Breast Cancer Diagnosis Research

By **Joyal Biju Kulangara (40237314)** and **Kevin Mandiouba (40243497)**

"We certify that this submission is the original work of members of the group and meets the Faculty's Expectations of Originality"

June 18th, 2024

***Abstract* - The objective of this project is to design and implement a machine-learning model for diagnosing breast cancer using data structures such as k-D Trees and Ball Trees, coupled with a K-nearest-neighbor algorithm. In this report we will discuss how these data structures can work with this powerful algorithm to solve real-world issues. The results obtained during our testing show that the machine-learning model designed and implemented is effective, with an acceptable level of accuracy.**

I. Introduction

Breast Cancer is a widely common and potentially deadly disease affecting generations of women around the world. Having accurate and early diagnosis is crucial for effective treatment in patient outcomes when inflicted with the diseases. The rise of machine learning has given developers proper tools to predict and aid in diagnosing diseases including breast cancer. This project aims to focus on using machine learning techniques, specifically the k-nearest neighbour (k-NN) algorithm, to classify breast cancer as either malignant (M) or benign (B) based on the provided data. Using two data structures including k-d trees and ball trees, we were able to enhance the k-NN algorithm's performance. This project aims to optimize the computational efficiency and scalability of the k-NN algorithm, making it suitable to run and work with large datasets to predict breast cancer solutions before they appear.

This report will discuss the concise overview of the project's problem, objectives and methods, descriptions of the data structures and algorithms used, and a results and analysis section to present the findings and insights gained from the project. By the end of this report, readers will be able to derive a comprehensive understanding of the approach employed, along with the outcomes achieved to improve breast cancer diagnosis through machine learning.

II. Problem

This research involves the study of developing an efficient and accurate machine learning model to be able to accurately classify breast cancer diagnoses based on patient cell data. According to the breast cancer dataset [1], the classification of the disease is either malignant (M) or benign (B) based on attributes that are taken into account when deciding its diagnosis. Being malignant suggests that the tumors found through breast cancer are cancerous, signifying that they need immediate treatment, whereas having a benign treatment suggests that the tumors are not cancerous, showing that they do not invade nearby tissues or spread to other parts of the body [2]. The objective of this research is to implement a machine learning model, namely constructing two tree-based models to be able to enhance both the accuracy and computational efficiency of the classification process. Having the k-NN algorithm will enable the machine learning system to conduct adhered searches to classify the new instances based on attributes linked to each patient, which helps in evaluating the performance in terms of predictive accuracy and scalability with varying sizes of the provided training data.

III. Description of Model

This section provides a comprehensive characterization of the models used to solve the problem of classifying breast cancer diagnoses. The implementation utilizes advanced data structures including k-d trees and ball trees, which we used to enhance the efficiency and accuracy of the k-NN algorithm.

A. Data Analysis

The implementation stemmed from reading the breast cancer dataset that was downloaded as a CSV file including the 569 instances of patients with their IDs, the diagnosis, and 10 other records. A

LinkedHashMap named *dataMap* was used to store the information of the CSV file. The IDs were then taken from the HashMap and then put in an array of IDs thus shuffled to have the patients chosen at random from the complete set for the instances of training and testing the data. The shuffling helps create diverse training and testing sets for the k-NN algorithm. Two different HashMaps are then created to be used for training and testing. Dividing the two HashMaps required going through the shuffled IDs and splitting the original set into training and testing. The training and testing HashMaps only included the IDs as the key and all the other 10 attributes as the value. This is because it will be useful when implementing the data structures for the machine learning model to analyze if the algorithm correctly identifies the diagnosis or not. The program iterates through the shuffled list of IDs to assign the first 'N' patients to the training set, and the next 'T' patients to the testing set, to ensure that no patient appears in both sets. The number of records used depended on the size of the training sample and the testing sample, whose size is ¼ of the training sample. One thing to note is that if the dataset could not maintain the 4 to 1 ratio, the remaining data sets would be taken from the original HashMap to complete the testing records. If the training set uses all the patients in the dataset or tries to use more patients, our program would return an error to alert the user that it surpassed the dataset.

B. Data Structures

For our program, we implemented two different tree-based data structures to test the efficiency and accuracy of our program. The k-D tree and Ball tree are both implemented differently using the k-NN algorithm to efficiently find the diagnosis based on their implementation.

1) k-D tree

A k-D tree (k-dimensional tree) is used as a space-partitioning data structure to organize points in a k-dimensional space. Being a useful data structure for searches involving a multidimensional search key, k-d trees are efficient in finding the nearest neighbor in high-dimensional spaces [3]. For this research, the k-d tree was constructed by recursively splitting the training data along different dimensions. We utilized a 10-dimensional k-d tree since 10 attributes were defined as the values for its implementation from the dataset. At every level of the tree, the dimension with the greatest spread was used for splitting the data, and the median along the point of the dimension was selected as the node. The left and right children of the tree are assembled recursively using the points on both sides of the median. A low-level pseudocode describing our k-D tree implementation is presented in Figure 1.

2) Ball tree

A ball tree is also used as a space-partitioning data structure for organizing points in a multi-dimensional space to organizing points in a multi-dimensional space. Its approach is quite different from the k-D tree as it partitions the data points into a nested set of balls, which also makes it efficient for the nearest neighbor search. Being a binary tree, each of its nodes defines a D-dimensional ball-like structure, that contains the subset of points to be searched [4]. For this research, the ball tree was constructed by recursively splitting portioning the training data into two subsets each contained within a ball. The root node is created using the median point, and the radius is determined by finding the maximum distance from the pivot point to any point in the subset. The left and right children of the tree are created recursively using the points on either side of the pivot. A low-level pseudocode describing our k-D tree implementation is presented in Figure 2.

