# Furthest-Pair-Based Binary Search Tree for Speeding Big Data Classification Using K-Nearest Neighbors

**Article** *in* Big Data · September 2018

DOI: 10.1089/big.2018.0064

CITATIONS
13

READS
74

**1 author:**

Ahmad Hassanat
University of Tabuk
**76** PUBLICATIONS **590** CITATIONS

SEE PROFILE

Big Data
Volume 6, Number 3, 2018

DOI: 10.1089/big.2018.0064

ORIGINAL ARTICLE

# Furthest-Pair-Based Binary Search Tree for Speeding Big Data Classification Using K-Nearest Neighbors

Ahmad B.A. Hassanat*

**Abstract**

The advances in information technology of both hardware and software have allowed big data to emerge recently, classification of such data is extremely slow, particularly when using K-nearest neighbors (KNN) classifier. In this article, we propose a new approach that creates a binary search tree (BST) to be used later by the KNN to speed up the big data classification. This approach is based on finding the furthest-pair of points (diameter) in a data set, and then, it uses this pair of points to sort the examples of the training data set into a BST. At each node of the BST, the furthest-pair is found and the examples located at that particular node are further sorted based on their distances to these local furthest points. The created BST is then searched for a test example to the leaf; the examples found in that particular leaf are used to classify the test example using the KNN classifier. The experimental results on some well-known machine learning data sets show the efficiency of the proposed method, in terms of speed and accuracy compared with the state-of-the-art methods reviewed. With some optimization, the proposed method has a great potential to be used for big data classification and can be generalized for other applications, particularly when classification speed is the main concern.

**Keywords:** AI; algorithms; BST; d-dimensional data sets; diameter; machine learning



## Introduction

Over the past two decades, the advances in information technology of both hardware and software allow for easy and cheap ways to access, create, and share data by most humans on this earth. Such a massive interaction results in generating gigantic data sets,[1] for example, Flickr creates about 3.6 TB image data, whereas Google deals with about 20,000 TB daily data.[2] Such data are referred to as big data, which can be defined by their major characteristics that include and not limited to large volume, large variety, large velocity, and large veracity.[2,3] For the same reasons, machine learning data sets have become gigantic in volume too; this is good in a way that a big data set provides more information to the classifier for better learning. However, it is a curse at the same time since the classification of such gigantic data using traditional methods becomes time-consuming, and even unacceptably slow, particularly when using a lazy learner classifier such as the K-nearest neighbors (KNN) classifier.[4]

In the machine learning domain, the KNN classifier is the most well known and easy to apply, sometimes it achieves significant classification rates compared with the more complex methods, and it is the only available choice in certain cases, such as when used for content-based image retrieval.[5,6] However, and since it cannot produce a constant-sized training model to be used for faster testing, it is considered a slow and lazy learner. For testing a single example, the KNN uses the whole training data set each time. For example, running the KNN on a high-performance computing machine took more than 6 weeks to classify the Higgs data set with 11 million examples.

*IT Department, Mutah University, Karak, Jordan.*

*Address correspondence to: *Ahmad B.A. Hassanat, IT Department, Mutah University, P.O. Box 1212, Karak 61710, Jordan*, E-mail: hasanat@mutah.edu.jo

Given a training data set of $n$ examples, in $d$ dimensional Euclidean feature space, the time complexity to classify one example using KNN is O($n.d$) time, and in the case of big data, where $n$ and/or $d$ are very large values, the KNN becomes inefficient, particularly when used for online systems.

Throughout the last half-century, starting from the works of Fix and Hodges[7] and Cover and Hart[8] until today, remarkable efforts have been made to improve the speed of the KNN classifier. However, in this article, we will review the most recent researches in this domain, focusing on those that addressed the big data classification using KNN, such as the work of Maillo et al.,[9] where the authors presented a parallel implementation based on mapping the training set examples and reducing the number of examples that are related to a test sample, they called their method map-reduce KNN (MR-KNN). The reported accuracy rates were similar to those of the KNN, but the results obtained faster than the KNN. However, the speed of the MR-KNN depends largely on the number of maps used, which were in the range of 16–256. Moreover, the number of neighbors (K) used, found to be related to the speed of the algorithm, as the mapping step consumes significant time when large values of K are used. This exciting work was later developed by the same authors,[10] where they proposed a new variant of KNN based on Spark (KNN-IS), which is inspired by the MR-KNN but with multiple reducers to further speed up the classification process.

