Type of the Paper (Article)

On converting the Furthest-Pair-Based Binary Search Tree to a Decision Tree: Experimental Results on Big Data Classification

Ahmad B. A. Hassanat [1]*

[1] IT Department, Mutah University, Karak, Jordan, 61710; hasanat@mutah.edu.jo
* Correspondence: hasanat@ mutah.edu.jo; Tel.: +962-79-889-7192

Abstract: Big Data classification has recently received a great deal of attention due to the main properties of Big Data, which are volume, variety, and velocity. The furthest-pair-based binary search tree (FPBST) shows a great potential for Big Data classification. This work attempts to improve the performance the FPBST in terms of computation time, space consumed and accuracy. The major enhancement of the FPBST includes converting the resultant BST to a decision tree, in order to remove the need for the slow K-nearest neighbors (KNN), and to obtain a smaller tree, which is useful for memory usage, speeding both training and testing phases and increasing the classification accuracy. The proposed decision trees are based on calculating the probabilities of each class at each node using various methods; these probabilities are then used by the testing phase to classify an unseen example. The experimental results on some (small, intermediate and big) machine learning datasets show the efficiency of the proposed methods, in terms of space, speed and accuracy compared to the FPBST, which shows great potential for further enhancements of the proposed methods to be used in practice.

Keywords: machine learning; AI; BST; diameter; algorithms; d-dimensional datasets; decision tree

1. Introduction

Big Data analytic has received a great deal of attention recently, particularly in terms of classification, this is due to the main properties of Big Data; volume, variety, and velocity [1] [2]. Having a large number of examples and various types of data, Big Data classification attempts to seize these properties to obtain better learning models with fast learning/classification [3] [4] [5].

The problem of Big Data classification is similar to the tradition classification problem, taking into consideration the main properties of such data, and can be defined as follows, given a training dataset of n examples, in d dimensions or features; the learning algorithm needs to learn a model that will be able to classify an unseen example E efficiently. In the case of Big Data, where n and/or d are very large values, tradition classifiers become inefficient, for example the K-nearest neighbors (KNN) [6] [7] took weeks to classify some Big Data sets [8].

Recently, we proposed three methods for Big Data classification [9] [10] and [8]. All of these methods employ an approach based on creating a binary search tree (BST), in order to speed up the Big Data classification using a KNN classifier with a smaller number of examples, those which are found by the search process. The real distinction between these methods is in the way of creating the BST. The first uses the furthest-pair of points to classify the examples along the BST, the second uses two extreme points based on the minimum and maximum points found in a dataset, and the third uses the Euclidian norms of examples. Each has its own weakness and strength. However, the common weakness is the use of the slow KNN classifier.

The main goal and contribution of this paper is to improve the performance of the first method-the furthest-pair-based BST (FPBST), by removing the need for the slow KNN classifier, and converting the BST to a decision tree (DT). However, any enhancement made for this method can be easily generalized to the other two methods.

The new enhancement might make the FPBST (and its sisters) a practical alternative for the KNN classifier, since the KNN might be the only available choice in certain cases, such as for example when used for content-based image retrieval (CBIR) [11] and [12].

The FPBST sorts the numeric training examples into a binary search tree, to be used later by the KNN classifier, attempting to speed up the Big Data classification, knowing that searching a BST for an example is much faster than searching the whole training data. This method depends mainly on finding two local examples (points) to create the BST, these points are the furthest-pair of points (diameter) of a set of points in d-dimensional Euclidean feature space [9], these two points are found using a greedy algorithm proposed by [13]. These points are then used to sort the other examples based on their similarity using the Euclidian distance. The training phase of the FPBST ends by creating the BST, which is searched later for a test example E to a leaf-node, where similar examples are found, the KNN classifier is then used to classify E.

Having known that the KNN is slow, we opt for disusing it in this paper, and we do this by converting the resultant BST to a decision tree. To do so, we opt for calculating the probabilities of each class at each node, we calculate the probabilities using four methods, 1) calculating them at the leaf node only, 2) calculating the accumulated probabilities along the depth of the tree, 3) calculating the weighted- accumulated probabilities using the tree's level as a weight, and 4) calculating the weighted- accumulated probabilities using the tree's level as an exponential weight. Therefore, we propose four methods based on these calculations, these four methods stop clustering when having examples of only one class. We propose the fifth method which use the accumulated probabilities of the classes but continues clustering until there is only one example (or similar examples) in a leaf-node.

We further enhanced these five methods by swapping the furthest-pair of points based on the minimum class, so as to obtain a coherent decision tree, where examples of similar classes are stored closer to each other, unlike the FPBST, which use the minimum/maximum norms for this purpose, thus, we propose 10 methods in this paper. These methods/enhancements of the FPBST solve (by default) another related problem associated with the FPBST use of the KNN, which is finding the best k for the KNN [14] and [15]. In this work, there is no need to determine such a parameter since there is no need to use the KNN.

The important of this research stems from the decreasing the size of the resultant tree by trimming the tree where all examples found in a node were of the same class, and this speeds up the training process, reduces the space needed for the resultant tree and increases the speed of testing. In addition to increasing the accuracy of classification as possible as could.

The rest of this paper is organized as follows. Section 2 presents some related methods used for Big Data classification. Section 3 describes the proposed enhancements and the data set used for experiments. Section 4 evaluates and compares the proposed enhancements to FPBST. Section 5 draws some conclusions, shows the limitations of the proposed enhancements and gives directions for future research.

2. Related work

Recently, tremendous efforts have been made to find new methods for Big Data classification, in addition to the FPBST reference [10] used two extreme points, which are the minimum and maximum points found in a dataset to create a BST to sort the examples of a training set, this BST is then searched for a test example to a leaf-node, where similar examples can be found, the KNN classifier is then used to classify a test example. Similarly, reference [8] used the same methodology, except for the way of creating the BST, where it was created based on the Euclidian norms of the training examples. Both methods were very fast, however the accuracy results were slightly less than that of the FPBST [9] in general.

Two recent and interesting approaches proposed by Wang et al. [16] deal with the problem differently, using random and Principal Component Analysis (PCA) techniques to divide the data in order to obtain multivariate decision tree classifiers. Both methods were evaluated on several Big Datasets, the reported accuracy results considering all the datasets used, show that the data partitioning using PCA performs better than that of a random technique used.

Maillo et al. [17] proposed a parallel implementation based on mapping the training set examples, followed by reducing the number of examples that are related to a test sample. The reported results were similar to that of an exact KNN but faster, i.e. about up to 149 times faster than the KNN when tested on 1 million examples; the speed of this parametric method depends mainly on the K neighbors as well as the number of maps used. This work is further improved by almost the same team [18], where they proposed a new KNN based on Spark, which is similar to the Mapping/Reducing technique but with using multiple reducers to speed up the process, the size of the dataset used was up to 11 million examples.

Based on clustering the training set using K-means clustering algorithm, Deng et al. [19] proposed two methods to increase the speed of KNN, the first used random clustering and the second used landmark spectral clustering, when finding the related cluster, both utilize the KNN to test the input example with a smaller set of examples. Both algorithms were evaluated on 9 Big Datasets showing reasonable approximations to the sequential KNN, the reported accuracy results were dependent on the number of clusters used.

Another clustering approach is utilized recently by Gallego et al. [20], who proposed two clustering methods to accelerate the speed of the KNN, both are similar; however, the second is an enhancement of the first, where a cluster augmentation process is employed. The reported average accuracy of all the Big Datasets used was in the range of 83 to 90% depending on the K-neighbors and number of clusters used. The performance of both methods has improved significantly when the Deep Neural Networks has been employed for learning a suitable representation for the classification task.

