LITERATURE REVIEW: Performance analysis of k-nearest neighbor algorithms

Heli Alpeshkumar Patel
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
helialpeshkumarpatel@cmail.carleton.ca

October 7, 2022

1 Introduction

The amount of various datasets has significantly increased in recent years. There is a great need for innovative techniques that can transform these enormous quantities of data into useful information automatically. The intriguing, important, and comprehensible patterns that data mining reveals in huge datasets. Important application domains include business intelligence, customer relationship management, e-commerce, the World Wide Web, scientific simulation, and many more.

Higher intelligence is needed to deal with the present expansion of Big Data because standard computer performance cannot analyze this enormous volume of data. High-performance computing (HPC) must be taken into account while designing and developing systems since it can speed up calculation and produce precise results at a lower cost. There are several different types of high-performance computing, including computer clusters, grid computing, cloud computing, graphic processing units (GPU), MIC, and FPGA. One technique for achieving great performance on the GPU level is distributed memory programming with several processors. Parallelizing machine learning algorithms is one technique to use high performance computing to manage Big Data[5].

Classification is one of the core techniques in data mining, involving the training of a model on a dataset containing class labels and using the output to infer the class of unlabeled objects. The K-Nearest Neighbors (KNN) technique is one of the most popular machine learning algorithms since it makes categorization simple and effective[8]. KNN is a technique that classifies an item by using the training set's nearest point[8]. Since the KNN method is easy to use, quick to implement, and has a low error rate, it can effectively manage noisy data sets and produce accurate results. These are the reasons why it is utilized in so many diverse areas[5][10].

High-dimensional data sets can be used with the K-Nearest Neighbor technique since its computation is straightforward. However, when the test set, train set, and data dimension are higher than anticipated, the computational complexity will be quite high and the operating time will be extremely long [7]. To adress the problem parallel processing can be utilized and the time complexity can be significantly reduced with the use of parallel processing.

A standardised method of communicating across many computers executing concurrent programs via distributed memory is the message passing interface (MPI)[4]. When a message is sent to an object, parallel process, subroutine, function, or thread, it is usually referred to as "passing a message," and that message is then utilised to start a different process.

The goal of this study is to analyze the performance of KNN and parallized KNN. I am planning to use Message Passing Interface(MPI) to parallelize KNN algorithm. I will use openly available dataset for both implementations to discover the performance differences in terms of speedup and efficiency. Based on the obtained performance results I would like to draw some conclusions and to see which one is better.

2 Literature Review

2.1 K Nearest Neighbour

The K Nearest Neighbor method, which is used for regression and classification, is categorized as supervised learning algorithm[3]. It may be applied to classification or regression issues, although it is most frequently used as a classification technique since it relies on the idea that related points can be discovered close to one another [2].

Based on the following method, the operation of the K-NN can be explained:[1]
Step 1: It is to choose the neighbours' K-number.
Step 2: Determine the K-number of neighbours' Euclidean distances from each other.
Step 3: Using the Euclidean distance estimate, select the K closest neighbours.
step 4: Count the amount of data points in each category among these k neighbours.
Step 5: Place the new data points in the category where the number of neighbours is highest.
Step 6: Our model is complete.

2.1.1 Classification

A class label is chosen for classification issues based on a majority vote, meaning that the label that is most commonly expressed around a particular data point is adopted[2]. Let's think about the case displayed in Figure 1. There are two classes, A and B, and nearby data points will be used to categorise purple coloured dots to identify whether they belong to A or B by calculating their distance from the k closest data points.

2.1.2 Regression

Similar to classification issues, regression problems utilize the same principle; however, in this instance, a classification prediction is made by averaging the classifications of the first k nearest neighbors[2].

2.2 Message Passing Interface(MPI)

In parallel computing, nodes are groups of computers or even individual processing cores on a single computer. A fraction of the entire computing issue is generally worked on by each node in a parallel configuration. To synchronise each parallel node's activities, communicate data across nodes, and exert command and control over the entire parallel