Clustering the training set to reduce the search time is a common approach in this domain. Such an approach includes the work of Deng et al.,[11] who proposed a couple of almost similar methods to speed up the KNN classifier using K-means clustering. The difference between these methods is in the clustering approach, as one of them used landmark spectral clustering KNN (LC-KNN) and the other used random clustering KNN (RC-KNN). When finding the related cluster of a test example, both methods work the same by employing the KNN on a small set of examples. Deng et al.[11] evaluated their methods on nine big data sets; the accuracy results were approximated well to those of the KNN, and both were much faster than the KNN. Nevertheless, the speed of both methods depends mainly on the number of clusters used.

Another recent clustering approach is developed by Gallego et al.,[12] who proposed two clustering methods to enhance the performance of the KNN in terms of time consumed, the first method is called cluster-based KNN (cKNN) and the second is an enhancement of the first method and called cKNN$^+$, both are almost similar, but the enhanced version used a cluster augmentation technique. Similar to Deng et al.,[11] the average classification accuracy on all the big data sets used was depending on the number of clusters used; however, the reported results improved significantly when Deep Neural Networks was used for learning a proper representation for the classification task.

A different approach is presented by Wang et al.,[13] who proposed two multivariate decision tree classifiers to speed up the big data classification using KNN. Both methods partition the training data to build a decision tree to be used efficiently in the test phase, the first method is based on random partitioning and called MDT1 and the second is based on principal component analysis (PCA) partitioning and called MDT2. Since both methods used a tree structure with a class decision in the leaf node, there was no need for the KNN algorithm to be used at all, and this increased the speed of both methods significantly, without greatly affecting the classification accuracy rates, which were competitive to those of the KNN. Other related problems include big data clustering,[14] big data series indexing,[15,16] and finding the best k for the KNN.[17,18] All these are out of the scope of this article, since this article focuses on big data classification of numeric data using the nearest neighbor (1NN) rather than KNN; however, we will use both terms interchangeably.

It is logical to classify the vast majority of the proposed methods for big data classification as divide and conquer algorithms, particularly those that depend on clustering, splitting, or partitioning the data to reduce its gigantic size to a manageable size. However, in most of these approaches, the user needs to determine the best number of clusters/partitions, as the performances of most of these methods (in terms of accuracy and speed) depend mainly on this number, and tuning such a number is a problem in its own.

Despite having high-quality methods for big data classification in the literature, there is still room for improving the classification's performance in regard to accuracy and speed of both training and testing phases.

In this article, we propose a nonparametric approach based on sorting the numeric training examples into a

binary search tree (BST), to be used later by the KNN classifier, attempting to hasten the big data classification, which is the main goal of this article. Since searching a BST for an example should be faster than searching the whole training data, and faster than searching between and within clusters, particularly when each cluster hosts a large number of examples.

The proposed method that creates the BST depends mainly on finding two local examples (points), these points are the furthest-pair (diameter) of a set of points in d-dimensional Euclidean feature space and these points are then used to sort the other examples based on their similarity. The assumption behind this approach is that considering the furthest points, means considering the most dissimilar points, which are more likely to be belonging to different classes, and therefore sort the examples of a specific class together or nearby. However, this is not necessarily true all the time, but even though, by using this approach, we get similar examples sorted nearby in the BST regardless of their classes, and this is enough for nearest neighbor search.

The use of efficient data structures for speeding up the lazy learner classifiers is not a fresh idea. There exist extensive literature on tree structures, such as k-d trees,[19] metric trees,[20] cover trees,[21] and other related works of Kibriya and Frank,[22] and Cislak and Grabowski.[23] Our work is similar to the mentioned tree-based approaches in exploiting a tree structure to increase the search speed. However, its interpretability, simplicity, ability to process big data, and ease of implementation altogether separate our work from such tree-based approaches. In addition, the proposed work is different in how such a tree is constructed, which is based on finding the diameter of the data set in d-dimensional Euclidian feature space.

Finding the exact diameter in d-dimensional Euclidian feature space is a time-consuming process, as it takes quadratic time.[24,25] Therefore, we opt for an approximate solution using a hill climbing technique,[26] which takes O(n.d) time. The words point and example will be used interchangeably in this article to refer to the feature vector (FV) in d-dimensional Euclidean feature space obtained from testing or training machine learning data sets.

