Breast Cancer Wisconsin (Diagnostic) Prediction in Java

Anh Quan Truong
2nd Year Undergraduate Student
Dept. of Electrical and Computer Engineering
Concordia University
Montréal, Québec, Canada
quanat1x4@gmail.com

***Abstract* - Breast cancer remains, to this day, one of the more common cancers in woman with 2.3 million cases being diagnosed and treated each year [1]. Breast cancer is classified either as benign cancer or malignant cancer, with the latter requiring urgent treatment. It is therefore of paramount importance that a reliable and accurate diagnostic system be developed. One of the ways to achieve such a system is to use Machine Learning (ML), a branch of Artificial Intelligence (AI) that forms patterns and connections using historical data in order to make future predictions [2]. The objective of this exercise is to classify breast cancer using k-nearest neighbors (KNN).**

***Keywords* - breast cancer, Machine Learning, Artificial Intelligence, KNN**

A. Overview of the Exercise

A dataset of 569 breast cancer cases is given, with each case classified either as benign or malignant based on their mean measurements in 10 different categories that are used in the diagnosis of cancer. The ML model employed in this exercise is KNN which uses proximity to classify a random set of measurements (i.e., a random case). In other words, if the neighboring (i.e., closest) cases of the said random case mostly bear the benign classification then chances are, it is also benign [3]. A subset of the given dataset is used for training, while another subset is used for testing. The validity of the model is then evaluated based on its ability to correctly predict and how efficiently it makes those predictions. As a result, in order to achieve both speed and accuracy, an abstract data type known as tree is used, namely k-d tree and ball tree. The chosen tree is first populated with the train data points (along with their respective classifications) according to the tree's specifications. A random point, whose classification is omitted, is then selected and a search is performed on the tree to find the closest neighbors to this random point.

B. Model Description

To construct a K-d tree, a point is first selected at random within the train data pool. This is set as the root of the tree. The data space is then split into 2 halves along the first dimension (i.e., the very first category which, in this case, is radius mean). Another random point from the same pool is selected and placed on the tree, where it ends up depends on its value in the first dimension. For example, if the second point has a radius mean that is smaller than that of the root, then it will be placed on the left of the root (left child). If its radius mean is larger than that of the root, then it will go to the right of the root (right child). The data space is then split into 2 halves again but this time, along the second dimension (i.e., texture mean) and where the next point goes depends on its texture mean, assuming that its value in the first dimension is smaller than that of the root. This process is repeated until the train data pool is completely exhausted. It is worth noting that after the last and tenth dimension split, the next split cycles back to the first dimension [4]. As an attempt to balance the tree (i.e., to ensure that the tree is not lopsided), instead of selecting points at random and putting them on the tree, dimensional median points are chosen. For example, for the first dimension, the point whose radius mean is the median (i.e., half of the train data points are to the left of it and the other half, to the right of it) is picked. To accomplish this, a Stack is used along with a modified version of QuickSelect. A Stack is first built with all the available data points and then modified QuickSelect is used to ensure that the point

desired (i.e., median point) always stays at the top of the Stack. As the tree is being constructed, the top of the Stack gets popped off.

The construction of a ball tree is similar to that of a k-d tree in that it still heavily depends on dimensionality. However, the way that dimensionality is used here is slightly different. Instead of starting out by choosing a singular point in the train dataset, the entire data pool is used and instead of going through the dimensions sequentially, at every level, a dimension is picked based on how spread out the data points of that dimension are. For example, at the first level, in order to cover all the data points in the data pool, the chosen dimension must be the one that has the highest variance, meaning that it has the widest range of values. As a result, a ball made using this dimension would enclose the entire data space. The median point of this dimension is then chosen using QuickSelect and set as the center (or centroid) of the first ball. The distance between this center and the point farthest from it is calculated and set as the radius of the ball. The 2 left and right subsets of data points of the centroid are used to construct 2 new circles whose respective centroids are picked using the abovementioned method of dimension selection.