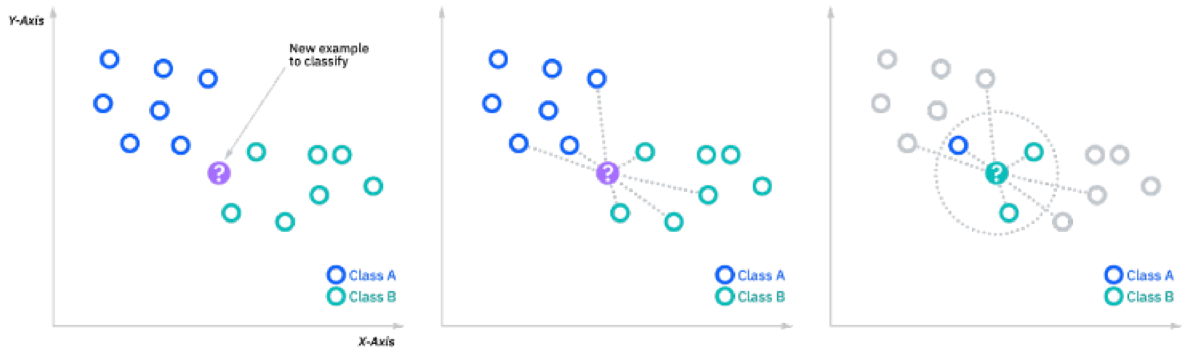

Figure 1: KNN diagram[2]

cluster is the next difficult task. For these responsibilities, the message passing interface specifies a standard set of functions [4]. Benefits of the message passing interface include standardization, portability, speed, and functionality.

2.3 Related work

KNN algorithm is worth researching to improve since it is extremely practical, helpful, and widely utilised in the business. Numerous research have been conducted on the subject of improving the KNN algorithm and attempting to broaden the application of its use in solving big-data issues. P.P. Halkarnikar et al. presented a case study for categorizing a large database, such as the electoral data for the Kolhapur constituency, into age-based categories utilizing the well-liked K-Nearest Neighbor algorithm on multi-core CPUs. Election candidates were categorized as YOUNG, MIDDLE, and OLD. A total of 5 lakh (approximate) votes were broken up into 72 wards with around 7000 votes apiece. They selected 4 of the wards for their investigation. For comparison, authors processed 4 wards independently on a multi-core CPU and on a single CPU. The processing time resulted in a noticeable improvement[7].

Chi Zhang et al. describes that MapReduce is one of the most frequently used frameworks for the parallel and distributed processing of massive data today thanks to the growing support it has received from both business and academia over the past several years[15]. In MapReduce, parallel kNN joins were suggested by Chi Zhang et al. There were suggested exact (H-BRJ) and approximation (H-zkNNJ) methods. In contrast to the precise method's requirement for a quadratic number of reducers, they only needed a linear number of reducers for H-zkNNJ (to the number of blocks from splitting the underlying dataset). Experiments on enormous actual datasets showed that H-zkNNJ performed orders of magnitude better than baseline approaches. In every circumstance that was tested, it also produced exceptional quality.

CUKNN, a parallel KNN implementation built on the CUDA multi-thread paradigm. To increase GPU efficiency, a variety of CUDA optimization approaches were used. CUKNN performs far better and can accelerate computations by up to 15.2X. When the training dataset's dimension and record count are changed, it also demonstrated strong scalability[9].

The primary contribution of Jan Masek et al. in 2015 study is a technique that uses

OpenCL to speed the k-Nearest Neighbor machine learning algorithm. The technique may run concurrently on many GPUs. The updated version of the technique developed by the authors produces excellent results for k neighbours that are less than 10. In comparison to utilising a single core CPU (Intel Core i7-3770, 4.1GHz), they discovered that using very inexpensive hardware (2x NVIDIA GeForce GTX 690) made it feasible to calculate 4 million items (each with 10 characteristics) in 3 minutes as opposed to almost 31 hours. Up to 750x of acceleration was obtained[11].

In 2016, M. M. A. Patwary et al. introduced PANDA, a distributed kd-tree-based KNN implementation that parallelizes the building of kd-trees and querying on enormous datasets of up to 189B particles drawn from several scientific fields. PANDA performs more than an order of magnitude quicker on a single node than the most recent KNN implementations. Additionally, they demonstrated that PANDA scales effectively for all phases of computation up to 50,000 cores in both strong and weak scaling senses. As a consequence, PANDA can create kd-trees for 189 billion particles in about 48 seconds and respond to 19 billion KNN queries in around 12 seconds utilizing 50,000 cores[13].

Another study in 2019, used calls to MPI and GDAL in the C++ development environment on an eight-core CPU to further construct the parallel gwk-NN classifier that the researchers had created throughout the model training and image classification stages. Additionally, the effectiveness of several parallel techniques, including data parallelism, task parallelism, and their combination, dual parallelism, was assessed in terms of running time, speedup, efficiency, as well as their limiting aspects. According to the authors' early experiment, the suggested parallel gwk-NN classifier can increase the effectiveness of high-resolution remotely sensed photos with various types of land cover. Due to parallel overhead, the outcome demonstrates that, the data parallelism technique was successful throughout both the model training and image classification stages, whereas the dual parallelism method could take use of data parallelism and task parallelism in image classification [14].