The rest of this article presents the proposed method and describes the machine learning data set used for evaluation, in addition, it evaluates and compares our proposed method with other state-of-the-art big data classification methods, namely cKNN, cKNN$^+$, MDT1, MDT2, KNN-IS, MR-KNN, RC-KNN, and LC-KNN. It is noteworthy that we did not compare the proposed method with the other mentioned tree-based methods, as most of them were not designed for handling big data. For example, when k-d trees are used to classify high-dimensional data (as in big data), most of the points will be evaluated and the tree's efficiency will not be better than exhaustive search.[27]

## Furthest-Pair-Based Binary Search Tree

### Methods

Using the standard KNN algorithm as described by Fix and Hodges[7] and Cover and Hart[8] is time-consuming, particularly when classifying big data sets. In this article, we propose the use of a BST to sort the examples (points) of machine learning data sets in a way that facilitates the search process. Such a BST sorts all examples in a training set based on their distances from two local points (P1 and P2), these are two examples from the training data set, and they are different based on the host node and the level/location of that node in a BST.

Here, we opt for finding the furthest points P1 and P2 to create the BST. Since the furthest points are the most dissimilar points, they are more likely to be belonging to different classes, and therefore, sorting other examples based on their distances to these points might be a good choice, as similar examples are sorted nearby, whereas dissimilar examples are sorted faraway in the created BST. However, this is not necessarily always true. In certain cases, the furthest points are belonging to the same class; for example, if the distribution of the classes follows something like having one class's points inside or contained by another class's points. Even though, this should not make a big difference, as the BST will not classify examples by its own, it only sorts these examples based on their similarity to specific points, and the distribution of classes over the BST might not affect the final classification results, as we always have at least one example per class there.

The training phase of the proposed method (furthest-pair-based binary search tree or FPBST) creates a BST, which speeds up searching for a test example compared with the unacceptable time consumed by the KNN classifier, particularly when classifying big data sets. Those examples, which are similar to

P1, are sorted to the left of their host node in a higher level, and the others, which are more similar to P2 than P1, are sorted to the right of the same host node in the same higher level. That is, the examples of a tree node are split into two sets based on their similarity to P1 and P2; these two sets form two child nodes, left and right nodes, respectively. The training phase of the method builds the BST same way recursively, by calculating the new local furthest points and split examples based on them. Typically, we expect to see only one example stored within each leaf node; however, in certain cases, we have more than one, particularly when we have duplicate examples, and/or when two or more examples are of the same distance to both of the local furthest points. FPBST employs Euclidian distance metric (ED) for measuring distance, seeing that most of the state-of-the-art methods in this domain use ED. Algorithm 1 depicts the pseudo-code for the training phase of the FPBST method.

The hill climbing algorithm, which we employed to find the approximate furthest points, works as follows: a random point is chosen from a set of points (examples), then the rest points are compared with these points using Euclidean distance, the point with the maximum distance is taken to recursively compare it with the others. The algorithm repeats itself until no enhancement in the maximum distance is found and then output the last 2 points with the maximum distance between them, as described by Hassanat.[26] To make the resultant BST coherent, we enforced P1 to be the point with the minimum Euclidean norm (EN) and P2 to be the point with the maximum one, as to keep similar examples nearby as possible.

The time complexity of training phase to build the BST is O($cdn$ log $n$), where ($cdn$) is the time consumed to find the approximate furthest points since the constant $c$ is the number of iterations needed to find the approximate furthest points and found experimentally to be in the range of 2–5. The (log $n$) time is consumed along the depth of the BST, an extra ($2nd$) time is consumed by comparing each FV (or example) by the local furthest points. This time can be added to $c$ to make it in the range of 4–7; however, $c$ is still a constant and the overall time complexity can be asymptotically approximated to O($dn$ log $n$), and if $n >> d$, the time complexity can be further approximated to $O(n$ log $n)$.