Most of the proposed work in this domain is based on divide and conquer approach, this is a logical approach to use with Big Datasets, and therefore, most of these approaches are based on clustering, splitting, or partitioning the data to turn and reduce the huge size to a manageable size that can be used for and efficient classification. One major problem associated with such approaches is that the determination of the best number of clusters/parts, sine more clusters means fewer examples, and therefore faster testing. However, fewer examples also means less accuracy, as the examples found in a specific cluster might not be related to the tested example. On the contrary, few clusters indicate a large number of examples per each, which increases the accuracy but slows down the classification process if the KNN is used.

Similar to references [9] [10] [8] there exist extensive literature on tree structures such as k-d trees [21], metric trees [22], cover trees [23], and other related work such as [24], [25]. Regardless of the plethora of the proposed methods in this domain, there is still room for improvement in terms of accuracy and time consumed for both training and testing stages. And this work is nothing but an attempt to improve the performance of one of these methods.

3. Furthest-pair-based decision trees (FPDT)

This section describes and illustrates the proposed methods, in addition to describing the data used for evaluation and experiments.

3.1. Methods

The main improvement of the FPBST [9] includes the unemployment of the standard KNN algorithm as described by [6] and [7], which is time-consuming particularly when classifying Big Data. In this paper, we propose the use of the probabilities of the classes found in the leaf-nodes to decide the class of a test example, without having to use the slow KNN, even if there are a small number of examples found in a leaf-node. We keep the same functionality of the binary search tree (BST), which is employed to sort the examples (points) of machine learning datasets in a way that facilitates the search process. This BST sorts all the examples taken from a training dataset based on their distances from two local points (P1 and P2), which are two examples from the training dataset itself, and they vary based on the host node and the level/location of that node in a BST.

The FPBST builds its BST by finding the furthest points P1 and P2 [13], assuming that the furthest points are the most dissimilar points, and therefore, are more likely to be belonging to different classes. Thus, sorting other examples based on their distances to these points might be a good choice, as similar examples are sorted nearby, while dissimilar examples are sorted faraway in the created BST.

Similarly to the FPBST, the training phase of the proposed method (FPDT) creates a binary decision tree (DT), which speeds up searching for a test example comparing to the unacceptable time an exhaustive search, particularly when classifying Big Datasets. We use the same Euclidian distance metric (ED) for measuring distance, to compare the results of the proposed method to those of the FPBST.

While creating the DT, we calculate the probability of each class to occur in each node, her we opt for several options:

1. Accumulate the classes' probabilities by adding the parent's probabilities to its children's; we call this method decision tree 0 (DT0).
2. Accumulate the probabilities by adding the parent's probabilities to its children's; and weighting these probabilities by the level of the node, assuming that the more we go deeper in the tree, the more likely we reach to a similar example(s), this is done by multiplying the tree level by the classes' probabilities at a particular node; we call this method decision tree 1 (DT1), and is shown in Algorithm 1 as (Dtype =1).
3. Accumulate the classes' probabilities by adding the parent's probabilities to its children's; and weighting these probabilities exponentially, for the same reason in 2, but with higher weight, this is done by multiplying the classes' probabilities by 2 to the power of the tree's level at a particular node; we call this method decision tree 2 (DT2), and is shown in Algorithm 1 as (Dtype =2).
4. No accumulation of the probabilities, we use just the probabilities found in a leaf node; we call this method decision tree 3 (DT3), and is shown in Algorithm 1 as (Dtype !=3).
5. Similarly to DT0 (normal accumulation), but the algorithm continues to cluster until there is only one or a number of similar examples in a leaf-node, this is done even if all the examples of a current node belong to the same class. While DT0-DT3 stop the recursive clustering when all the examples of the current node are belonging to the same class, and consider the current class as a leaf-node, we call this method decision tree 4 (DT4), and is shown in Algorithm 1 as (Dtype =4).

The idea behind accumulating the probabilities is to remove the effect of unbalanced datasets, as some datasets contains more examples of a specific class than the other classes, and this will increase the probability of the dominant class, since it is calculated in the root node and accumulated along the depth of the tree, so by moving deeper, less number of the dominant examples remain.

Algorithm (1) shows the pseudo code for the training phase of the FPDT method, which works well for DT0, DT1, DT2, DT3 and DT4 depending on the input (Dtype), and Algorithm 2 shows the pseudo code for the testing phase of the FPDT method, which is the same for DT0, DT1, DT2, DT3 and DT4, as these methods differ in the way of creating the decision tree only, i.e. the training phase.

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Algorithm 1:   Training Phase (DT building) of FPDT.
Input: Numerical training dataset DATA with n FVs and d features, and DT type (Dtype)
Output: A Root pointer to the FPBST.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Create a DT Node → RootN
RootN.Examples←FVs //all indexes of FVs from the training set
(P1, P2) ← Procedure Furthest(DATA ←RootN.Examples, n) //hill climbing algorithm [13]
if   EN(P1) > EN(P2) swap(P1, P2)
```

```
RootN.P1← P1
RootN.P2← P2
RootN.Left=Null
RootN.Right=Null
Procedure BuildDT(Node←RootN)
    for each $FV_i$ in Node, do
        D1←ED($FV_i$, Node.P1)
        D2←ED($FV_i$, Node.P2)
        If (D1<D2)
            Add index of $FV_i$ to Node.Left.Examples
        else
            Add index of $FV_i$ to Node.Right.Examples
    end
    if (Node.Left.Size==0 or Node.Right.Size == 0)
        return //this means a leaf node
      end
      (P1, P2) ← Furthest(Node.Left.Examples, size(Node.Left.Examples))//work on the left child
      if   EN(P1) > EN(P2) swap(P1, P2)
      Node.Left.P1← P1
      Node.Left.P2← P2
      Node.Left.ClassP [numclasses]={0}//initialize the classes' probabilities to 0;
      for each i in Node.Left.Examples do
          Node.Left.ClassP [DATA.Class[i]]++//histogram of classes at Left-Node
      bool   LeftMulticlasses =false; //check for single class to prune the tree
      if there is more than one class at Node.Left.ClassP
        LeftMulticlasses=true;
      if (Dtype ==4)          //no pruning if chosen
        LeftMulticlasses=true   //even if there is only one class in a node=> cluster it further
      for each i in numclasses do    //calculate probabilities of classes at the left node
        Node.Left.ClassP [i]= Node.Left.ClassP [i]/ size(Node.Left.Examples)
        //weigt if any node->left->level
      if (Dtype ==1)   //increase the probabilities by the increased level
         for each i in numclasses do
            Node.Left.ClassP [i]= Node.Left.ClassP [i]* Node.Left.level
      if (Dtype ==2)    //increase the probabilities exponentially by the increased level
         for each i in numclasses do
            Node.Left.ClassP [i]= Node.Left.ClassP [i]* $2^{Node.Left.level}$
      if (Dtype!=3) //do accumulation for probabilities, if 3, use   just the probabilities in a leaf node
          for each i in numclasses do
          Node.Left.ClassP [i]= Node.Left.ClassP [i]+ Node.ClassP [i]
      Node.Left.Left =NULL;
      Node.Left.Right=NULL;
      Repeat the previous steps on Node.Right
```

