrightarrow Pr(h(\mathbf{p}) = h(\mathbf{q})) \ge P_1 \quad (1)$$

$$\|\mathbf{p} - \mathbf{q}\| > cr \longrightarrow Pr(h(\mathbf{p}) = h(\mathbf{q})) \le P_2 \quad (2)$$

More specifically, given the points $\mathbf{p}, \mathbf{q} \in \Re^d$, if $\|\mathbf{p} - \mathbf{q}\| \le r$ then the points are considered close and a randomised hash function $h(\mathbf{x}) \in H$ will produce a collision with probability at least P_1; albeit if $\|\mathbf{p} - \mathbf{q}\| > cr$, the probability of collision under a random hash function will be small, i.e. $Pr(h(\mathbf{p}) = h(\mathbf{q})) \le P_2$. Note that the first property tries to assure correctness with high probability, while the second property helps to prune useless distance computations. The higher P_1 and the lower P_2, the more effective the algorithm will be and this depends on the family H for a given distance $\|\cdot\|$. Clearly, $P_1 > P_2$ for a hashing family to be meaningful. If the difference between P_1 and P_2 is sufficiently large, by composing longer codes using several random functions gathered from H, the probability of collision is very low for points further away and maintained for closer points. Furthermore, a probability union bound argument proves that the use of several tables can boost the accuracy of the search (see details in [10]).

2.1 Implementing the Algorithm

We propose Variable Radius LSH (VRLSH)[1], a LSH based algorithm that explores buckets that progressively increase in size while simplifying the dataset at each step. All points of the dataset are hashed using a LS-Hasher. Points with the same hash value are then grouped into subsets (buckets) and the corresponding sub-graph for each subset is computed by brute-force. Subsequently, the resulting sub-graphs are merged to obtain an approximate graph. After performing this, a novel method is applied, consisting of removing from the dataset those points that have been involved in a fixed number of pair-wise computations (MAX_COMPS in Algorithm 1), and the process is repeated until all points are removed from the dataset. This loop is described in line 2 of Algorithm 1. Another improvement in the algorithm is that whilst the number of points is monotonically reduced for each iteration of the algorithm, the size of the buckets is increased. This in turn aids in increasing the accuracy of the final kNNG. The start radius can be set by the user, and is by default set to 0.1, which empirically showed to be inferior than the distance of the two nearest neighbors for many datasets. Finally, to address the rare cases when a single point is left without its k nearest neighbors, a search of the neighbors of its neighbors is performed if needed from line 9 on. This algorithm has been designed to be suitable for distributed computation and to be capable of handling large datasets. The addition of a registry that keeps track of the pair-wise computations already performed would vastly reduce the execution time, but it would have a great impact in the memory requirements for large datasets and could hinder the suitability for distributed computation, so we decided against including such a registry. The Spark implementation proves the adequacy to the distributed paradigm.

3 Experimental design and Results

To test the adequacy of our approach, three real-world web datasets were used (see Table 1) to benchmark our results against those already obtained in [7]. This is because NN-Descent is the preferred algorithm for obtaining kNNG [7].

We compare the accuracy of the obtained graphs, defined as the ratio of common edges with the real graph to the total number of edges. We also investigate the total pair-wise comparisons performed by each algorithm with the scan rate measure, defined as the ratio of comparisons performed to the number of comparisons that the brute algorithm would use for that dataset, which is $N(N-1)/2$. The results are represented in Figure 1. Our experiments showed that when the number of neighbors queried is small, the accuracy of the graph obtained by VRLSH is higher. For $K = 2$ the results range from 0.88–0.92, as opposed to ~ 0 obtained by NN-Descent. As K grows, the accuracy of the

[1]Spark implementation available for download at https://github.com/eirasf/KNiNe

Algorithm 1: Pseudo-code for VRLSH algorithm

```
    Input: S,k ←Set of points, Number of neighbors to be obtained
    Output: G ← Graph containing the k nearest neighbors for each point
 1  $G \leftarrow \emptyset$ , $originalS \leftarrow S$, $radius \leftarrow RADIUS\_START$
 2  while $|S| > 1$ and $|buckets| > 1$ do
 3      $hashElems \leftarrow LShash(S, radius)$
 4      $buckets \leftarrow hashElems.groupByHash()$
 5      foreach $b$ in $buckets$ do
 6          if $(b.size > 1)$ then $G \leftarrow G \cup bruteForce(b.elems, k)$ end
        end
 7      $S \leftarrow S - G.getNodesWithAtLeastNComparisons(MAX\_COMPS)$
 8      $radius \leftarrow radius * 2$
    end
 9  if $|S| > 1$ then
10      $G \leftarrow G \cup singleElementNeighborDescent(S[0], G)$
11      $S \leftarrow S - G.getNodesWithAtLeastNComparisons(MAX\_COMPS)$
    end
12  if $|S| > 1$ then
13      foreach $p$ in $S$ do
            $G \leftarrow G \cup bruteForceElement(p, originalS, k)$
        end
    end
```

Dataset	Size	Dimensionality
Audio	54387	192
Shape	28775	544
Isolet	6238	617

Table 1: Datasets used in the study.

graph rendered by VRLSH decreases, while that of the graph calculated by NN-Descent increases. For instance, when $K = 32$ the accuracy of NN-Descent exceeds 0.99 in all cases, while VRLSH obtains values ranging from 0.57 to 0.67. Nevertheless, NN-Descent obtains these superior results at expense of greatly increasing the number of pair-wise distance calculations performed, while this number remains constant for VRLSH. Notably when the scan rate approaches 1 the algorithm is ineffective, since the brute force algorithm has a scan rate of 1. Moreover, NN-Descent is dependant on a registry of the pair-wise distances that have already been computed in order to avoid repeating calculations and therefore maintain the low cost. This results in a large memory requirement that thwarts its use for big datasets . Without the use of this registry the scan rate of NN-Descent grows much faster. In order to allow the use of large datasets, such a registry is not present in VRLSH. To increase the accuracy of the graph obtained with VRLSH, a single step of neighbor descent can be performed on

ESANN 2017 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 26-28 April 2017, i6doc.com publ., ISBN 978-287587039-1. Available from http://www.i6doc.com/en/.