**Algorithm 1. Training Phase (BST Building) of FPBST.**

```
Input: Numerical training data set DATA with n FVs and d features.
Output: A Root pointer to the FPBST.
Create a BST Node → RootN
RootN.Examples←FVs //all indexes of FVs from the training set
(P1, P2) ← Procedure Furthest(DATA ← RootN.Examples, n) //hill
  climbing algorithm
  P1 ← random point (1, n)
  MAX ← 0
  for each point piinDATA, do
    D ← ED (P1, pi) //Euclidian distance
    if D > MAX then
      MAX ← D
      P2 ← pi
    end
  end
  Stop ← false
  while not Stop, do
    MAX2 ← 0
    for each point pi in DATA, do
      D=ED (P2, pi)
      if D > MAX2 then
          MAX2 ← D
          P3 ← pi
      end
    end
    if MAX2 > MAX then
          P1 ← P2
          P2 ← P3
          MAX ← MAX2
    else Stop ← true
    end
  end
  return(P1, P2)
end Procedure Furthest
if EN(P1) > EN(P2) swap(P1, P2)
RootN.P1 ← P1
RootN.P2 ← P2
RootN.Left=Null
RootN.Right=Null
Procedure BuildBST(Node ← RootN)
  for each FVi in Node, do
    D1 ← ED(FV[i], Node.P1)
    D2 ← ED(FV[i], Node.P2)
    If (D1 < D2)
      Add index of FVi to Node.Left.Examples
    else
      Add index of FVi to Node.Right.Examples
  end
  if (Node.Left==Null orNode.Right==Null)
    return //this means a leaf node
  end
  (P1, P2) ← Furthest(Node.Left.Examples, size(Node.Left.Examples))
  if EN(P1) > EN(P2) swap(P1, P2)
  Node.Left.P1 ← P1
  Node.Left.P2 ← P2
  BuildBST(Node.Left) //recursive call
  (P1, P2) ← Furthest(Node.Right.Examples, size(Node.Right.Examples))
  if EN(P1) > EN(P2) swap(P1, P2)
  Node.Right.P1 ← P1
  Node.Right.P2 ← P2
  BuildBST(Node.Right) //recursive call
end Procedure
return RootN
```

**Table 1. A hypothetical training data sample to exemplify the resultant binary search tree of the furthest-pair-based binary search tree**

| *No. of example* | *F1* | *F2* | *F3* | *F4* | *Class* | *Norm* |
|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 2 |
| 1 | 1 | 1 | 3 | 3 | 2 | 4.472136 |
| 2 | 1 | 1 | 3 | 4 | 2 | 5.196152 |
| 3 | 5 | 5 | 5 | 5 | 1 | 10 |
| 4 | 1 | 1 | 3 | 5 | 2 | 6 |
| 5 | 1 | 1 | 4 | 1 | 2 | 4.358899 |
| 6 | 1 | 1 | 5 | 1 | 2 | 5.291503 |
| 7 | 1 | 1 | 5 | 3 | 2 | 6 |
| 8 | 1 | 1 | 5 | 4 | 2 | 6.557439 |
| 9 | 5 | 5 | 5 | 1 | 3 | 8.717798 |
| 10 | 5 | 5 | 5 | 2 | 3 | 8.888194 |
| 11 | 1 | 1 | 4 | 2 | 2 | 4.690416 |
| 12 | 5 | 5 | 5 | 3 | 3 | 9.165151 |
| 13 | 5 | 5 | 5 | 4 | 3 | 9.539392 |
| 14 | 1 | 1 | 5 | 2 | 2 | 5.567764 |

If we apply FPBST on the data sample shown in Table 1, which is taken from the Balance data set, as a hypothetical training set, we get the BST shown in Figure 1.

As shown in Figure 1, examples (0 and 3) are the furthest points in the data set, and therefore, the 15 examples will be sorted based on their distances from these examples. We also note that the (0) example is always sorted to the left as it has the minimum norm, same note for example (3), which has the maximum norm and always goes to the right of the BST. It is interesting to note how the sorting works, a closer look at examples 9 and 10 in Table 1 shows how these examples are similar, this similarity is reflected by the resultant BST, as both of examples are located nearby and share the same parent node, same applies to examples (2 and 4) and the rest of examples. However, examples (0 and 3) are sorted far away from each other despite belonging to the same class (1), this is normal as we are sorting the examples based on their features and not based on their classes. It is also interesting to note the small depth of the tree, which is in the range of 4–7, and 5.33 on average, which is close to $\log_2$ (15), this small depth shall speed up the searching process and leave the KNN with a very small number of examples to compare.