C. Classification Algorithm

For this research, to implement both the k-D and ball trees, we utilized the K-Nearest Neighbor (KNN) algorithm. The kNN algorithm is a supervised classification algorithm used in machine learning for various applications, to properly distribute data, based on prior given data, to classify coordinates identified by an attribute [5]. To be able to identify the nearest points as a query point, we utilized the Euclidean Distance calculation as the metric to calculate the cartesian distance between the points found in the dimensional trees. This is what enabled the calculation of the net displacement that is carried out between various points, to get to the nearest neighbors. For our program, we implemented the *kNearestNeighbors* method, with kValues 1, 5, and 7, for each test instance to find the k nearest neighbors from the training set, and the majority vote among the neighbors to determine the predicted diagnosis. The implementation for both the k-D tree and the Ball tree had a few differences based on the implementation of the k-NN algorithm.

1) k-NN search using a k-D tree

For the k-D tree, the traversal occurred recursively to find the k-nearest neighbors. Using a max-heap (a priority queue), we were able to track the k nearest neighbors. The algorithm was able to calculate the distance to the target point and update the heap if necessary for each node. It then determines which subtree has to be searched first and thus recursively searches both subtrees if they contain closer points or not. The flowchart of our K-D tree implementation coupled with the k-NN algorithm is presented in Figure 3.

2) k-NN search using ball tree

For the ball tree, the traversal also occurred recursively to find the k-nearest neighbors. A max-heap (a priority queue) is used similar to the k-d tree search. The algorithm was able to calculate the distance to the target point and update the heap if necessary for each node. It then calculates the distance to the pivot point and the radius, and recursively searches the child nodes if they contain closer points or not. The flowchart of our Ball Tree implementation coupled with the k-NN algorithm is presented in Figure 4.

3) Majority Vote for Classification

The classification occurs based on each test instance, whereby the k nearest neighbors is retrieved. The majority vote among their diagnosis thus determines the predicted diagnosis. The *majorityVote* method counts the occurrences of each diagnosis among the neighbors and selects the most frequent one, which is then used as the predicted diagnosis. We then calculate the accuracy of the predicted diagnosis to the actual diagnosis based on various training and testing values that are provided.

IV. Results and Discussions

The results from the testing of the training model showed a variety of results that the accuracy and running time of how our code works for both the data structures that are implemented. The running time and accuracy values shown as screenshots based on how our program runs are presented in the Appendix in Figure 9 and Figure 10.

A. k-D Tree

The accuracy of the k-D tree algorithm can be demonstrated in Figure 5, whereby it can be noted how the accuracy of our k-D Tree algorithm improves with an increase in the training sample size (N). The curves for k=1, k=5, and k=7, show a positive trend, giving an idea of how more training data generally leads to better accuracy in classification.

Figure 5 also shows that for smaller training sets (N=100 to N=200), k=1 shows lower accuracy compared to k=5 and k=7. However, for larger training sets (N=400 and above), the accuracy differences between different k values reduce, with all the k values achieving close to similar high accuracy. Another note is that k=5 and k=7 perform slightly better than k=1 for smaller training sizes, however, the performance gap narrows as the value of N increases steadily.

In terms of running time for the k-D tree, based on Figure 6, it can be noted that the runtime generally increases with the training sample size (N), but not linearly. The curves for k=1, k=5 and k=7 show flexibility, indicating that the runtime is massively influenced by the value of k and the training sample size. Specifically, it can be noted that the running time for k=7 consistently has the highest runtime across all training sizes, which reflects the increased computational cost of finding more neighbors using the data structure. We can also identify that k=1 generally has the lowest runtime, which is its expected outcome, due to the simpler nearest neighbor search. There is a notable spike in runtime for k=7 at N=300, which shows certain inefficiencies or program challenges could have occurred at this point.

B. Ball Tree

The accuracy of the Ball Tree algorithm can be demonstrated in Figure 7, whereby it can be noted how the accuracy of our Ball Tree algorithm also steadily improves with an increase in training sample size (N). The curves for k=1, k=5, and k=7 also show a positive trend, but the improvement rate varies more compared to how the k-D Tree is implemented.

Figure 7 shows that for smaller training sets (N=100), k=1 shows higher accuracy compared to k = 5 and k=7, which is quite different from the k-D Tree results. For the larger training sets including k=5 and k=7, a similar performance can be noted with slightly higher accuracy compared to k=1. The accuracy for k=7 remains relatively stable after N=200, which shows less sensitivity to additional training data compared to k=1 and k=5.

In terms of the running time for the Ball tree, based on Figure 8, it can be noted that the runtime shows variability with the training sample size (N), but the trends are less consistent compared to the k-D tree. The k values exhibit fluctuations in runtime, which highly indicates the sensitivity to both the values of k and the distribution of training data. On an overall basis, it can be noted that k=1 consistently has the lowest runtime, which aligns with the simpler nearest neighbor search. However, k=5 and k=7 both portray higher runtimes, with k=7 generally having the highest runtime due to the increased complexity. The runtime for k=7 also peaks at N=400, indicating that there is a very high computation that is taking place for a high level of training size.