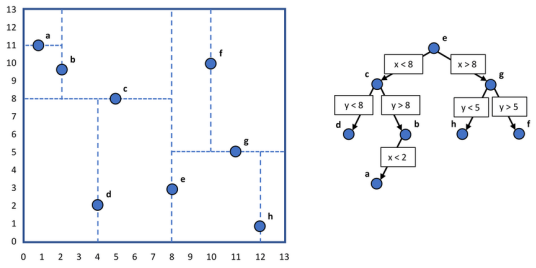

Figure 1. K-d tree (k = 2) [6]

```
function construct_kdtree is
    input:
        P, point to be inserted into tree
        h, depth of tree
    output:
        N, root or node of tree

    if P is not found in tree then
        create N containing P
        return N
    else
        let d be current dimension
        d := h mod 10
        if value of P in d is smaller than or equal to value of N in d then
            N.left := construct_kdtree(P, h+1)
        else
            N.right := construct_kdtree(P, h+1)
        end if
        return N
    end if

end function
```

Figure 2. Pseudocode for the Construction of K-d Tree [4]

The construction of a ball tree is considered to be finished if no more balls can be created [5].

Given a tree and a test point that it has not "seen" before (i.e., not a node on the tree), KNN search is commenced by introducing the data point into the tree. The test point is then "allowed" to traverse down the tree, and as it makes it way down to where it would have ended up if this were an insertion operation, all the nodes that it has seen are recorded into a list. The nodes in this list also contain information on how far they are from the test point. However, such a list could be cumbersome and difficult to manage. A method to address this issue is to use max priority queue (MaxPQ) to keep track of only the relevant neighbor nodes. For example, if prediction on the test point is to be made based on 7 closest neighbors, then this max priority queue is of size 7 or less and when it is at its full capacity (i.e., 7 nodes) and a new node is introduced into the queue, the node that is farthest from the test point is popped off and replaced by the second farthest node in the queue. By doing so, the size of the queue does not get larger than 7 as the test point traverses down the tree and all the nodes that are not close enough to it are eliminated. This method of KNN search is the same across both tree types.

```
function classify is
    input:
        P, test point
        k, number of nearest neighbors to search for
        Q, max-first priority queue with at most k points
        N, root or node of tree
    output:
        C, classification of P

    let h be depth of tree := 0
    while N is not null then
        let d be current dimension
        d := h mod 10
        h++
        if distance(P, N) < distance(P, Q.first) then
            add P to Q
            if size(Q) > k then
                remove max in Q
            end if
        end if
        if value of P in d is smaller than value of R in d then
            N := N.left
        else if value of P in d is larger than to value of R in d then
            N := N.right
        else
            if left of N is null then
                N := N.right
            else
                N := N.left
            end if
        end if
    end while
    if more "malignant" than "benign" nodes in Q then
        return C := "malignant"
    else
        return C := "benign"
    end if

end function
```

Figure 3. Pseudocode for KNN Search Using K-d Tree [4]

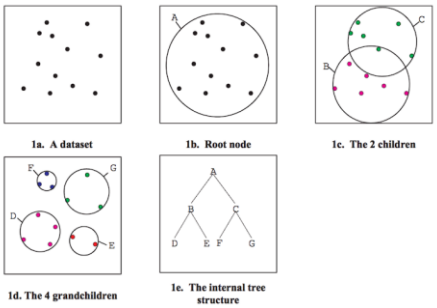

Figure 4. Ball tree [7]

```
function construct_balltree is
    input: A, array of points
    output: B, root or ball node of tree

    if no point in A then
        return null
    else if 1 point in A then
        create B containing A
        return B
    else
        let d be dimension with largest spread
        let c be centroid with respect to d
        let L, R be subsets of points to left and right of c
        let B.radius be largest distance between c and its children
        B.centroid := c
        B.left := construct_balltree(L)
        B.right := construct_balltree(R)
        return B
    end if

end function
```

Figure 5. Pseudocode for the Construction of Ball Tree [5]

```
function classify is
    input:
        P, test point
        k, number of nearest neighbors to search for
        Q, max-first priority queue with at most k points
        B, root or ball node of tree
    output:
        Q, max-first priority queue with at most k points from tree

    if distance(P, B.centroid) - B.radius >= distance(P, Q.first) then
        ignore this B since it's too far away from P
        return Q
    else if B has no children
        for each point p in B
            if distance(P, p) < distance(P, Q.first) then
                add p to Q
                if size(Q) > k then
                    remove max in Q
                end if
            end if
        end for
    else (B is an internal ball node)
        Q := classify(P, k, Q, B.left)
        Q := classify(P, k, Q, B.right)
    end if
    return Q

end function


function classify is
    input:
        P, test point
        k, number of neighbors
        B, root of tree
    output:
        C, classification of P

    let Q := classify(P, k, Q, B)

    if more "malignant" than "benign" nodes in Q then
        return C := "malignant"
    else
        return C := "benign"
    end if

end function
```