The test phase of the proposed method searches the created BST for a test example starting from the root node to a leaf node, where similar examples are supposed to be there, and then, the normal KNN algorithm is employed to classify the test example based on those examples found in a leaf node. There is no need to use KNN or 1NN if there is one example in the leaf found; however, in our experiments, we used KNN for all cases systematically, as sometimes we have more than one example there. The test phase of the FPBST costs $O(2d.\log n)$ time for each example tested. The $(2d)$ time is consumed by the calculation of the ED, which

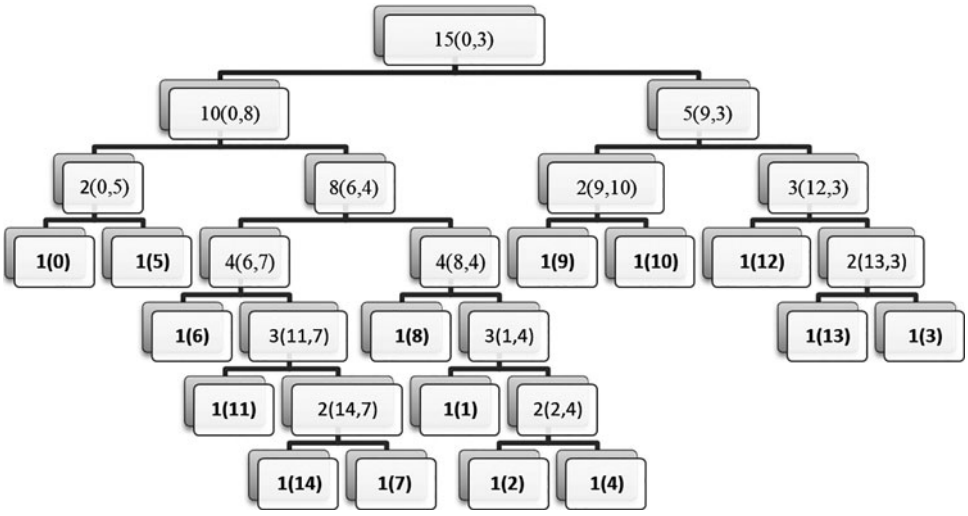


**FIG. 1.** The resultant BST after applying the FPBST in the training phase on the sample training data from Table 1. The leaf nodes are bolded. The number outside the brackets is the counter of the examples hosted by each node, and those inside the brackets are the index of the examples in a leaf node or the furthest points (P1 and P2) otherwise. BST, binary search tree; FPBST, furthest-pair-based binary search tree.

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; however, if $n >> d$, the $d$ time can be ignored leaving the testing time with O(log $n$), which is very fast. The time consumed by the KNN algorithm can be neglected too, as it works on one or very small number of examples. Algorithm 2 depicts the pseudo-code for the testing phase of the proposed FPBST.

**Algorithm 2. Testing Phase of FPBST.**

```
Input: test data set TESTDT with n FVs and d features.
Output: Testing Accuracy Acc.
Acc ←0
for each FV_i in TESTDT, 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
   Procedure KNN(FV_i, Node)
      Array Distance;
      for each fv_j in Node, do
         Distance[j] ← ED[fv_j, FV_i]
      end
      index ← argmin(Distance[j])
      class ← Class(fv_index)
         return class
      end Procedure KNN
      if class==Class(FV_i)
         Acc ← Acc +1
      end
end
Acc ← Acc/n
return Acc
```

## Data

Similar to the previous work on big data classification, we evaluated and compared the proposed FPBST utilizing some of the well-known machine learning data sets, which are used by state-of-the-art work in this domain, these data sets were downloaded from either the LIBSVM Data[28] or the Machine Learning Repository.[29] We used the data sets of different sizes, dimensions, and data types to evaluate the efficiency of the proposed method in terms of accuracy and time consumed. All data sets used contain numerical data, real numbers and/or integers. The sizes of these data sets are in the range of 625–11,000,000 examples; the dimensions are in the range of 4–5000 features. The descriptions of these data sets are shown in Table 2.

