PARALLEL COMPUTING

Aditya Rajyaguru, Brandon Daryl Wanji
6582282, 6151351
Brock University
St. Catharines, ON, Canada

MACHINE LEARNING, K-NEAREST NEIGHBORS

***Abstract*—The object of this document is to analyse the performance of the K-nearest neighbors algorithm's implementation in C++ to pinpoint its bottlenecks and discuss ways to improve its performance.**

I. INTRODUCTION

The K-nearest neighbors algorithm is one of the simplest supervised machine learning algorithm used in regression and classification.
The goal here is to classify data using the KNN algorithm and showing the difference in execution time when it's implemented sequentially vs when it's implemented in parallel. Therefore the heuristics and the accuracy of the algorithm isn't the main goal as the performance of it is.

II. IMPLEMENTATION

A. Sequential Implementation

A sequential implementation of the K-nearest neighbors implies there is a need to limit as much as possible the use of any specialized libraries that might help with parallel execution optimizations. [5]
This algorithm is as follows :
K-nearest neighbors

1: Load the training and testing data-sets into arrays of type Point
2: Set K as 10% of the size of training data
3: **for all** points in test data **do**
4: - Find Euclidean distance to all training data points
5: - Store the Euclidean distances in an array of Euclidean objects
6: - Sort the Euclidean array in increasing order of distance using quick sort
7: - Choose the first K points from the Euclidean array
8: - Assign a classification to the test point based on the most occurring classification in the first K points. =0

Please find attached the algorithm fully implemented in **C++**. The Euclidean and Point objects are described under their respective sections in the Design part of this report.

B. Assumptions

The following assumptions were made ;

- The accuracy of the algorithm is not prioritized.
- The different classes for each data-sets are balanced. That is, they have equal number of points per class.
- Finding the best value for **K** is not prioritized.

III. DESIGN

The Design phase for this project was a crucial phase as it had to be simple, easy to understand and modular. This phase brought into perspective the different needs for this project.
The different needs were divided into 5 sections, which are :

A. Data-sets

In order to test algorithm in different scenarios, 4 different data-sets were used. **The Prostate Cancer data-set** [3], **Abalone data-set** [1], **Letters data-set** [2] and **Breast Cancer data-set** [4].
The data-sets are used to demonstrate how long the algorithm could take with the number of data points increasing.

	Prostate Cancer	Breast Cancer	Abalone	Letters
# of data points	100	569	4178	20000

B. Point Class

The point class represents a data point (A row) in the CSV file. It's used to abstract the classification and the attributes for each data point. It carries an array of Floating point called coords which holds the attributes of each line in the CSV file, the size of the array as an int and a char containing the letter of the classification it belongs to. This class is also responsible for calculating the Euclidean distance between an instance of itself Point P, and another Point Q. It also has functions for printing the ID and classification of itself as well as having getters and setters for the variables defined earlier.

C. Euclidean Class

The Euclidean class abstracts the distance for a point P to be classified in the testing set to another Point Q in the training set. It also holds the pointer for Point Q. This class helps to easily identify the point from which the distance was calculated as we use an array of these objects and sort them as defined in step 5 of the pseudo-code. Therefore, once sorted the pointer to Point Q will help figure out what Point Q's classification is.

D. Sorting

A standard quicksort algorithm is used here. The pivot is not chosen at random, in this case it picks the element in the middle of the array as its pivot every time. This helps to divide the array into two equal parts. Because this algorithm divides the array into half it'll also gives opportunity to run it in parallel on the two halves of the array, allowing for data parallelism.

E. KNN.cpp

This file handles the parsing of all the data-sets and contains the implementation of the pseudo-code defined earlier. It defines three arrays for test data and train data of type Point and an array of type Euclidean for the distances that are calculated between each test point and train point. For every test point it'll calculate the distance to every training point and sort them using quicksort, figure out the most recurring classification using the function called mode and prints it to standard out. It also checks execution time that starts right before the distances are calculated and after the final classifications are returned.

F. Compiling and Running

A make file was created to ease the execution of the program.
The program's execution is as follows :

- On the terminal, run the make file with the command. **make**
- When the binary file KNN is run the user will be prompted to enter a number from 0 to 3 inclusive to choose the data-set to use.

IV. ANALYSIS ON PERFORMANCE

With the implementation done, the 4 data-sets are analyzed. The table below shows the average execution time of the algorithm over 5 runs for each data-set. The data-sets are in increasing order of points from left to right.

Prostate Cancer	Breast Cancer	Abalone	Letters
0.6	25	669	17775

TABLE I
AVERAGE EXECUTION TIMES OVER 5 RUNS FOR EACH DATA-SET

As seen above, we can see a considerate increase in time as we go from the prostate cancer data-set to the Letters data-set. The performance degrades as the number of data points increases greatly. The value selected for K neighbours whether it's 10% of the training data set or 90% does not have a significant impact on the performance. Not to the same extent as sorting and calculating the distances for these elements. The distance calculations run in $O(n^2)$ and the sorting also runs in $O(n^2)$. In comparison, selecting K numbers from the array only takes $O(K)$ time.

V. BOTTLENECKS

The two largest bottlenecks in this code are the quicksort and the euclidean distance calculations. Both euclidean distance calculations and quicksort have the possibility for data parallelism. For quicksort, the algorithm in KNN.cpp runs recursively on two halves of the array.

VI. CONCLUSION

To conclude, the analysis of the K-nearest neighbors algorithm's performance over 04 different data-sets shows the number of points per data-set is proportional to the parallelization potential for each data-set. That is, of all these data-sets, the Letters data-set has the most parallelization potential.

REFERENCES

[1] (a) Original owners of database: Marine Resources Division Marine Research Laboratories Taroona Department of Primary Industry and Hobart Tasmania 7001 Australia (contact: Warwick Nash +61 02 277277 wnash@dpi.tas.gov.au) (b) Donor of database: Sam Waugh (Sam.Waugh@cs.utas.edu.au) Department of Computer Science University of Tasmania GPO Box 252C Hobart Tasmania 7001 Australia (c) Date received: December 1995 Fisheries, Tasmania GPO Box 619F. *Abalone Data-set.* 1995.

[2] 1991 Creator: David J. Slate Odesta Corporation; 1890 Maple Ave; Suite 115; Evanston, IL 60201 Donor: David J. Slate (dave@math.nwu.edu) (708) 491-3867 Date: January. *Letter Immage Recognition Data.* 1991.

[3] https://www.kaggle.com/harshkumarkhatri/prostate-cancer/metadata Harsh Kumar Khatri. *Prostate Cancer Data-set.* 2020.

[4] https://www.kaggle.com/nsaravana/breast-cancer/metadata N Saravana. *Breast Cancer Data-set.* 2018.

[5] Project step 2 file Robson Degrande. *Project Requirement.* 2020.