Figure 6. Pseudocode for KNN Search Using Ball Tree [5]

C. Analysis of Results

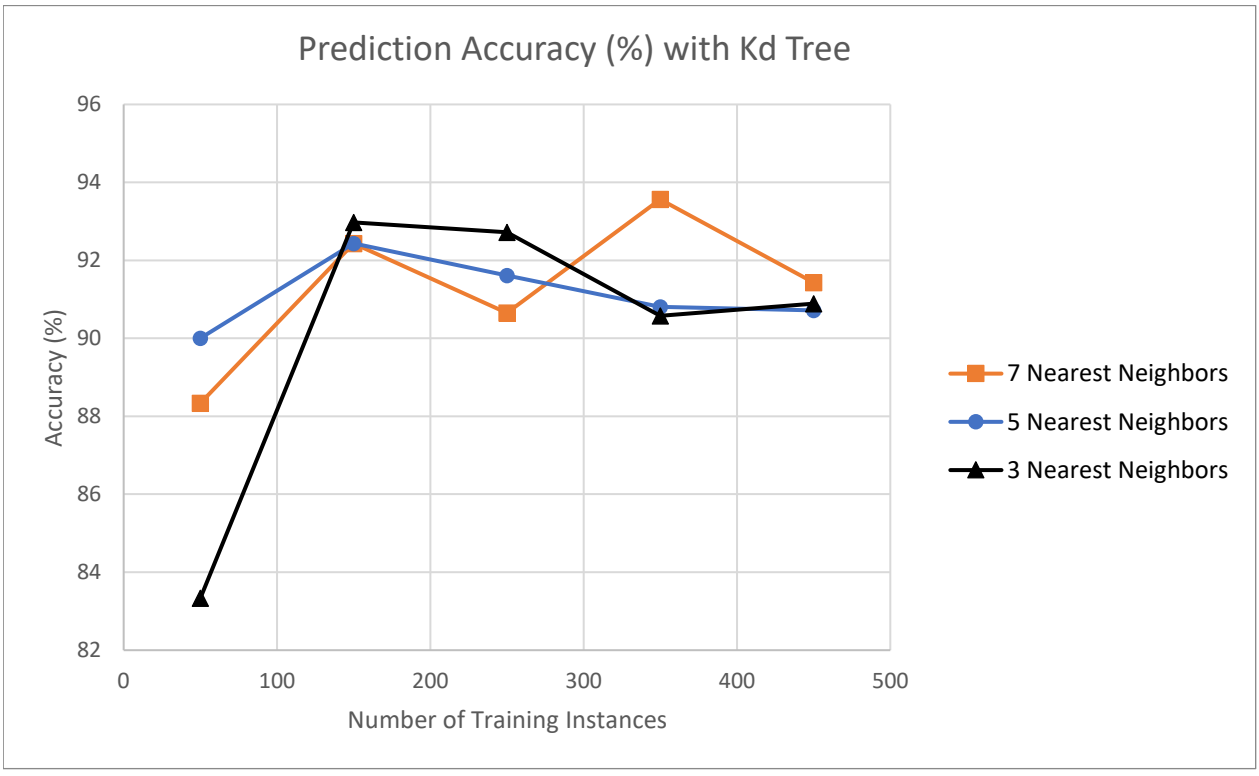

Figure 7. Prediction Accuracy (%) with K-d Tree

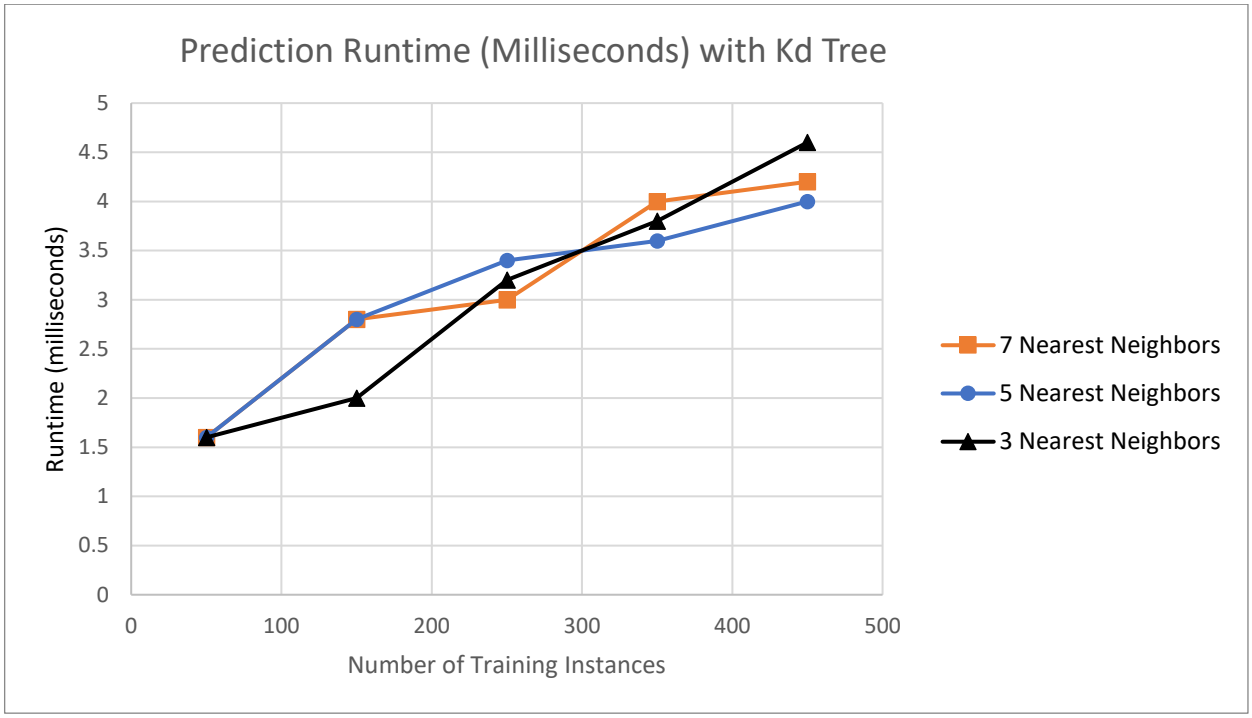

Figure 8. Prediction Runtime (Milliseconds) with K-d Tree

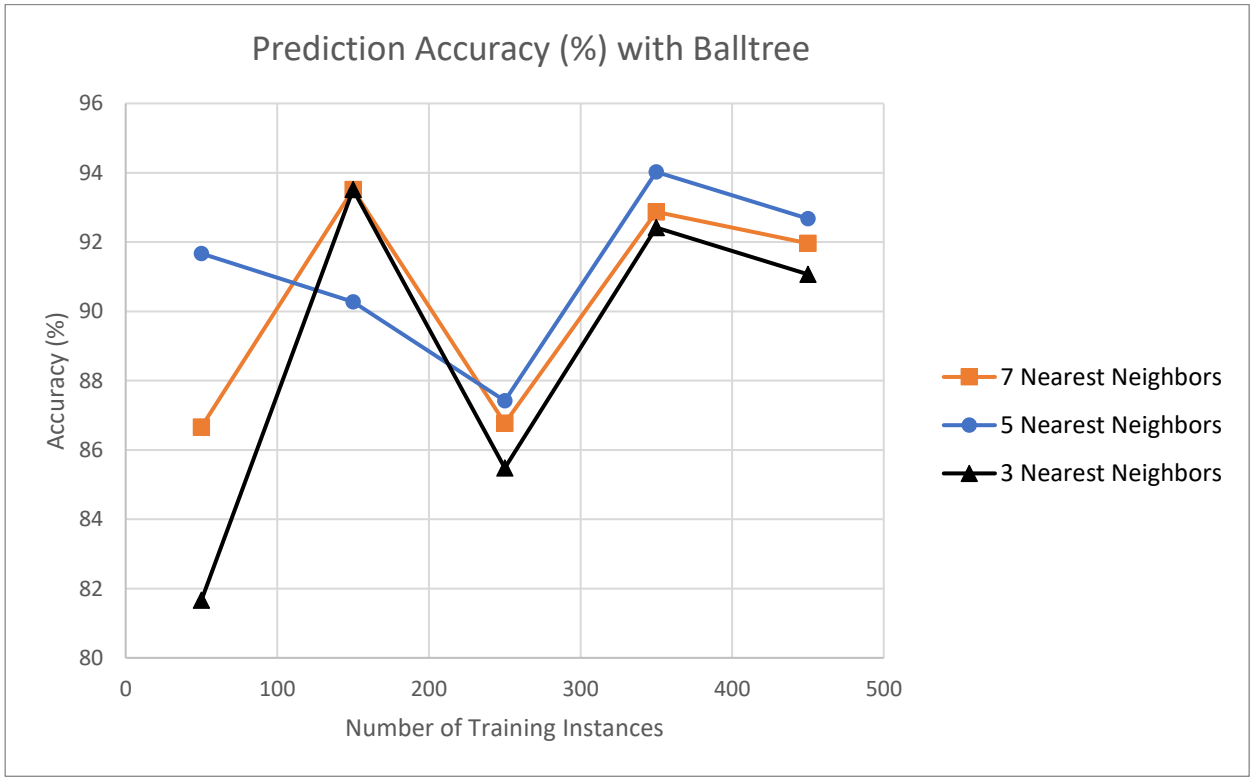

Figure 9. Prediction Accuracy (%) with Ball Tree

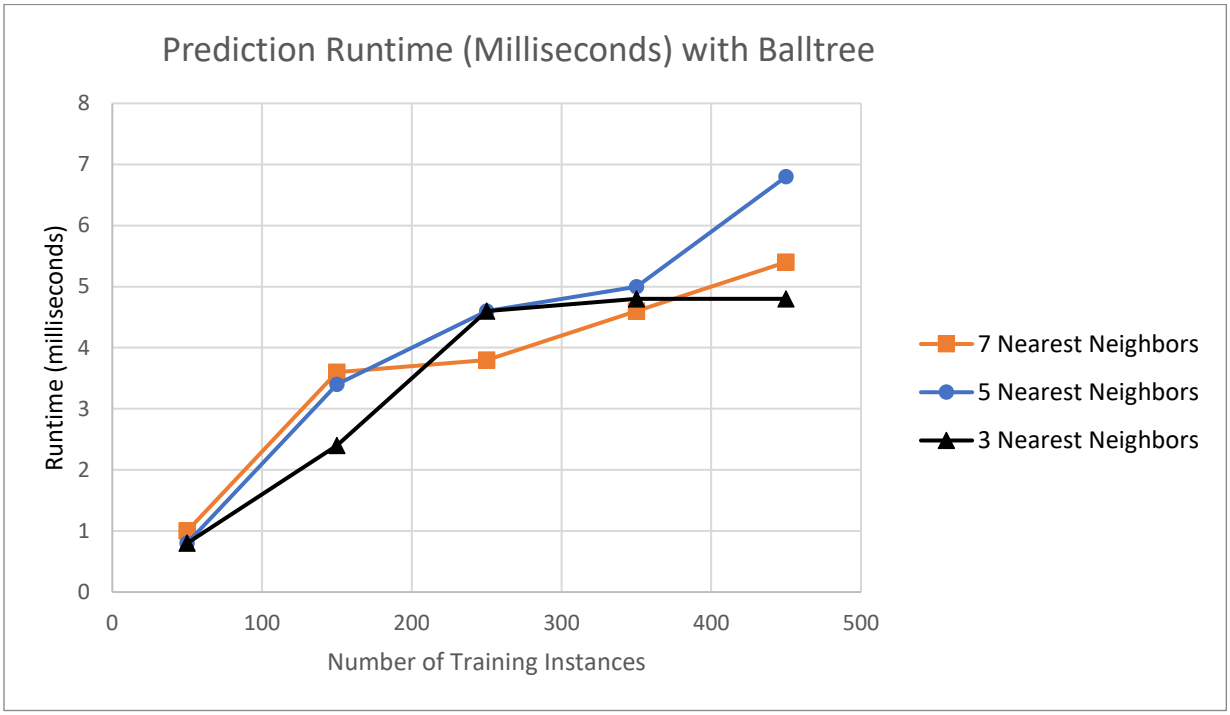

Figure 10. Prediction Runtime (Milliseconds) with Ball Tree

*Each point is an average of 5 different runs

Throughout this exercise, the number of test instances is always roughly a quarter of that of training instances. It seems that the runtime increases as a function the number of training instances. This is because as the number of training instances goes up, so does the number of test instances. As a result, there are more test instances for the model to go through. This is true in both trees. However, for ball tree, the runtime seems to fluctuate more than it does for kd-tree. This is potentially due to the fact that there is a nested loop in the classify function of ball tree which does not exist in that of k-d tree. Another thing worth noting is that ball tree is not always balanced as one child ball could have more data points than the other one (for example, left ball has more points than