**Table 2. Description of data sets used for evaluation and comparison of the furthest-pair-based binary search tree**

| *Data set* | *Size* | *Dimensions* | *Type* |
|---|---|---|---|
| Abalone | 4177 | 8 | Real |
| Australian | 690 | 14 | Real |
| Balance | 625 | 4 | Integers |
| Blood | 748 | 4 | Integers |
| Cancer | 683 | 9 | Integers |
| Climate | 1178 | 18 | Real |
| Connect4 | 67,557 | 42 | Integers |
| Covtype | 581,012 | 54 | Integers |
| German | 1000 | 24 | Integers |
| Gisette | 7000 | 5000 | Integers |
| HIGGS | 11,000,000 | 28 | Real |
| Homus | 15,199 | 1600 | Integers |
| Letter | 20,000 | 16 | Real |
| Mnist | 70,000 | 784 | Integers |
| Nist | 44,951 | 1024 | Integers |
| Pendigits | 10,992 | 16 | Integers |
| Poker | 1,025,010 | 11 | Integers |
| Satimage | 6435 | 36 | Real |
| SUSY | 5,000,000 | 18 | Real |
| Usps | 9298 | 256 | Real |

## Results and Discussions

To evaluate the proposed FPBST method, we programmed both Algorithms 1 and 2 using MS VC++.Net framework and conducted several classification experiments on all the data sets described in the Data section. We utilized Azure's cloud high-performance computing platform (https://portal.azure.com), the specification of the virtual machine used includes 16 Intel® CPUs at 2.3 GHz and 32 GB RAM. Being a single-threaded algorithm, the proposed FPBST in both phases had not benefited from the multi-CPUs available in the virtual machine used; however, this high-performance computing environment allowed for multi-experiments to run simultaneously, and this shortened the time needed to conduct the experiments significantly, particularly when experimenting on big data sets.

Table 3 shows the characteristics of the BST built by the proposed FPBST when training.

**Table 3. The resultant binary search trees after applying the proposed furthest-pair-based binary search tree in the training phase, time in (ms)**

| *Characteristic* | *Covtype* | *HIGGS* | *Gisette* | *Usps* |
|---|---|---|---|---|
| Building time | 75,045 | 2,870,330 | 74,205 | 8782 |
| No. of examples | 464,810 | 8,800,000 | 6000 | 7291 |
| No. of leaves | 464,810 | 8,621,062 | 6000 | 7291 |
| Maximum no. of examples in a leaf | 1 | 3 | 1 | 1 |
| Average no. of examples in a leaf | 1 | 1.02 | 1 | 1 |
| Maximum depth | 32 | 54 | 27 | 33 |
| Log no. of examples | 18.826 | 23.07 | 12.55 | 12.83 |

It is interesting to note (from Table 3) the maximum depth of the resultant BSTs, which is not much larger than log $n$, and the average depth is very close to log $n$ for each data set, which shall speed up searching the BST. Another interesting point is the number of examples per leaf node, which is around 1 in all the data sets used, and this will speed up the KNN classifier significantly after the search ends. A closer look at the BST's building time reveals surprising results, as we see almost similar time consumed to build the BST for both of the Gisette and Covtype data sets, despite the very large difference in the size of both. This is due to the large number of features of the Gisette data set, which is 5000 compared with that of the Covtype, which is only 54. In such a situation, $n$ is not $>>d$, therefore $d$ has a big effect on the time complexity of building the BST since the proposed method uses the ED twice for each example in a node during the building process of the BST, as the distance metric uses all the dimensions. The resultant BST has nothing to do with $d$, as its main goal is to reduce the search time in terms of $n$, the number of examples, so if we have large values of $d$, not only the BST building time will be high but also the testing time too, solving this problem is beyond the scope of this article.

In this article, we compared the performance of the proposed method with the most recent methods proposed for the big data classification, which include cKNN, cKNN$^+$, MDT1, MDT2, KNN-IS, MR-KNN, RC-KNN, and LC-KNN. A fast review of these state-of-the-art methods shows that each one has its own evaluation methodology since the reviewed works show different data sets classified under different evaluation schemes. Some researchers used 10-fold cross-validation (CV), others used 5-fold, and others used the hold-out set, with different test/train ratios! Therefore, and to be able to compare with these state-of-the-art methods, we had to change our experiments settings as per each compared work.