```
            if (LeftMulticlasses)
              BuildDT (Node.Left)
            if (RightMulticlasses)
              BuildTree(Node.Right)

end Procedure
return RootN
-----------------------------------------------------------------
-----------------------------------------------------------------
Algorithm 2:   Testing Phase of FPDT.
Input: test dataset TESTDATA with $n$ FVs and $d$ features
Output: Testing Accuracy Acc.
-----------------------------------------------------------------
Acc←0
for each $FV_i$ in TESTDATA do
            Procedure GetTreeNode(Node←RootN, $FV_i$)
              D1←ED(FV[i], Node.P1)
              D2←ED(FV[i], Node.P2)
              if (D1<D2 and Node.Left)
                     return GetTreeNode (Node.Left, $FV_i$)
              else if (D2 ≤ D1 and Node.Right)
                     return GetTreeNode (Node.Right,$FV_i$)
                        else
                            return Node
                        end
       end Procedure GetTreeNode
       class ← argmax(Node.ClassP)// returns the class with the maximum probability
            if class== Class($FV_i$)
                     Acc← Acc+1
end
Acc← Acc/n
return Acc
-----------------------------------------------------------------
```

The training phase of the FPBST and the new DT0-DT4 use the Euclidian norm to regularize the resultant tree by swapping P1 and P2 if the norm of P2 is less than that of P1 (Line 22 in Algorithm 1). This is normally done to let the examples, which are similar to the point of the least norm to be sorted to the left side of the tree, and the others to be sorted to the right side of the tree, so as to have similar examples adjacent as possible as could in the resultant BST. Having know that the Euclidian norm is sensitive to the negative numbers (negative and positive similar numbers result the same Euclidian norm), the examples with many zeros or similar repeated numbers [8], we opt for an alternative of the norm to decide which goes to left and which goes to right. Here we propose the use of the class of the example, so we check the classes of P1 and P2 to see if P2 has the minimum class, if yes, we swap P1 with P2, otherwise they remain as they are. Such a swap allows for regularizing the resultant decision tree with the minimum cost, as creating the norm cost extra O(d) each time, while obtaining the class of an example costs O(1), and at the same time we get more coherent trees in terms of the classes distribution, since the examples of minimum class are forced to be sorted to the left and those with the maximum class are sorted to the right, this might have a good effect on the probabilities of the classes. This improvement is applied on all the proposed DT0-DT4 making new decision trees DT0+, DT1+, DT2+, DT3+ and DT4+.

Similarly to the FPBST, The time complexity of training phase to build the decision tree (DT) by the proposed methods (DT0-DT4 and DT0+ to DT4+) is

$$T(n, d) = O(cnd \log n) \tag{1}$$

where (cnd) is the time consumed to find the approximate furthest points, as the constant c is the number of iterations needed to find the approximate furthest points, which is found experimentally to be in the range of 2 to 5 [13]. The (log n) time is consumed along the depth of the DT.

An extra (2nd) time is consumed by comparing each example or feature vector (FV) to the local furthest points (P1 and P2). This time can be added to c to make it in the range of 4 to 7, however, c is still a constant and the overall time complexity can be asymptotically approximated to

$$T(n, d) = O(nd \log n) \tag{2}$$

and if n >> d, the time complexity can be further approximated to

$$T(n, d) = O(n \log n) \tag{3}$$

The space complexity can be defined by

$$S(n, d) = O(n \log n) \tag{4}$$

where the space consumed (S) is a function of n and d, which similar to the size of a normal BST.

The test phase of the proposed method (Algorithm 2) is the same for all the DTs, as it searches the created DT for a test example starting from the root node to a leaf node, where similar example(s) are supposed to be there. However, it is different from the test phase algorithm of the FPBST, where KNN algorithm is employed to classify the test example using those found in a leaf-node. The proposed DTs have no need to use the KNN, because the leaf-node has become able to decide the class of the tested example based on the pre-calculated probabilities it has, since the name (decision tree) suggests. Disusing the KNN with the proposed DTs allows for more speed. Therefore, the time complexity of the test phase of the proposed DTs for each tested example is

$$T(n, d) = O(2d \log n) \tag{5}$$

where the (2d) time is consumed by the calculation of the ED, which costs d time for each comparison with either P1 or P2. And the (log n) time is consumed along the depth of the BST, which is about (log n) on average.

And if n>>d, the d time can be ignored making the testing time

$$T(n, d) = O(\log n) \tag{6}$$

3.2. Implementation example

To further explain the proposed Dts, we implement some of them to create decision trees to be compared with the BST of the FPBST. For this end, we used a small synthesized data set for illustration purposes. The synthesized dataset used consists of two hypothetical features (X1 and X2) and two classes (0 and 1) having 20 examples as shown in Table 2 and illustrated in Figure 1.

Table 1. A hypothetical training data sample to exemplify the resultant BST of the FPBST, as well as the decision trees of the proposed methods.

#Example	X1	X2	Class	Euclidean Norms
0	4	3	0	5.0
1	2	5	0	5.4
2	2	4	0	4.5
3	4	4	0	5.7

4	3	6	0	6.7
5	1	0	0	1.0
6	1	3	0	3.2
7	3	1	0	3.2
8	3	2	0	3.6
9	4	6	0	7.2
10	4	5	1	6.4
11	3	7	1	7.6