C. Comparison between both data structures

Overall, it can be noted that both the k-D Tree and Ball Tree algorithms improve in their accuracy with larger training sample sizes. For k=5 and k=7, generally, there is a higher performance than k=1 for smaller training sets, but this trend vastly portrays similar high accuracy values for larger training sets.

In terms of runtime, there is an increase in training sample size for both algorithms, and k=7 consistently shows the highest runtime on average. Significant runtime fluctuations also indicate that both algorithms seem to be sensitive to the value of k and training data distribution. In terms of performance, it can be noted that the k-D Tree algorithm shows more volatility in runtime compared to Ball Tree. This suggests that the k-D is more optimizable with larger datasets.

In theory, both the k-D Tree and the Ball Tree algorithms have an $O(\log N)$ time complexity for nearest neighbor searches, whereby N defines the number of points in the dataset. However, the actual time complexity of the k-D tree is $O(N \log N)$, which is due to the sorting step that occurs after every level of the tree, along with the Ball tree, which also has a time complexity of $O(N \log N)$, due to the partitioning method used to organize data points, which affects its efficiency. With regards to the complexities, however, it can be noted that the Ball Tree has better scalability and stability with the different training sizes and k values, which makes it more suitable for larger and complex datasets.

V. Conclusion

In conclusion, this research study has demonstrated how the k-d Tree and Ball Tree algorithms were used to enhance the k-nearest neighbor (k-NN) algorithm for breast cancer diagnosis, showing high improvements in accuracy as the training sample grew. The k-D tree algorithm exhibited strong performance, with a significant runtime variability for certain higher k values, due to the dataset structure and how it was implemented based on our program. Based on the implementation of the Ball tree algorithm, it was evident that it provided more stable and consistent runtimes, giving it more scalability and efficiency for larger datasets and higher k values. Overall, this research helped us grasp how data structures and algorithms work, as well as how we can use them to implement machine learning models to sort and work with large datasets to solve real-world problems. This project could have been potentially improved by running the program multiple times to make it more scalable and reduce anomalies for optimal performance. This can minimize fluctuations, allowing us to diversify the use of this system for further real-world datasets. Overall, the use of these algorithms can be applied to other medical datasets or real-world scenarios, expanding our understanding of how far machine learning can be evolved. This would help in identifying domain-specific challenges which can be comprehensively by tailoring algorithms.

ACKNOWLEDGMENTS

We would like to express our deepest gratitude to Dr. Nawwaf Kharma for his guidance and support throughout the semester, which were instrumental in our understanding of data structures and algorithms. We also wish to extend our heartfelt thanks to Zhiyang Deng and Kaiyu Nie for their invaluable assistance and insightful guidance when we encountered challenges with the project. Thank you, Dr. Kharma, Zhiyang, and Kaiyu, for your encouragement and support.

REFERENCES

[1] UCI Machine Learning Repository, "Breast Cancer Wisconsin (Diagnostic) Data Set," [Online]. Available: https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data?resource=download. [Accessed 10 June 2024].

[2] Y. Brazier and T. Rush, "What are the different types of tumor?," Medical News Today, 16 November 2023. [Online]. Available: https://www.medicalnewstoday.com/articles/249141. [Accessed 18 June 2024].

[3] Rosetta Code, "K-d Tree," [Online]. Available: https://rosettacode.org/wiki/K-d_tree. [Accessed 10 June 2024].

[4] Wikipedia , "Ball tree," [Online]. Available: https://en.wikipedia.org/wiki/Ball_tree. [Accessed 12 June 2024].

[5] GeeksforGeeks, "K-Nearest Neighbor(KNN) Algorithm," 25 January 2024. [Online]. Available: https://www.geeksforgeeks.org/k-nearest-neighbours/. [Accessed 10 June 2024].

[6] N. Kharma, *COEN 352 Summer_1 Assignment 3*, Department of Electrical & Computer Engineering (ECE) Concordia University, 2024.

[7] micycle1, "KMeansCluster," [Online]. Available: https://github.com/micycle1/KMeansCluster. [Accessed 12 June 2024].

VI. Appendix

```
function constructK-D tree(trainingRecords):
    rootNode = null
    for patient in trainingRecords:
        if rootNode is null:
            rootNode = patient
            break

    insertRecursive(rootNode, trainingRecords, 0)

function insertRecursive(node, records, depth):
    if records is empty:
        return

    medianIndex = findMedian(records, depth)
    medianRecord = records[medianIndex]

    node.left = insertRecursive(node.left, records less than medianRecord, depth + 1)
    node.right = insertRecursive(node.right, records greater than medianRecord, depth + 1)

function findMedian(records, depth):
    dimension = depth % numberOfDimensions
    sort records by dimension
    return index of the median record

function kNearestNeighbors(rootNode, targetNode, k):
    priorityQueue = empty
    searchK-D tree(rootNode, targetNode, k, priorityQueue)
    return extractNodesFromQueue(priorityQueue)

function searchK-D tree(node, targetNode, k, priorityQueue):
    if node is null:
        return

    distance = calculateDistance(node, targetNode)
    addNodeToQueue(priorityQueue, node, distance, k)

    dimension = depth % numberOfDimensions
    if targetNode[dimension] < node[dimension]:
        searchK-D tree(node.left, targetNode, k, priorityQueue)
        if size of priorityQueue < k or distance to the other side < max distance in queue:
            searchK-D tree(node.right, targetNode, k, priorityQueue)
    else:
        searchK-D tree(node.right, targetNode, k, priorityQueue)
        if size of priorityQueue < k or distance to the other side < max distance in queue:
            searchK-D tree(node.left, targetNode, k, priorityQueue)
```