There is also another differentiating factor, which is the use of different hardware with different computation powers, which significantly affects the comparison time consumed. To work around this problem and to make a valid comparison in terms of time consumed by each method, we opt for reporting the speed-up factor of each algorithm similar to the work of Maillo et al.[9] We calculated the speed-up factor by considering the ratio of the time consumed by the KNN classifier to that of an algorithm on the same data set and same examples as follows

$$Speedup\ (X, D) = \frac{Time(KNN, D)}{Time(X, D)}, \quad (1)$$

where $D$ is the data set tested, $X$ is the algorithm, which needs its speed-up factor to be calculated, and the time consumed is returned by the Time function.

Poker and SUSY data sets were used by both of the MR-KNN[9] and KNN-IS[10] for evaluation using the five-fold CV. Therefore, we used the same data sets and the same CV to compare the proposed FPBST with these methods. The accuracy results (Table 4) and speed-ups (Table 5) of FPBST are compared with those of the MR-KNN and KNN-IS.

It is noteworthy that the accuracy results of MR-KNN and KNN-IS (Table 4) are the same as the KNNs because both of them are parallel-exact algorithms.

As shown in Table 4, the accuracy of the proposed FPBST is higher than that of MR-KNN, KNN-IS, and the KNN. Same applies to the interesting speed of the proposed FPBST stated in Table 4, where its speed is more than twice the speed of the KNN-IS when tested on Poker, and more than five times the speed of the KNN-IS when tested on SUSY, as well as significantly higher than the speed of MR-KNN considering both

**Table 4. Accuracy results of furthest-pair-based binary search tree compared with map-reduce K-nearest neighbors and K-nearest neighbors based on Spark using fivefold cross-validation**

| *Method* | *Data set* | *Accuracy* | *Average run time (ms)* |
|---|---|---|---|
| KNN-IS | Poker | 0.502 | 102,938 |
| | SUSY | 0.694 | 1,900,039 |
| MR-KNN | Poker | 0.502 | 804,456 |
| | SUSY | 0.694 | 12,367,966 |
| KNN | Poker | 0.502 | 105,475,006 |
| | SUSY | 0.694 | 3,258,848,811 |
| FPBST | Poker | **0.536** | **35,516** |
| | SUSY | **0.709** | **255,481** |
| Our KNN | Poker | 0.594 | 76,414,385 |
| | SUSY | ? | 2,360,966,234* |

Bold values signify the best performance, either highest accuracy or shortest consumed time.

*Our KNN algorithm took >6 weeks and did not end so we stopped it; however, this time consumed when testing SUSY is approximated based on the ratio between the reported time consumed by KNN when testing SUSY and Poker data sets, the reported KNN took 30.9 times the time consumed when testing Poker, so by multiplying this ratio by the time consumed by our KNN when testing poker we get this approximate time. The accuracy of our KNN is still unknown as the algorithm took >6 weeks and stopped.

FPBST, furthest-pair-based binary search tree; KNN, K-nearest neighbors; KNN-IS, KNN based on Spark; MR-KNN, map-reduce KNN.

**Table 5. Speed-up results of furthest-pair-based binary search tree compared with map-reduce K-nearest neighbors and K-nearest neighbors based on Spark using fivefold cross-validation**

| *Method* | *Poker* | *Susy* |
|---|---|---|
| MR-KNN | 131 | 263 |
| KNN-IS | 1025 | 1715 |
| FPBST | **2152** | **9241** |

Bold values signify the best performance, which is the highest speed-up factor.

data sets. We justify this extremely high speed with the use of the BST, which has a small (log $n$) average depth.

RC-KNN and LC-KNN[11] were evaluated using 10-fold CV when tested a number of big data sets; so to compare the proposed FPBST with these methods, we used the same CV to test the same data sets. The accuracy results of the proposed method are compared with those of the RC-KNN and LC-KNN in Table 6 and the speed-ups comparisons are also stated in Table 7.