12	8	6	1	10.0
13	9	7	1	11.4
14	3	4	1	5.0
15	5	7	1	8.6
16	8	3	1	8.5
17	3	5	1	5.8
18	4	8	1	8.9
19	8	8	1	11.3

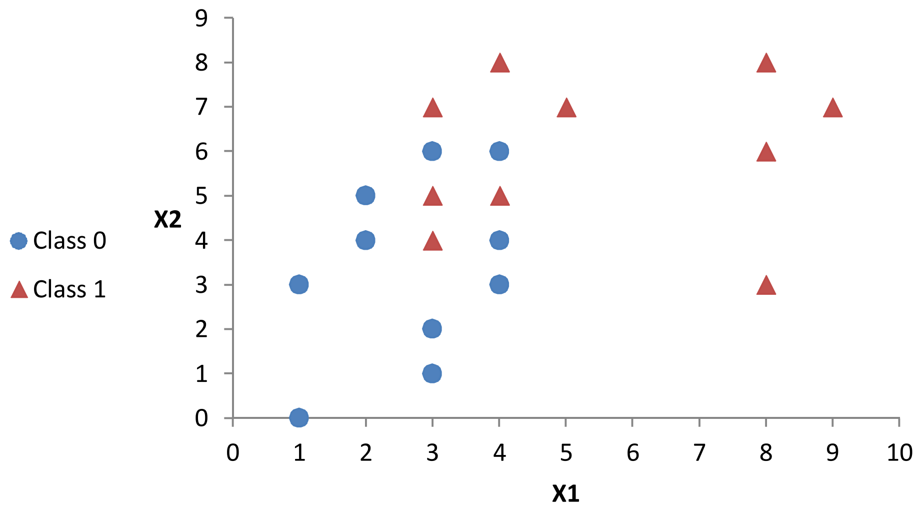

Figure 1. A visual illustration of the synthesized dataset obtained from Table 1.

If we apply the FPBST on the synthesized dataset we get the BST illustrated in Figure 2, and when applying the DT0, DT0+, DT1 and DT1+ on the same dataset we get the decision trees illustrated in Figures 3, 4, 5, and 6 respectively.

The purpose of these figures is not to prove anything, as we cannot draw significant conclusions from such weak evidence (the very small data set in Table 1). However, they are meant to show how the proposed DTs are constructed comparing to the BST of the FPBST. It is interesting to note that calculating the furthest-pair of points is an approximate algorithm and might not obtain the same pair of points always, as seen in the Figures 2 and 6, where the furthest pair was (5, 19), while the furthest pair was (5,13) in Figures 3, 4 and 5. Both of the pairs have the maximum distance in the dataset which is 10.63. In addition, we can note the smaller size of the DTs in (Figures 3-6) and the shallow nodes comparing to the BST in Figure 2, this is because the DTs stop the recursive process to create child-nodes when the node is pure, i.e. all the examples hosted belong to the same class. One exception is the DT4 and DT4+, which carry on sorting the examples until there is only one example (or similar examples) in a leaf-node, we mean by similar examples, those who share the same Euclidian distance to a reference point. Also we can note the difference between the DTs and the DT+s, for example, the point 14 is classified as class 0 in Figure 3, while it belongs to class 1, this is because its norm = 5, while the other point (1) sharing the same parent node has a norm =5.4, according to DT0, Point 14 goes to the left and Point 1 goes to the right, if the DT0 was not calculating accumulated probabilities this should not make a big difference, but since such type of probabilities is used by the DT0 and DT0+ without giving a higher weight to the deeper levels we get such a classification error. However, this situation is not happening when using DT1 and DT1+, because the tree level is used to weight the probabilities.

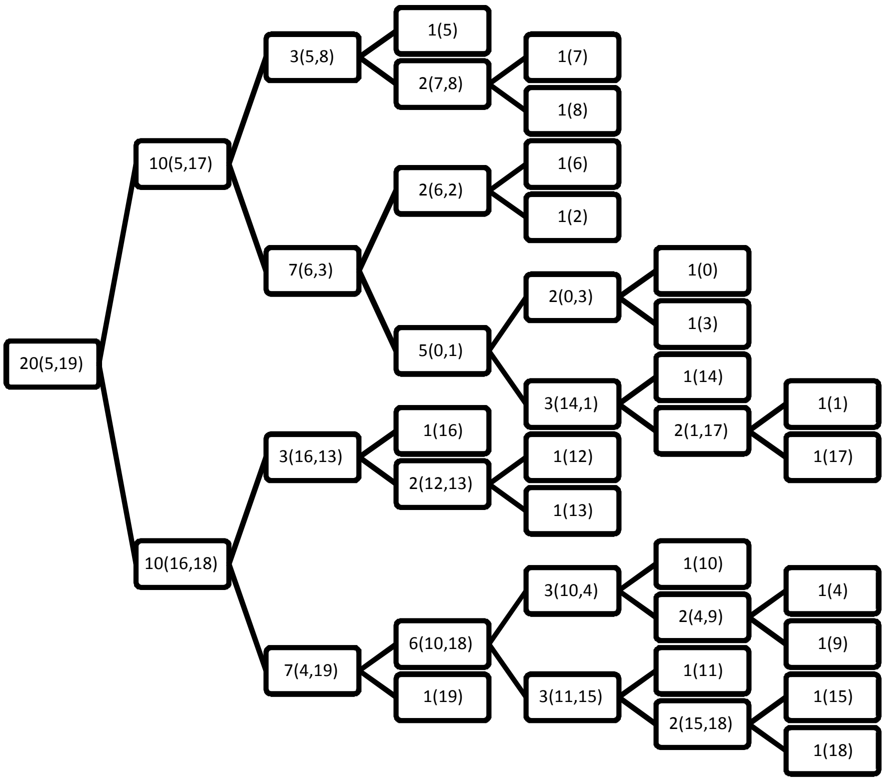

Figure 2. The resultant BST after applying the training phase of the FPBST on the sample data from Table 1. The number outside the brackets is the counter of the examples hosted by each node, and those inside the brackets are the index of the examples in a leaf node, or the furthest points (P1 and P2) otherwise.

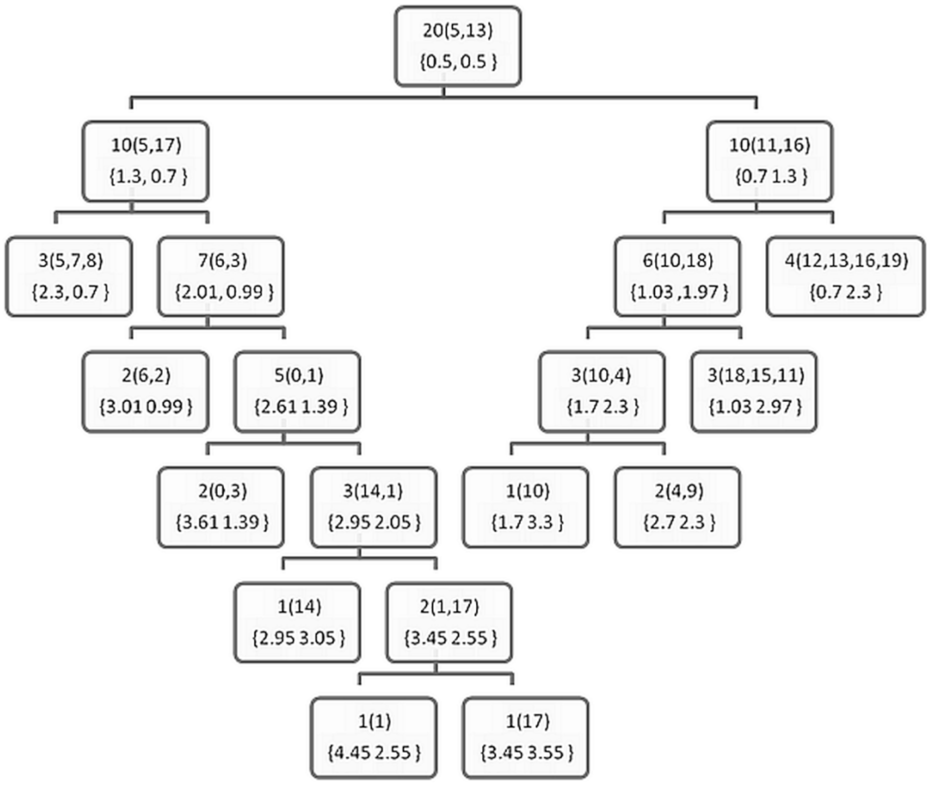

Figure 3. The resultant decision tree after applying the training phase of the DT0 on the sample data from Table 1. The number outside the rounded brackets () is the counter of the examples hosted by each node, and those inside the rounded brackets are the index of the examples in a leaf node, or the furthest points (P1 and P2) otherwise. The numbers in the curly brackets {} shows the probabilities of the classes at each node.

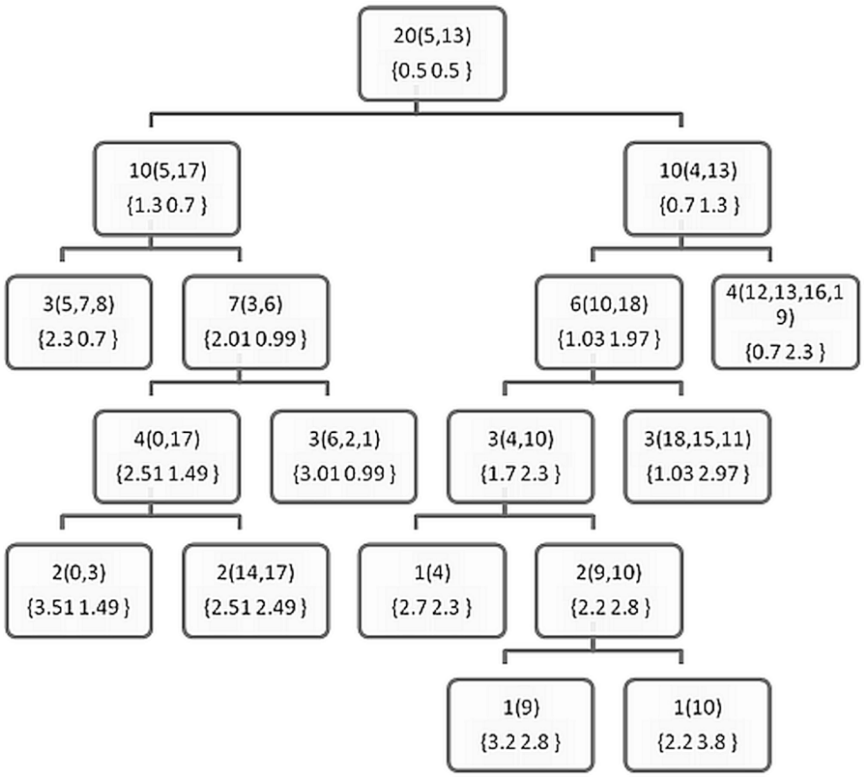

Figure 4. The resultant decision tree after applying the training phase of the DT0+ on the sample data from Table 1. The number outside the rounded brackets is the counter of the examples hosted by each node, and

those inside the rounded brackets are the index of the examples in a leaf node, or the furthest points (P1 and P2) otherwise. The numbers in the curly brackets {} shows the probabilities of the classes at each node.

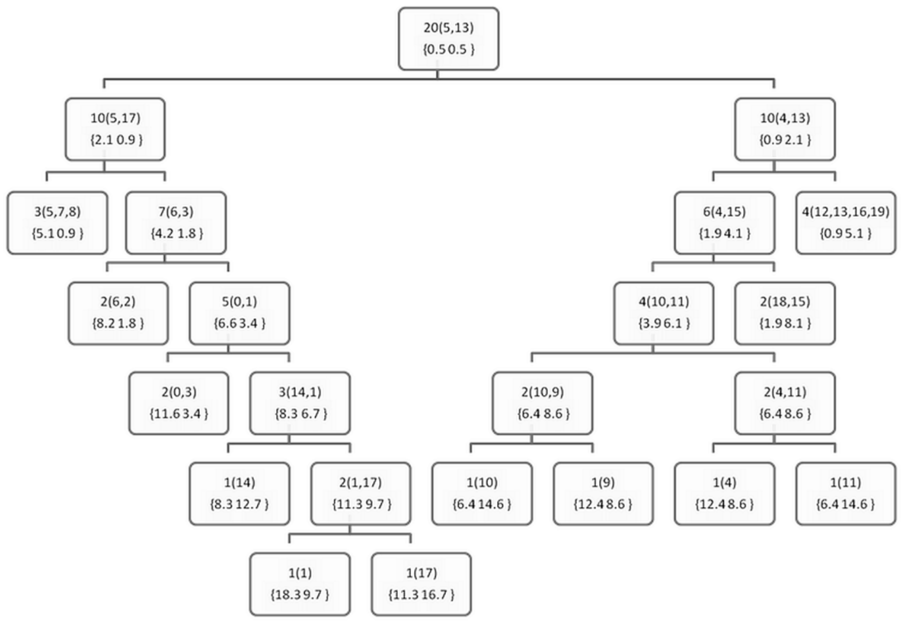

Figure 5. The resultant decision tree after applying the training phase of the DT1 on the sample data from Table 1. The number outside the rounded brackets is the counter of the examples hosted by each node, and those inside the rounded brackets are the index of the examples in a leaf node, or the furthest points (P1 and P2) otherwise. The numbers in the curly brackets {} shows the probabilities of the classes at each node.

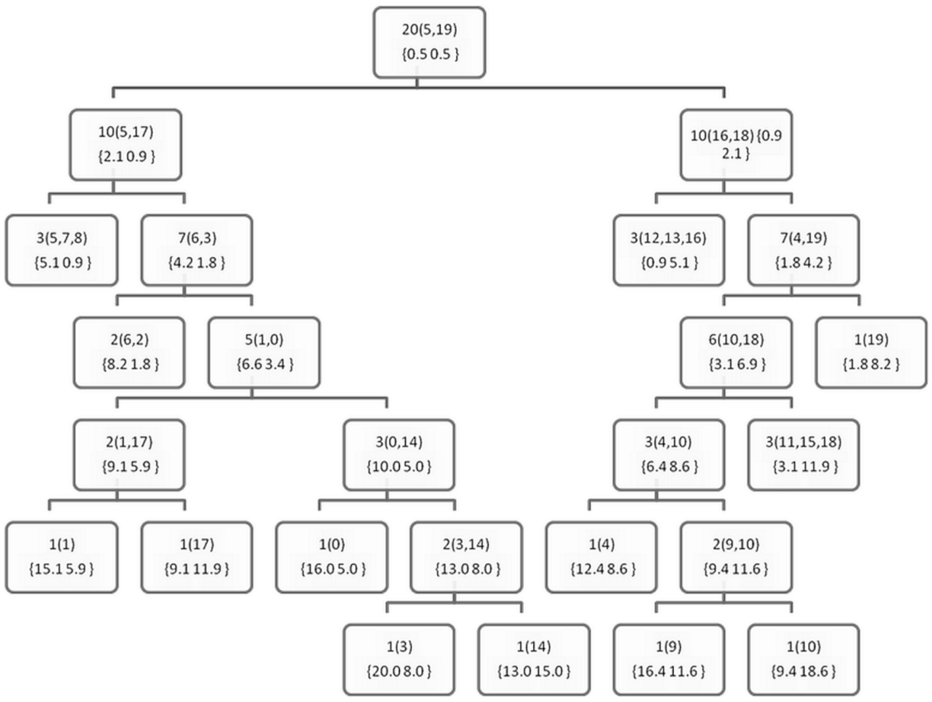

Figure 6. The resultant decision tree after applying the training phase of the DT1+ on the sample data from Table 1. The number outside the rounded brackets is the counter of the examples hosted by each node, and those inside the rounded brackets are the index of the examples in a leaf node, or the furthest points (P1 and P2) otherwise. The numbers in the curly brackets {} shows the probabilities of the classes at each node.

3.3 Data

In order to evaluate the proposed methods and compare the results to the FPBST on Big Data classification, we use some of the well-known machine learning datasets, which are used by state-of-the-art work in this domain. These datasets are freely available for download from either the support vector machines library (LIBSVM) Data [26] or the UCI Machine Learning Repository [27]. The datasets used are of different dimensions, sizes, and data types, such diversity is important to evaluate the efficiency of the proposed method in terms of accuracy and time consumed.

All datasets used contain numeric data, i.e. real numbers and/or Integers. The sizes of these datasets are in the range of 625 to 11000000 examples; the dimensions are in the range of 4 to 5000 features. Table 2 shows the descriptions of the datasets used.

Table 2. Description of datasets used for evaluation and comparison of the proposed methods

Dataset	Size	Dimensions	Type	#Class
HIGGS	11000000	28	Real	2
SUSY	5000000	18	Real	2
Poker	1025010	11	Integers	10
Covtype	581012	54	Integers	7
Mnist	70000	784	Integers	10
Connect4	67557	42	Integers	3
Nist	44951	1024	Integers	26
LetRec	20000	16	Real	26