Figure 1: K-D tree Pseudocode.

```
function constructBallTree(records):
    if records size is 1:
        return new Node(records[0])

    dimension = findGreatestSpreadDimension(records)
    sort records by dimension
    medianIndex = records.size / 2
    pivot = records[medianIndex]

    leftRecords = records from 0 to medianIndex - 1
    rightRecords = records from medianIndex + 1 to end

    child1 = constructBallTree(leftRecords)
    child2 = constructBallTree(rightRecords)

    radius = calculateMaxDistance(pivot, records)
    return new Node(pivot, child1, child2, radius)

function findGreatestSpreadDimension(records):
    initialize min and max for each dimension
    for each record:
        update min and max for each dimension
    return dimension with the greatest spread (max - min)

function kNearestNeighbors(rootNode, targetNode, k):
    priorityQueue = empty
    searchBallTree(rootNode, targetNode, k, priorityQueue)
    return extractNodesFromQueue(priorityQueue)

function searchBallTree(node, targetNode, k, priorityQueue):
    if node is null:
        return

    distance = calculateDistance(node, targetNode)
    addNodeToQueue(priorityQueue, node, distance, k)

    if node has children:
        distanceToPivot = calculateDistance(node, targetNode)
        if child1 is closer or queue size < k:
            searchBallTree(node.child1, targetNode, k, priorityQueue)
        if child2 is closer or queue size < k:
            searchBallTree(node.child2, targetNode, k, priorityQueue)
```