right ball) even with the use of QuickSelect for median point. This creates an inconsistency in the number for iterations that the inner loop has to go through to get all the points in a given ball. Moreover, the number of nearest neighbors that MaxPQ has to keep track of does not seem to have much of an effect, if at all, on the overall runtime of the model. This is because it does not matter how many nearest neighbors that MaxPQ has to keep tabs on, the model must still go through all the necessary train data points. Overall, it appears that k-d tree runs slightly faster than ball tree.

As for the accuracy of predictions, k-d tree has an edge over its competitor when it comes to stability. While both give predictions that range from 80% to 95% accurate, it appears that k-d tree is more consistent in getting results above 90%. The number of training instances again plays a predominant role here. As the number of training instances decreases so does the accuracy of predictions. While it may not be apparent for higher numbers of training instances like 150 or 250, with 50 train data points, there is an obvious drop in accuracy (ball tree has an outlier here at k = 5, with 92% accuracy). This calls attention to the fact that KNN is an ML model that operates on historical data, or in this case data points that exist in the trees. As a result, more past data means better accuracy for KNN. The number of nearest neighbors plays a more prominent role here as an increase in k seems to entail an increase in accuracy. This is because test points have more neighbors to "look" at before "making" a decision on what they are. However, there might be a caveat against "looking" at too many neighbors as there could be unwanted noise (i.e., train data points that are too far away from a given test point to have an impact on it) that could ultimately increase the number of false decisions. Overall, both trees have great accuracy of predictions and work better with more train data points.

D. Conclusion

With breast cancer being as relevant today as it has always been, it is important that a reliable diagnostic system be put in place. Not only does it help in early detection of cancer, but it also aids physicians in making prompt and correct decisions regarding treatment. While the ML model, implemented by either k-d tree or ball tree, in this exercise is accurate enough on the given dataset, it remains to be seen how functional it is on a much larger set of data since there might be fluctuations or phenomena that are not accounted for in smaller dataset. However, it is clear that KNN works better on larger datasets as it produces better accuracy. As a result, it continues to be one of the more often used ML models out there.