Homus	15199	1600	Integers	32
Gisette	13500	5000	Integers	2
Pendigits	10992	16	Integers	10
Usps	9298	256	Real	10
Satimage	6435	36	Real	6
Abalone	4177	8	Real	29
Climate	1178	18	Real	2
German	1000	24	Integers	2
Blood	748	4	Integers	2
Australian	690	14	Real	2
Cancer	683	9	Integers	2
Balance	625	4	Integers	3

4. Results and discussions

To evaluate the proposed methods (DT0-DT4 and DT0+-DT4+), we programmed both Algorithms 1 and 2 using MS VC++.Net framework, version 2017, and conducted several classification experiments on all the datasets described in the data section. We utilized a personal computer with the following specifications:

- Processor: Intel® Core™ i7-6700 CPU @ 340GHz
- Installed memory (RAM): 16.0 GB
- System type: 64-bit operating system, x64-based processor, MS Windows 10.

Table 3 shows the characteristics of the BT built using the proposed DTs comparing to that of the FPBST, her we used one dataset (Poker), as being one of the largest datasets and to save space for this paper.

Table 3. Some specifications of the resultant BST of the FPBST compared to the resultant DTs after applying the proposed FPDTs on the Poker dataset (training phase).

Method	Number of nodes	Number of Leaves	Maximum Depth	Total examples in all Leaves	Min number of examples in a Leaf	Max number of examples in a Leaf
FPBST	2045541	1022771	30	1025010	1	3
DT0	1433617	716809	29	1025010	1	49
DT1	1433089	716545	29	1025010	1	80
DT2	1433631	716816	29	1025010	1	71
DT3	1432131	716066	30	1025010	1	47
DT4	2045541	1022771	30	1025010	1	3
DT0+	1440047	720024	29	1025010	1	99
DT1+	1439295	719648	29	1025010	1	42
DT2+	1439113	719557	30	1025010	1	56
DT3+	1441107	720554	30	1025010	1	44
DT4+	2045541	1022771	30	1025010	1	3

As can be noted in Table 3, the maximum depth of the resultant BST and DTs is not much larger than log2 (1025010)=19.97, this of course increases the speed of the test phase for all the proposed methods including the FPBST. Although the number of nodes in a full BST is typically (n log n), and therefore should be around 20,421,879, we found it much less than that for all methods, this is due to the resultant BST and DTs being not full binary trees. It is interested to note that the number of nodes in the proposed DTs is significantly less than that of the BST; this is related to the number of hosted examples in the leaf-nodes, as it is higher in the DTs than the BST, i.e. the lower the number of nodes, the higher the number of examples per leaf-node. This is because the DTs stop the recursive process earlier, mainly, when all the existing examples are belonging to only one specific class. One exception is the DT4 and DT4+, obviously because both of them do not stop the recursive process and carry on creating nodes until there is only one example per each leaf-node, or similar examples.

The relatively small size of the DT created by the proposed DT0-DT3 and DT0+- DT3+ shall serve two purposes, 1) decreasing the space needed for the tree, and 2) speeding up the classification process, since searching a smaller tree is faster than a larger one. This is also complying with the number of leaf-node, as being significantly smaller than that of the FPBST, DT4 and DT4+.

In this paper, we compare the performance of the proposed methods to that of the FPBST, as the goal of this paper is to improve the performance of the FPBST, in terms of speed, space used and classification accuracy. For this end, we evaluated the proposed methods DT0-DT4 by employing them to classify the machine learning datasets stated in Table 2, using 10-fold cross-validation, so as to be able to compare their performances to that of the FPBS.

Since we used a different hardware with different computation powers, which might significantly affects the comparison in terms of time consumed, we opt for reporting the speed-up factor of each method similarly to [17] [13]. We calculate the speed-up factor by considering the ratio of the time consumed by the FPBST classifier to that of the proposed methods on the same dataset used and same examples tested as follows

$$\text{Speedup}(X, D) = \frac{T(\text{FPBST}, D)}{T(X, D)} \tag{7}$$

where D is the dataset tested, X is the method that we wish to calculate its speedup factor, and T is the time function, which returns the time consumed by the method X on the dataset D.

The accuracy comparison results are shown in Table 4. Tables 5 and 6 show the time consumed in the training and testing phases respectively, while Table 7 shows the speed-up comparison results.

Table 4. Accuracy results of the proposed methods DT0-DT4 compared to that of the FPBST, using 10-fold-cross validation

Dataset	FPBST	DT0	DT1	DT2	DT3	DT4
Abalone	0.4990	**0.5374**	0.5338	0.4906	0.5122	0.5326
Australian	0.6435	0.6667	0.6725	0.6203	0.6392	**0.6899**
Balance	0.8258	0.8226	**0.8323**	0.7855	0.8210	0.8290
blood	0.6784	0.7662	**0.7811**	0.7189	0.7135	0.7716
Cancer	0.9618	0.9574	0.9574	0.9544	0.9559	**0.9647**
Climate	0.8722	0.9148	0.9148	0.8537	0.8796	**0.9167**
German	0.6550	**0.7120**	0.7050	0.6240	0.6460	0.7110
LetRec	0.7897	0.7143	0.7379	0.7841	**0.7935**	0.7476
Usps	0.8631	0.7758	0.8179	**0.8665**	0.8614	0.8222
Satimage	**0.8672**	0.8342	0.8566	0.8594	0.8617	0.8588
Pendigits	0.9630	0.8779	0.9196	0.9625	**0.9678**	0.9392
Gisette	0.8907	0.8372	0.8541	0.8867	**0.8910**	0.8730
Mnist	0.8527	0.7720	0.8055	**0.8553**	0.8527	0.8135
Homus	0.4508	0.4289	0.4481	0.4560	**0.4572**	0.4386
Nist	0.4795	0.4507	0.4684	0.4853	**0.4858**	0.4605
Connect4	0.6222	0.6613	**0.6743**	0.6216	0.6197	0.6659
Covtype	0.9314	0.7752	0.8318	0.9313	**0.9315**	0.8533
Poker	0.5372	0.5881	**0.5889**	0.5366	0.5351	0.5870
SUSY	0.7098	0.7547	**0.7651**	0.7103	0.7093	0.7599