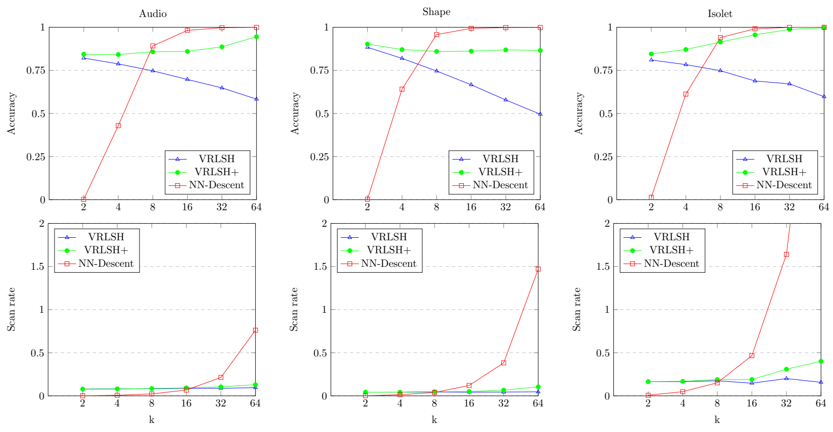

Fig. 1: Accuracy vs number of neighbors plots for Audio, Shape and Isolet datasets and Scan rate vs number of neighbors plots for those same datasets.

the calculated graph. The results of this enhancement, represented in Figure 1 with the name VRLSH+, are increased accuracy while ensuring the cost is low. In the aforementioned case of $K = 32$ the accuracy grows from $[0.57 - 0.67]$ to $[0.87 - 0.99]$ with this single descent step, and the scan rate is barely impacted, resulting in values ranging from 0.07 to 0.3, depending on the dataset as opposed to 0.21 to 1.64 required by NN-Descent, even with the use of a registry.

4 Conclusions and Future Work

We have proposed VRLSH, which has high accuracy and low scan rate irrespective of the number of neighbors. The resulting graph can be refined with additional neighbor descent steps, driving the accuracy up and still keeping the scan rate below its brute force method alternative. We have also provided a distributed implementation of this algorithm in Apache Spark and tested it against the preferred method for kNNG calculation. For future work, we will explore the use of Bloom filters [12] to keep track of the pair-wise calculations without having a large impact on memory. This could be applied to VRLSH for more explorations of the search space or its additional neighbour descent step, enabling the computation of additional descent steps without inflating the scan rate. Optimal hasher parameters that best suit the given dataset is a crucial part of any LSH algorithm. Different parameters offer very disparate results for the same dataset, but an estimating procedure needs to be provided since it is mandatory that the algorithm performs well with little or no configuration. Another avenue would be to explore the improvement of the heuristic we employed to calculate the hasher parameters, by studying a small sample of the dataset. Additionally, the requirement of a large number of hashes for each example constrains computation time and memory, so it would be interesting to investigate the effect of the aforementioned changes in driving the number of required hashes down.

References

[1] B. V. Dasarathy. *Nearest-neighbor approaches. Handbook of Data Mining and Knowledge Discovery, W. Klosgen, J. M. Zytkow, and J. Zyt*, pages 88-298. Oxford University Press, 2002.

[2] J. Sankaranarayanan, H. Samet, and A. Varshney. A fast all nearest neighbor algorithm for applications involving large point-clouds, *Computers and Graphics*, 31(2):157-174, 2007.

[3] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18: 509–517, 1975.

[4] D. C. Anastasiu, G. Karypis. L2Knng:Fast Exact K-Nearest neighbor Graph construction with L2-norm prunning. *Proceedings of the 24th ACM International Conference on Information and Knowledge Management*, 2015.

[5] J. Chen, H. Fang, Y. Saad. Fast approximate kNN graph construction for high dimensional data via recursive Lanczos Bisection. ,*Journal of Machine Learning Research*, 10:1989–2012, 2009.

[6] J. Wang, J. Wang, G. Zeng, Z. Tu, R. Gan and S. Li, Scalable k-NN graph construction for visual descriptors *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (CVPR 2012), 2012.

[7] W.Dong, M. Charikar, and K. Li, Efficient K-Nearest Neighbor Graph Construction for Generic Similarity Measures. *Proceedings of the International World Wide Web Conference Committee (IW3C2)* (WWW 2011) 2011.

[8] Y. Park, S. Park, S. Lee, W. Jung.Scalable K-Nearest Neighbor Graph Construction Based on Greedy Filtering.*Proceedings of the 22th International Conference on World Wide Web 2013 (WWW 2013)*, 2013.

[9] Y.M. Zhang, K. Huang, G. Geng and C.L. Liu. Fast k-NN graph construction with Locality Sensitive Hashing. *European Conference on Machine Learning (ECML-PKDD2013)*, 2013.

[10] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM* 51(1):117–122, 2008.

[11] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, I. Stoica. *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing.* 9th USENIX conference on Networked Systems Design and Implementation, 2012.

[12] Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 13(7), 422-426.