In 2019 study, the parallel KNN method was used for 3D reconstruction matching. For the research's higher quality photos, a GPU device running the Nvidia CUDA SDK was utilized. In order to reduce access time, the distance calculating result is kept in shared memory and used in the pipeline for real-time feature tracking. The outcome demonstrated that parallel KNN is 10 times quicker than sequential KNN with superior effectiveness and efficiency[6].

A extremely effective and privacy-preserving kNN classification (PkNC) over encrypted data was put out by authors in 2020. The preceding kNN classification required 12.02 to 55.5 minutes to complete using tests with the same dataset, however PkNC only needed 4.16 minutes[12].

References

[1] K-nearest neighbor. https://medium.com/swlh/k-nearest-neighbor-ca2593d7a3c4. (Accessed: October 2022).

[2] Simple understanding and implementation of knn algorithm! https://www.ibm.com/topics/knn. (Accessed: October 2022).

[3] Simple understanding and implementation of knn algorithm! https://www.analyticsvidhya.com/blog/2021/04/simple-understanding-and-implementation-of-knn-algorithm/, 2021. (Accessed: October 2022).

[4] message passing interface. https://www.techtarget.com/searchenterprisedesktop/definition/message-passing-interface-MPI, 2022. (Accessed: October 2022).

[5] Maha A. Alanezi and Abdulla AlQaddoumi. Applying parallel processing to improve the computation speed of k-nearest neighbor algorithm. In *2020 International Conference on Data Analytics for Business and Industry: Way Towards a Sustainable Economy (ICDABI)*, pages 1–6, 2020.

[6] Ming-Wei Cao, Lin Li, Wen-Jun Xie, Wei Jia, Zhi-Han Lv, Li-Ping Zheng, and Xiao-Ping Liu. Parallel k nearest neighbor matching for 3d reconstruction. *IEEE Access*, 7:55248–55260, 2019.

[7] P. P. Halkarnikar, Ananda P. Chougale, H. P. Khandagale, and P. P. Kulkarni. Parallel k-nearest neighbor implementation on multicore processors. In *2012 International Conference on Radar, Communication and Computing (ICRCC)*, pages 221–223, 2012.

[8] S.B. Imandoust and Mohammad Bolandraftar. Application of k-nearest neighbor (knn) approach for predicting economic events theoretical background. *Int J Eng Res Appl*, 3:605–610, 01 2013.

[9] Shenshen Liang, Cheng Wang, Ying Liu, and Liheng Jian. Cuknn: A parallel implementation of k-nearest neighbor on cuda-enabled gpu. In *2009 IEEE Youth Conference on Information, Computing and Telecommunication*, pages 415–418, 2009.

[10] Chun Jie Ma and Zheng Sheng Ding. Improvement of k-nearest neighbor algorithm based on double filtering. In *2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE)*, pages 1567–1570, 2020.

[11] Jan Masek, Radim Burget, Jan Karasek, Vaclav Uher, and Malay Kishore Dutta. Multi-gpu implementation of k-nearest neighbor algorithm. In *2015 38th International Conference on Telecommunications and Signal Processing (TSP)*, pages 764–767, 2015.

[12] Jeongsu Park and Dong Hoon Lee. Parallelly running k-nearest neighbor classification over semantically secure encrypted data in outsourced environments. *IEEE Access*, 8:64617–64633, 2020.

[13] Md. Mostofa Ali Patwary, Nadathur Rajagopalan Satish, Narayanan Sundaram, Jialin Liu, Peter Sadowski, Evan Racah, Suren Byna, Craig Tull, Wahid Bhimji, Prabhat, and Pradeep Dubey. Panda: Extreme scale parallel k-nearest neighbor on distributed architectures. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 494–503, 2016.

[14] Yingxia Pu, Xinyi Zhao, Guangqing Chi, Shuhe Zhao, Jiechen Wang, Zhibin Jin, and Junjun Yin. Design and implementation of a parallel geographically weighted k-nearest neighbor classifier. *Computers Geosciences*, 127:111–122, 2019.

[15] Chi Zhang, Feifei Li, and Jeffrey Jestes. Efficient parallel knn joins for large data in mapreduce. In *Proceedings of the 15th International Conference on Extending Database*

Technology, EDBT '12, page 38–49, New York, NY, USA, 2012. Association for Computing Machinery.