References:

[1] Arnold, Melina, et al. "Current and future burden of breast cancer: Global statistics for 2020 and 2040." The Breast 66 (2022): 15-23.

[2] Osareh, Alireza, and Bita Shadgar. "Machine learning techniques to diagnose breast cancer." 2010 5th international symposium on health informatics and bioinformatics. IEEE, 2010.

[3] Odajima, Katsuyoshi, and Alberto Palacios Pawlovsky. "A detailed description of the use of the kNN method for breast cancer diagnosis." 2014 7th International Conference on Biomedical Engineering and Informatics. IEEE, 2014.

[4] Wayne, Kevin Daniel. “Geometric Applications of BSTs.” Algorithms, Addison-Wesley, Boston, 2016.

[5] Omohundro, Stephen M. Five balltree construction algorithms. Berkeley: International Computer Science Institute, 1989.

[6] Ullrich, Paul & Zarzycki, Colin. (2017). TempestExtremes: A framework for scale-insensitive pointwise feature tracking on unstructured grids. Geoscientific Model Development. 10. 1069-1090. 10.5194/gmd-10-1069-2017.

[7] Rajani, Nazneen, Kate McArdle, and Inderjit S. Dhillon. "Parallel k nearest neighbor graph construction using tree-based data structures." 1st High Performance Graph Mining workshop. Vol. 1. 2015.