Figure 2: Ball Tree Pseudocode

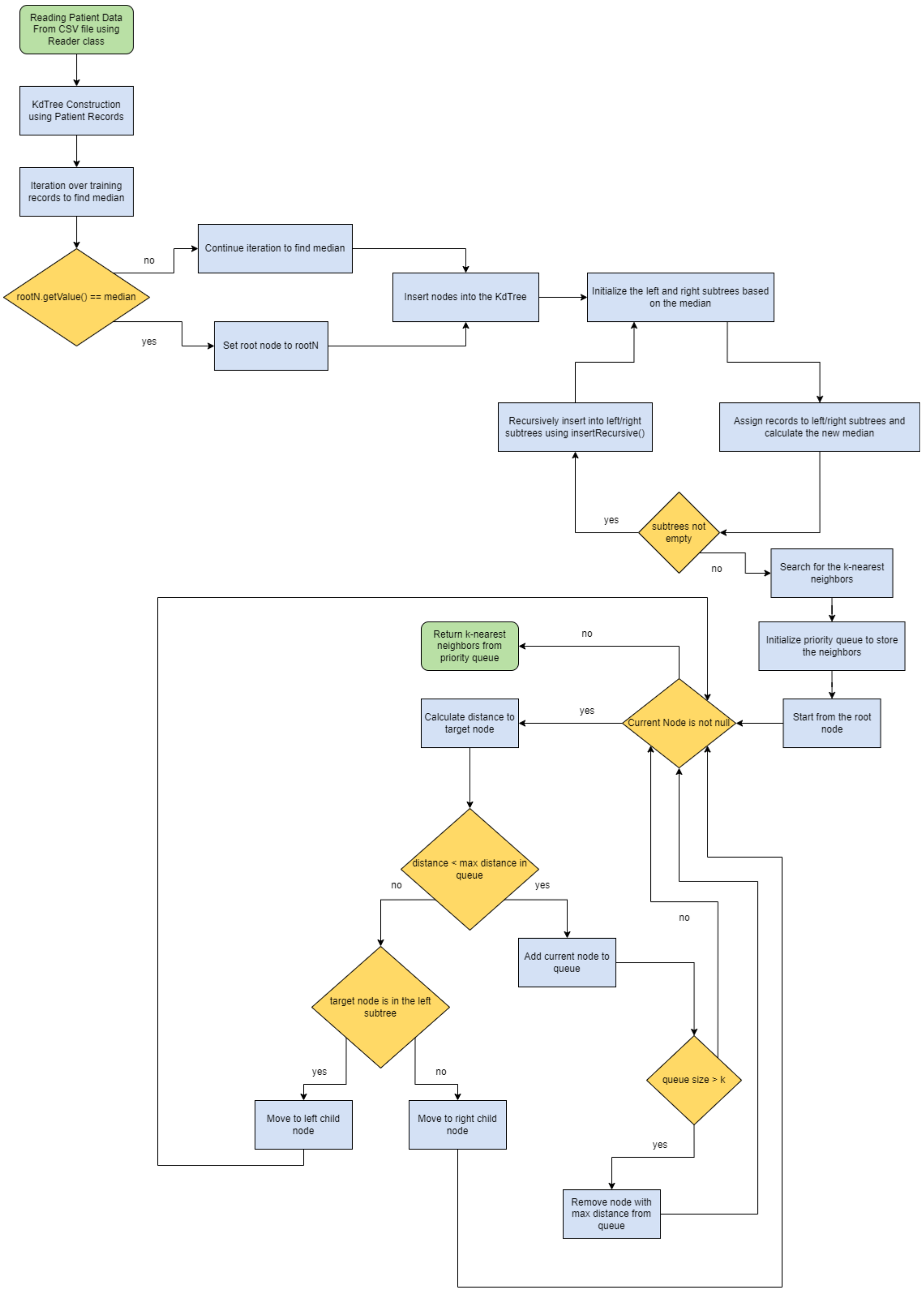

Reading Patient Data From CSV file using Reader class
KdTree Construction using Patient Records
Iteration over training records to find median
rootN.getValue() == median
no
Continue iteration to find median
yes
Set root node to rootN
Insert nodes into the KdTree
Initialize the left and right subtrees based on the median
Recursively insert into left/right subtrees using insertRecursive()
Assign records to left/right subtrees and calculate the new median
yes
subtrees not empty
no
Search for the k-nearest neighbors
Initialize priority queue to store the neighbors
Start from the root node
Return k-nearest neighbors from priority queue
no
Calculate distance to target node
yes
Current Node is not null
distance < max distance in queue
no
yes
Add current node to queue
no
target node is in the left subtree
queue size > k
yes
no
Move to left child node
Move to right child node
yes
Remove node with max distance from queue

Figure 3: K-D tree with KNN algorithm Flowchart.

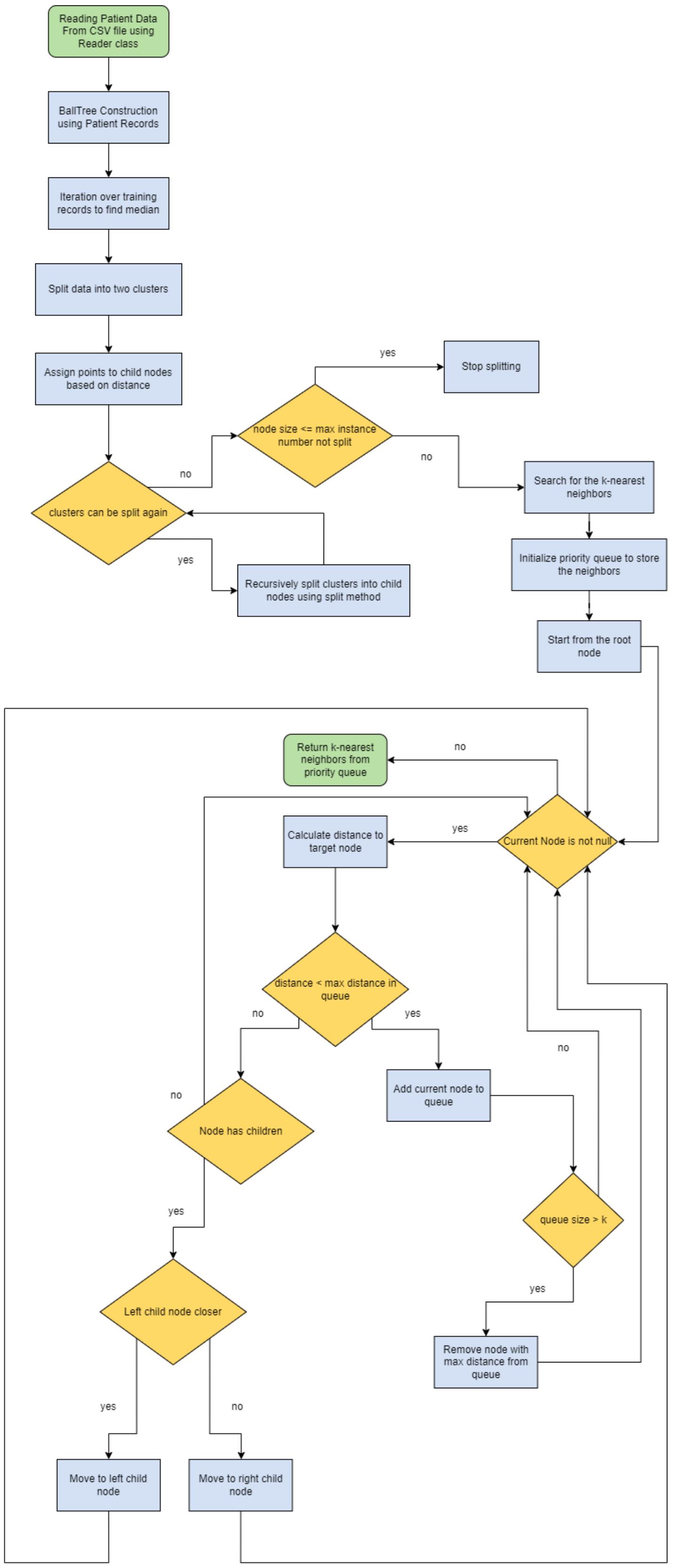

Reading Patient Data From CSV file using Reader class
BallTree Construction using Patient Records
Iteration over training records to find median
Split data into two clusters
Assign points to child nodes based on distance
clusters can be split again
no
node size <= max instance number not split
yes
Stop splitting
no
Search for the k-nearest neighbors
yes
Recursively split clusters into child nodes using split method
Initialize priority queue to store the neighbors
Start from the root node
Return k-nearest neighbors from priority queue
no
yes
Current Node is not null
Calculate distance to target node
distance < max distance in queue
no
yes
Add current node to queue
no
no
Node has children
queue size > k
yes
yes
Left child node closer
Remove node with max distance from queue
yes
no
Move to left child node
Move to right child node