HIGGS	0.5860	0.5998	**0.6062**	0.5857	0.5856	0.6010
Average	0.7339	0.7224	0.7386	0.7294	0.7360	0.7418

As can be seen from Table 4, one or more of the proposed methods DT0-DT4 slightly outperform the FPBST in terms of accuracy when testing on all datasets except for the Satimage, which works better with the FPBST, however the difference is not significant (less than 1%), and it might due to randomness of the train/test examples, on average, we can see that the DT1, DT3 and DT4 perform slightly better than the FPBST. The maximum average classification accuracy is attributed to the DT4; this is due to the nature of the DT that created by the DT4, which continues the recursive process until there is only one class or similar classes per leaf-node. We are not favoring the DT4 as its size is similar to the BST of the FPBST, however its accuracy is not significantly higher than the other DTs and the FPBST, for example the DT3 outperforms all methods in terms of the number of datasets tested. We can say with some confidence that the proposed approach (using the decision tree instead of the KNN-regardless the creation method of the decision tree) performs well on all the evaluated datasets, and this performance is almost similar to the FPBST in some cases or slightly better in other cases.

Table 5. Time (ms) consumed by the proposed methods DT0-DT4 to build their DTs compared to that of the FPBST to build its BST, this is the average training time of the 10 folds.

Dataset	FPBST	DT0	DT1	DT2	DT3	DT4
Abalone	196	170	163	164	**162**	156
Australian	45	44	**37**	39	39	42
Balance	15	**8**	**8**	9	**8**	10
blood	19	**13**	**13**	**13**	**13**	15
Cancer	29	11	9	**8**	9	20

Climate	29	21	21	**20**	**20**	30
German	56	62	**61**	68	62	70
LetRec	1595	1249	1151	**1142**	1176	1338
Usps	8096	**8030**	8119	8084	8098	9922
Satimage	860	743	769	**771**	774	958
Pendigits	910	**574**	576	578	581	882
Gisette	76157	60297	**58053**	59413	58600	72881
Mnist	142712	118787	118678	117771	**116226**	141527
Homus	83337	76153	**72626**	73118	73821	77758
Nist	118746	99569	**98802**	100926	98987	106223
Connect4	6860	5832	5854	5408	**5323**	5857
Covtype	94265	73359	**71481**	75018	72736	90682
Poker	74536	70164	66805	68613	**61009**	75957
SUSY	923749	955593	948978	959964	**906365**	1017843
HIGGS	2974260	311 7121	**2855369**	2958233	2951326	2939329

It is interesting to note from Table 5 that the proposed DT0-DT3 consumed less time in general than the FPBST and the DT4, this is due to the smaller decision trees created by these methods, However, the time saved while building the decision tree by DT0-DT3 is not significant on some datasets, this is due to the extra calculations of the probabilities of each class for each dataset. It is also interesting to note the time consumed by the DT4 is almost similar to that of the FPBST and sometimes longer; this is because it has a similar tree size to that of the FPBST, with extra time for calculating the probabilities.

Table 6. Time (ms) consumed by the FPBST to test the entire test examples compared to that of the proposed methods DT0-DT4, this is the average test time of the 10 folds.

Dataset	**FPBST**	**DT0**	**DT1**	**DT2**	**DT3**	**DT4**
Abalone	13.5	10.5	10.3	9.8	10.1	**8.6**
Australian	3.1	2.7	**2.0**	2.2	**2.0**	**2.0**
Balance	1.5	0.9	**0.5**	1.0	**0.5**	0.8
blood	1.9	1.3	**1.0**	**1.0**	1.1	**1.0**
Cancer	3.1	1.1	**0.7**	**0.7**	0.8	1.3
Climate	2.1	1.4	1.2	**1.0**	1.1	1.9
German	3.5	**3.4**	3.5	3.8	**3.4**	4.0
LetRec	85.2	69.4	60.2	**59.8**	60.9	67.0
Usps	390.3	361.4	365.4	366.3	**361.2**	441.9
Satimage	47.9	38.7	38.7	**38.0**	38.8	48.8
Pendigits	53.9	33.6	33.6	**32.3**	34.8	46.8
Gisette	3752	2920	**2804**	2865	2884	3480
Mnist	6869	5631	5563	**5474**	5571	6591
Homus	3965	3520	3492	3695	3540	**3443**
Nist	5686	5088	5728	5236	**4893**	5058
Connect4	395	**304**	308	**304**	308	324
Covtype	4935	**3561**	3719	3662	3764	4326
Poker	4243	3834	3546	3641	**3381**	3924
SUSY	46906	46521	47801	**46027**	46430	50604

HIGGS	164652	161623	**142220**	151414	149363	164853

As can be seen from Table 6, the consumed time in the testing phase for DT0-DT3 is less than that of the FPBST, this is due to the smaller size of these threes, and the disuse of the KNN classifier, it is interesting to note that the DT4 speed in the testing phase is almost similar to that of the FPBST, this is due the large and equal size of their trees. It is also interesting to note that there is no significant difference in the time consumed by the proposed DT0-DT3 in the testing phases, since they are almost the same except for the method of calculating the probability for each class.

Table 7. Speed-up results (training and testing phases) of FPBST compared to that of the proposed methods DT0-DT4.

	DT0 Speed		**DT1 Speed**		**DT2 Speed**		**DT3 Speed**		**DT4 Speed**	
Dataset	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
Abalone	1.15	1.29	1.20	1.31	1.19	1.38	1.21	1.34	1.25	1.57
Australian	1.03	1.15	1.21	1.55	1.17	1.41	1.16	1.55	1.06	1.55
Balance	1.82	1.67	1.80	3.00	1.78	1.50	1.89	3.00	1.50	1.88
blood	1.40	1.46	1.44	1.90	1.44	1.90	1.47	1.73	1.22	1.90
Cancer	2.72	2.82	3.23	4.43	3.59	4.43	3.30	3.88	1.46	2.38
Climate	1.40	1.50	1.41	1.75	1.42	2.10	1.45	1.91	0.97	1.11
German	0.90	1.03	0.92	1.00	0.82	0.92	0.90	1.03	0.80	0.88
LetRec	1.28	1.23	1.39	1.42	1.40	1.42	1.36	1.40	1.19	1.27
Usps	1.01	1.08	1.00	1.07	1.00	1.07	1.00	1.08	0.82	0.88
Satimage	1.16	1.24	1.12	1.24	1.11	1.26	1.11	1.23	0.90	0.98
Pendigits	1.59	1.60	1.58	1.60	1.57	1.67	1.57	1.55	1.03	1.15
Gisette	1.26	1.28	1.31	1.34	1.28	1.31	1.30	1.30	1.04	1.08
Mnist	1.20	1.22	1.20	1.23	1.21	1.25	1.23	1.23	1.01	1.04
Homus	1.09	1.13	1.15	1.14	1.14	1.07	1.13	1.12	1.07	1.15
Nist	1.19	1.12	1.20	0.99	1.18	1.09	1.20	1.16	1.12	1.12