Figure 4: Ball Tree with KNN algorithm Flowchart

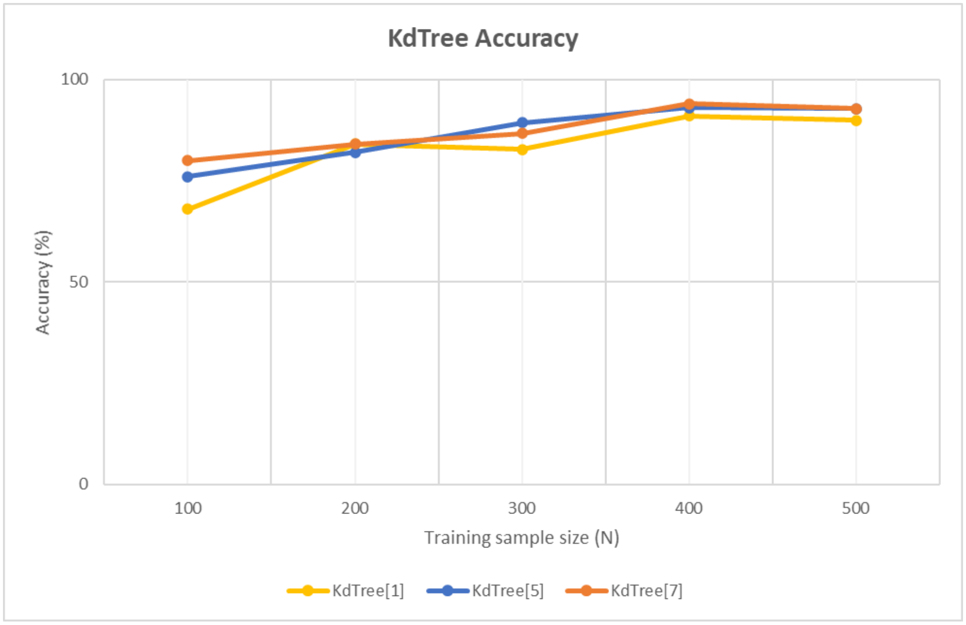


Figure 5: K-D tree Accuracy.

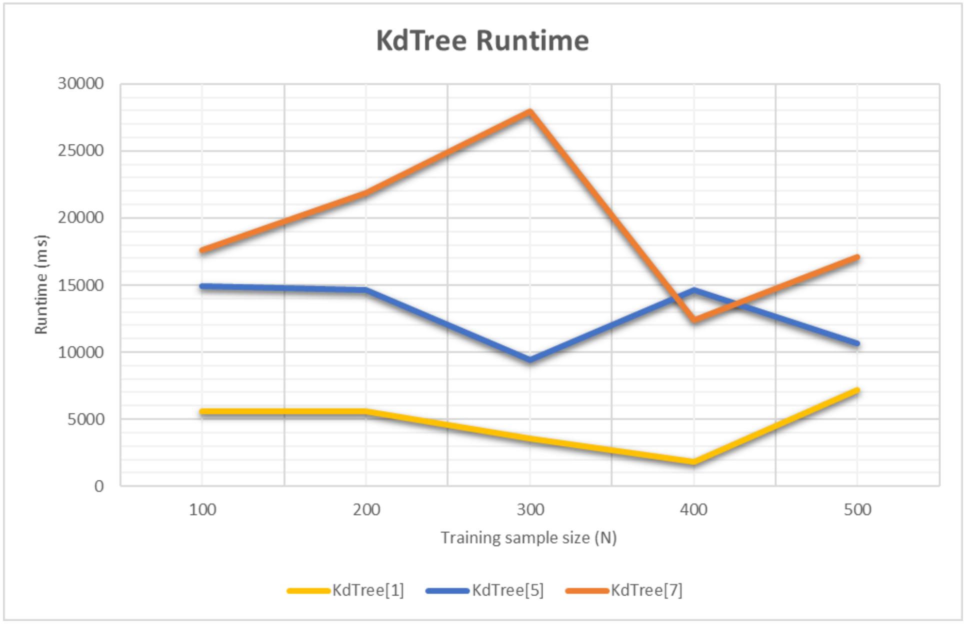


Figure 6: K-D tree Runtime.