Connect4	1.18	1.30	1.17	1.28	1.27	1.30	1.29	1.28	1.17	1.22
Covtype	1.28	1.39	1.32	1.33	1.26	1.35	1.30	1.31	1.04	1.14
Poker	1.06	1.11	1.12	1.20	1.09	1.17	1.22	1.25	0.98	1.08
SUSY	0.97	1.01	0.97	0.98	0.96	1.02	1.02	1.01	0.91	0.93
HIGGS	0.95	1.02	1.04	1.16	1.01	1.09	1.01	1.10	1.01	1.00
Average	1.28	1.33	1.34	1.55	1.34	1.48	1.36	1.52	1.08	1.27

The speed up results shown in Table 7 are calculated from both Table 5 (the speed of training phase), and Table 6 (the speed of testing phase) using Equation 7. Here, we can see that the speed of DT4 in training phases is almost similar to that of the FPBST, this is due to the similar trees created by both methods, however the speed of the DT4 in the testing phases is significant 1.27 times of the FPBST testing phases on average, this is due to the disuse of the KNN by DT4. It is interesting to note the high speed of the proposed DT0-DT3 methods, which is about one and half times faster than the FPBST, which might be due to the smaller size of the resultant trees and the disuse of the KNN. It is also interesting to note that the training speeds of the proposed DT0-DT3 are not significant as their testing speeds; this is due to the extra time which is needed for the extra computations of the probabilities of the classes in each node.

As mentioned above, the proposed DT0-DT4 have been further improved attempting to provide more regular trees in terms of class distribution, this improvement includes the enforcement of the examples that are similar to the furthest point of the lower class to be sorted to the left-side of the tree, and those which are similar to the other furthest point with the higher class to be sorted to the right-side of the tree. We conducted several experiments to measure the effect of this improvement on both accuracy and speed. Here we choose the DT of the best performance on each dataset and compare its performance to the FPBST, the comparison results are shown in Table 8.

Table 8. Accuracy (Acc.) and Speed-up results (training and testing phases) of FPBST compared to the proposed DT0+-DT4+.

	FPBST			DT+				DT+ Speed	
Dataset	Acc.	Train	Test	DT	Acc.	Train	Test	Train	Test
Abalone	0.4990	196	14	0+	**0.5441**	206	12	0.95	1.13
Australian	0.6435	45	3	4+	**0.6957**	48	3	0.94	1.03
Balance	0.8258	15	2	1+	**0.8468**	10	1.2	1.50	1.25
blood	0.6784	19	2	1+	**0.7635**	19	1.5	0.98	1.27
Cancer	**0.9618**	29	3	4+	0.9574	32	2.9	0.92	1.07
Climate	0.8722	29	2	4+	**0.9148**	32	2.1	0.91	1.00
German	0.6550	56	4	0+	**0.7100**	52	3.4	1.08	1.03
LetRec	**0.7897**	1595	85	3+	0.7884	1268	68.6	1.26	1.24
Usps	0.8631	8096	390	2+	**0.8694**	6456	306.3	1.25	1.27
Satimage	**0.8672**	860	48	3+	0.8652	670	34.8	1.28	1.38
Pendigits	0.9630	910	54	3+	**0.9642**	583	31.7	1.56	1.70
Gisette	**0.8907**	76157	3752	3+	0.8904	57101	2762.8	1.33	1.36
Mnist	**0.8527**	142712	6869	2+	0.8523	118015	5493.6	1.21	1.25
Homus	**0.4508**	83337	4023	3+	0.4468	80330	3554	1.04	1.13
Nist	0.4795	118746	5686	3+	**0.4828**	108402	4775	1.10	1.19
Connect4	0.6222	6860	395	1+	**0.6723**	5735	298	1.20	1.33
Covtype	**0.9314**	94265	4935	3+	0.9312	74155	3728	1.27	1.32
Poker	0.5372	74536	4243	1+	**0.5889**	66029	3570	1.13	1.19
SUSY	0.7098	923749	46906	1+	**0.7639**	864713	44285	1.07	1.06
HIGGS	0.5860	2974260	164652	1+	**0.6061**	2854760	141130	1.04	1.17
Average	0.7339	225324	12103		0.7577	211931	10503	1.15	1.22

As can be seen from Table 8, the accuracy of the proposed DT after the improvement has increased by about 2.38%, this is due to the sorting of the examples based on their classes, which is the only change that has been made to the decision trees. However there is no improvement in the speed of both training and testing phases, since swapping the furthest points based on their classes need the same computation of swapping them based on their minimum/maximum norms, so there is no extra calculations needed by the new improvement, and that why the time consumed by both phases is not improved.

5. Conclusions

In this paper, we propose a new approach to improve the performance of the FPBST when classifying small, intermediate and Big Data sets, the major improvement includes the disuse of the slow KNN, which is used with the FPBST to classify a small number of examples found in a leaf-node, instead, we convert the BST to a decision tree by its own seizing the labeled examples in the training phase, by calculating the probability of each class in each node, we used various method to calculate these probabilities. The experimental results show that the proposed decision trees improve the performance of the FPBST in terms of classification accuracy, training speed, testing speed and the size of the model (the tree in our case). We also made another simple enhancement on the FPBST algorithm in the training phase, which is the swapping of the furthest pair of points based on their classes rather than being based on their minimum/maximum Euclidian norms. This makes the resultant decision tree more coherent in terms of the distribution of the classes, making them closer to each others as possible as could, such enhancement further improved the accuracy of the proposed decision trees compared to that of the FPBST as the results suggest.

This approach is still based on finding the furthest-pair of points (diameter), which has two major disadvantages, first, it takes time to find the diameter of the data at each node, and second, these two points might be belonging to the same class, and this might affect the classification accuracy. Therefore, we need to find a more accurate and perhaps faster algorithm to cluster the data in each node. Moreover, the Euclidian distance might not be the perfect choice for the proposed methods, therefore, we need to investigate other distance metrics such as [28] [29]. The aforementioned limitations will be addressed in our future work.

Supplementary Materials: The data sets used in this paper are available online: http://archive.ics.uci.edu/ml and https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets.

Funding: This research received no external funding.

Conflicts of Interest: The author declares no conflict of interest.

References

[1] P. Zerbino, D. Aloini, R. Dulmin and V. Mininno, "Big Data-enabled Customer Relationship Management: A holistic approach," *Information Processing & Management*, vol. 54, no. 5, pp. 818-846, 2018.

[2] S. LaValle, E. Lesser, R. Shockley, M. Hopkins and N. Kruschwitz, "Big data, analytics and the path from insights to value," *MIT sloan management review*, vol. 52, no. 2, p. 21, 2011.

[3] Q. Zhang, L. T. Yang, Z. Chen and P. Li, "A survey on deep learning for big data," *Information Fusion*, vol. 42, pp. 146-157, 2018.

[4] V. Bolón-Canedo, B. Remeseiro, K. Sechidis, D. Martinez-Rego and A. Alonso-Betanzos, "Algorithmic challenges in Big Data analytics," in *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, ESANN, 2017.

[5] X. Lv, "The big data impact and application study on the like ecosystem construction of open internet of things," *Cluster Computing*, pp. 1-10, 2018.

[6] E. Fix and J. Hodges, "Discriminatory Analysis: Nonparametric Discrimination: Consistency Properties," USAF School of Aviation Medicine, Randolph Field, Texas, 1951.

[7] T. M. Cover and P. E. Hart, "Nearest Neighbor Pattern Classification," *IEEE Trans. Inform. Theory*, Vols. IT-13, pp. 21-27, 1967.

[8] A. B. Hassanat, "Norm-Based Binary Search Trees for Speeding Up KNN Big Data Classification," *Computers*, vol. 7, no. 4, p. 54, 2018.

[9] A. B. Hassanat, "Furthest-Pair-Based Binary Search Tree for Speeding Big Data Classification Using K-Nearest Neighbors," *Big Data*, vol. 6, no. 3, p. 225–235, 2018.

[10] A. B. Hassanat, "Two-Point-Based Binary Search Trees for Accelerating Big Data Classification using

KNN," *PLOS One,* vol. 13, no. 4, p. 14, 2018.

[11] A. B. Hassanat and A. S. Tarawneh, "Fusion of Color and Statistic Features for Enhancing Content-Based Image Retrieval Systems," *Journal of Theoretical & Applied Information Technology,* vol. 88, no. 3, pp. 644-655, 2016.

[12] A. S. Tarawneh, D. Chetverikov, C. Verma and A. B. Hassanat, "Stability and Reduction of Statistical Features for Image Classification and Retrieval: Preliminary Results," in *ICICS2018*, Irbid, 2018.

[13] A. B. Hassanat, "Greedy Algorithms for Approximating the Diameter of Machine Learning Datasets in Multidimensional Euclidean Space: Experimental Results," *Advances in Distributed Computing and Artificial Intelligence Journal ,* vol. 7, no. 3, p. 12, 2018.

[14] S. Zhang, X. Li, M. Zong, X. Zhu and R. Wang, "Efficient knn classification with different numbers of nearest neighbors," *IEEE transactions on neural networks and learning systems,* vol. pp, no. 99, pp. 1-12, 2017.

[15] A. Hassanat, M. Abbadi, G. Altarawneh and A. Alhasanat, "Solving the Problem of the K Parameter in the KNN Classifier Using an Ensemble Learning Approach," *International Journal of Computer Science and Information Security,* vol. 12, no. 8, pp. 33-39, 2014.

[16] F. Wang, Q. Wang, F. Nie, W. Yu and R. Wang, "Efficient tree classifiers for large scale datasets," *Neurocomputing,* vol. 284, no. 1, pp. 70-79, 2018.

[17] J. Maillo, I. Triguero and F. Herrera, "A mapreduce-based k-nearest neighbor approach for big data classification," in *Trustcom/BigDataSE/ISPA*, 2015.

[18] J. Maillo, S. Ramírez, I. Triguero and F. Herrera, "kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for big data," *Knowledge-Based Systems,* vol. 117, no. 1, pp. 3-15, 2017.

[19] Z. Deng, X. Zhu, D. Cheng, M. Zong and S. Zhang, "Efficient kNN classification algorithm for big data," *Neurocomputing,* vol. 195, no. 1, pp. 143-148, 2016.

[20] A. J. Gallego, J. Calvo-Zaragoza, J. J. Valero-Mas and J. R. Rico-Juan, "Clustering-based k-nearest neighbor classification for large-scale data with neural codes representation," *Pattern Recognition,* vol. 74, no. 1, pp. 531-543, 2018.

[21] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM,* vol. 18, no. 9, pp. 509-517, 1975.

[22] J. K. Uhlmann, "Satisfying general proximity / similarity queries with metric trees," *Information Processing Letters,* vol. 40, no. 4, pp. 175-179, 1991.

[23] A. Beygelzimer, S. Kakade and J. Langford, "Cover trees for nearest neighbor," in *23rd international conference on Machine learning*, 2006.

[24] A. M. Kibriya and E. Frank, "An empirical comparison of exact nearest neighbour algorithms," in *European Conference on Principles of Data Mining and Knowledge Discovery*, Berlin, 2007.

[25] A. Cislak and S. Grabowski, "Experimental evaluation of selected tree structures for exact and approximate k-nearest neighbor classification," in *Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2014.

[26] R.-E. Fan, "LIBSVM Data: Classification, Regression, and Multi-label," 2011. [Online]. Available: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/. [Accessed March 2018].

[27] M. Lichman, University of California, Irvine, School of Information and Computer Sciences, 2013. [Online]. Available: http://archive.ics.uci.edu/ml.

[28] A. B. A. Hassanat, "Dimensionality Invariant Similarity Measure," *Journal of American Science,* vol. 10, no. 8, pp. 221-226, 2014.

[29] M. Alkasassbeh, G. Altarawneh and A. B. Hassanat, "On Enhancing The Performance Of Nearest Neighbour Classifiers Using Hassanat Distance Metric," *Canadian Journal of Pure and Applied Sciences,* vol. 9, no. 1, pp. 3291-3298, 2015.