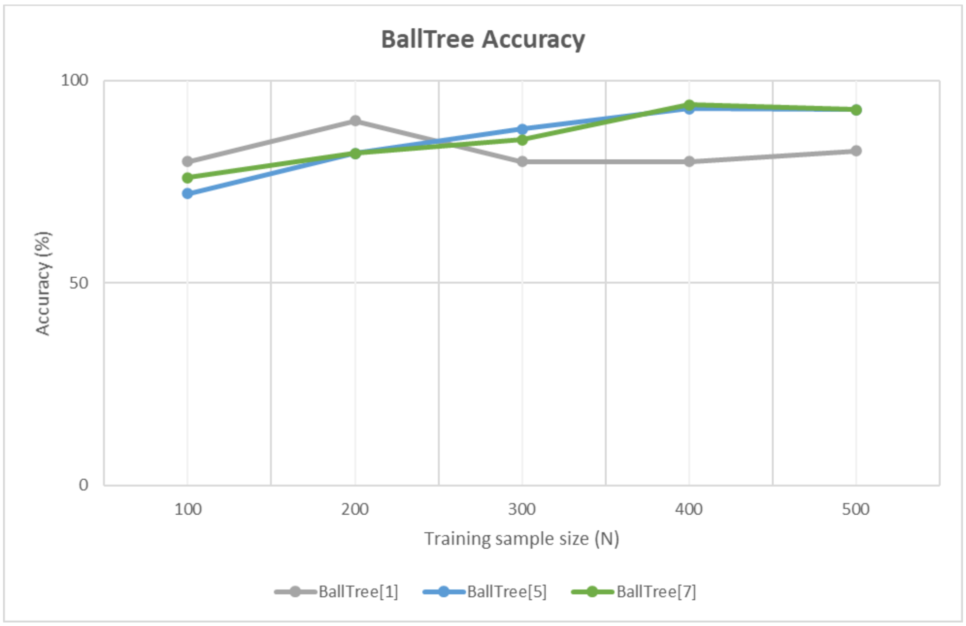


Figure 7: Ball Tree Accuracy

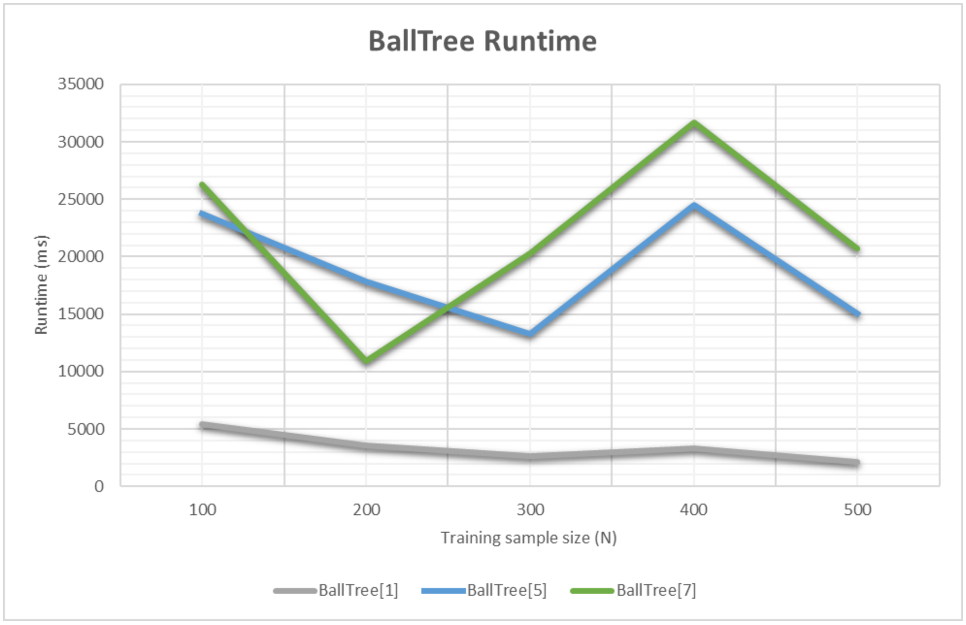
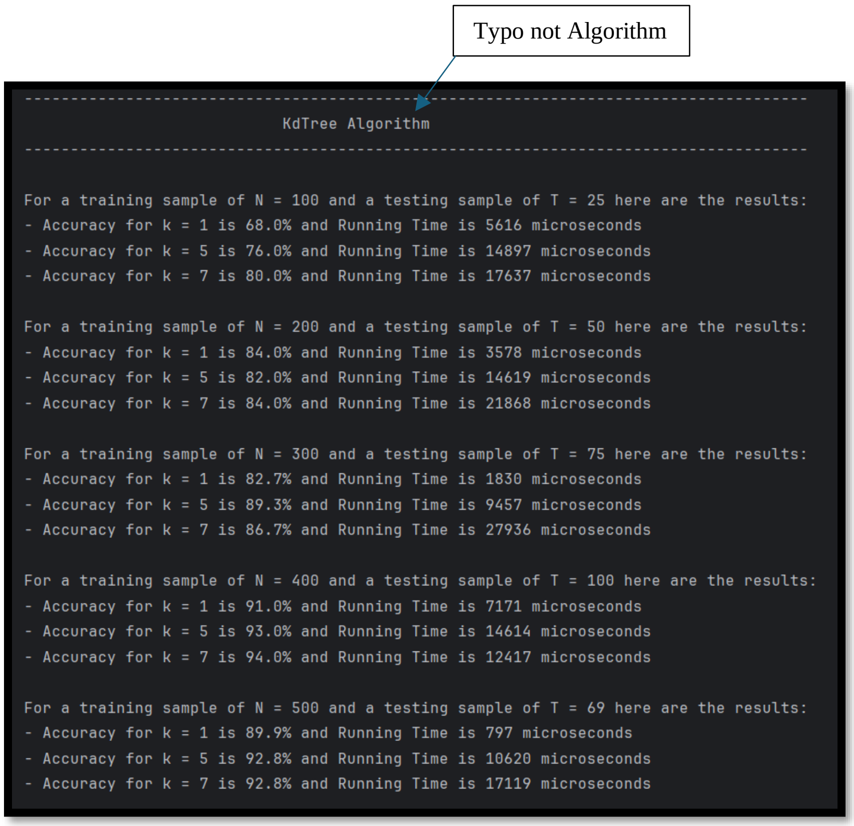
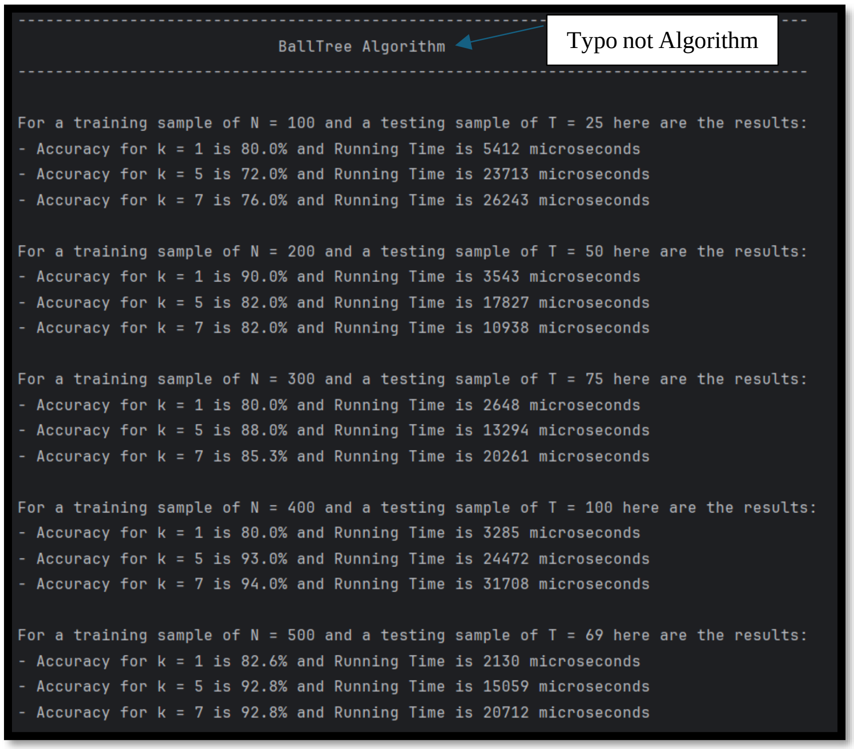


Figure 8: Ball Tree Runtime



```
----------------------------------------------------------------------------------------
                            KdTree Algorithm
----------------------------------------------------------------------------------------

For a training sample of N = 100 and a testing sample of T = 25 here are the results:
- Accuracy for k = 1 is 68.0% and Running Time is 5616 microseconds
- Accuracy for k = 5 is 76.0% and Running Time is 14897 microseconds
- Accuracy for k = 7 is 80.0% and Running Time is 17637 microseconds

For a training sample of N = 200 and a testing sample of T = 50 here are the results:
- Accuracy for k = 1 is 84.0% and Running Time is 3578 microseconds
- Accuracy for k = 5 is 82.0% and Running Time is 14619 microseconds
- Accuracy for k = 7 is 84.0% and Running Time is 21868 microseconds

For a training sample of N = 300 and a testing sample of T = 75 here are the results:
- Accuracy for k = 1 is 82.7% and Running Time is 1830 microseconds
- Accuracy for k = 5 is 89.3% and Running Time is 9457 microseconds
- Accuracy for k = 7 is 86.7% and Running Time is 27936 microseconds

For a training sample of N = 400 and a testing sample of T = 100 here are the results:
- Accuracy for k = 1 is 91.0% and Running Time is 7171 microseconds
- Accuracy for k = 5 is 93.0% and Running Time is 14614 microseconds
- Accuracy for k = 7 is 94.0% and Running Time is 12417 microseconds

For a training sample of N = 500 and a testing sample of T = 69 here are the results:
- Accuracy for k = 1 is 89.9% and Running Time is 797 microseconds
- Accuracy for k = 5 is 92.8% and Running Time is 10620 microseconds
- Accuracy for k = 7 is 92.8% and Running Time is 17119 microseconds
```

Figure 9: K-D tree test results.



```
----------------------------------------------------------------------------------------
                            BallTree Algorithm
----------------------------------------------------------------------------------------

For a training sample of N = 100 and a testing sample of T = 25 here are the results:
- Accuracy for k = 1 is 80.0% and Running Time is 5412 microseconds
- Accuracy for k = 5 is 72.0% and Running Time is 23713 microseconds
- Accuracy for k = 7 is 76.0% and Running Time is 26243 microseconds

For a training sample of N = 200 and a testing sample of T = 50 here are the results:
- Accuracy for k = 1 is 90.0% and Running Time is 3543 microseconds
- Accuracy for k = 5 is 82.0% and Running Time is 17827 microseconds
- Accuracy for k = 7 is 82.0% and Running Time is 10938 microseconds

For a training sample of N = 300 and a testing sample of T = 75 here are the results:
- Accuracy for k = 1 is 80.0% and Running Time is 2648 microseconds
- Accuracy for k = 5 is 88.0% and Running Time is 13294 microseconds
- Accuracy for k = 7 is 85.3% and Running Time is 20261 microseconds

For a training sample of N = 400 and a testing sample of T = 100 here are the results:
- Accuracy for k = 1 is 80.0% and Running Time is 3285 microseconds
- Accuracy for k = 5 is 93.0% and Running Time is 24472 microseconds
- Accuracy for k = 7 is 94.0% and Running Time is 31708 microseconds

For a training sample of N = 500 and a testing sample of T = 69 here are the results:
- Accuracy for k = 1 is 82.6% and Running Time is 2130 microseconds
- Accuracy for k = 5 is 92.8% and Running Time is 15059 microseconds
- Accuracy for k = 7 is 92.8% and Running Time is 20712 microseconds
```

Figure 10: Ball Tree test results.