It can be noted from Table 6 that the accuracy results achieved by the proposed FPBST are close to those of the LC-KNN and higher than those of the RC-KNN on average. However, the relatively small decline (5%) of the average accuracy compared with the LC-KNN can be compensated by the extremely high speed of the proposed FPBST, which found to be much higher than both of the RC-KNN and LC-KNN, as shown in Table 7. Once more, we regard such extremely high speed to the efficiency of the BSTs that created in the training phase and used in the testing phase.

cKNN and cKNN$^+$ [12] were evaluated using fivefold CV when tested a number of big data sets. Therefore,

**Table 6. Accuracy results of furthest-pair-based binary search tree compared with random clustering K-nearest neighbors and landmark spectral clustering K-nearest neighbors using 10-fold cross-validation**

| *Data set* | *RC-KNN* | *LC-KNN* | *KNN* | *FPBST* | *Our KNN* |
|---|---|---|---|---|---|
| Usps | 0.903 | **0.936** | 0.95 | 0.873 | 0.97 |
| Mnist | 0.722 | 0.839 | 0.86 | **0.855** | 0.97 |
| Gisette | 0.931 | **0.953** | 0.97 | 0.895 | 0.96 |
| Letter | 0.789 | **0.950** | 0.95 | 0.789 | 0.96 |
| Pendigits | 0.945 | **0.972** | 0.98 | 0.965 | 0.99 |
| Satimage | 0.860 | **0.888** | 0.91 | 0.864 | 0.90 |
| Average | 0.858 | **0.923** | 0.936 | 0.873 | 0.961 |

Bold values signify the best performance, which is the highest accuracy for each data set.

LC-KNN, landmark spectral clustering KNN; RC-KNN, random clustering KNN.

**Table 7. Speed-up results of furthest-pair-based binary search tree compared with random clustering K-nearest neighbors and landmark spectral clustering K-nearest neighbors using 10-fold cross-validation**

| *Data set* | *RC-KNN* | *LC-KNN* | *FPBST* |
|---|---|---|---|
| Usps | 9.2 | 8.7 | **190** |
| Mnist | 8.2 | 6.8 | **1158** |
| Gisette | 9.3 | 7.6 | **162** |
| Letter | 6.1 | 6.0 | **95** |
| Pendigits | 3.1 | 3.0 | **35** |
| Satimage | 2.8 | 2.7 | **40** |
| Average | 6.5 | 5.8 | **280.0** |

Bold values signify the best performance, which is the highest speed-up factor.

and to further evaluate and compare our proposed method, we tested the proposed FPBST on the same data sets with the same fivefold CV. Table 8 shows the accuracy results of the proposed FPBST without comparing with those of the cKNN and cKNN because the reported results were averaged over all the data sets used and not reported for each data set. Moreover, there was no reported consumed time to compare the speed as well. Nevertheless, we compared our results with the reported average accuracies, which found to be varying in the range of 83%–90% for both of the cKNN and cKNN$^+$ depending on the number of clusters and neighbors used.

As shown in Table 8, the FPBST achieved lower accuracy results compared with those of the KNN. This is due to the approximate nature of the proposed method, as it mainly relies on the small number of examples found in the leaf node, which is normally one example (except for some certain cases), without having the opportunity to compare with other examples. This actually speeds up the testing phase, but it was on the

**Table 8. Accuracy and speed results of furthest-pair-based binary search tree compared with K-nearest neighbors using fivefold cross-validation**

| | *Accuracy* | | *Speed-up* |
|---|---|---|---|
| *Database* | *FPBST* | *KNN* | *FPBST* |
| Gisette | 0.885 | 0.960 | 145 |
| Letter | 0.785 | 0.957 | 78 |
| Homus | 0.451 | 0.647 | 223 |
| Satimage | 0.860 | 0.904 | 32 |
| Pendigits | 0.964 | 0.994 | 27 |
| Mnist | 0.851 | 0.972 | 1100 |
| Usps | 0.862 | 0.973 | 151 |
| Nist | 0.478 | 0.797 | 610 |
| Average | 0.767 | 0.901 | 296 |

**Table 9. Accuracy results of furthest-pair-based binary search tree compared with MDT1and MDT2 with different test/train ratios**

| *Data set* | *Train ratio, %* | *Test ratio, %* | *MDT1* | *MDT2* | *KNN* | *FPBST* | *Our KNN* |
|---|---|---|---|---|---|---|---|
| Poker | 2 | 98 | 0.507 | **0.543** | 0.511 | 0.492 | 0.511 |
| SUSY | 80 | 20 | 0.729 | **0.749** | — | 0.710 | — |
| Covtype | 80 | 20 | 0.858 | 0.901 | — | **0.928** | 0.965 |
| Letter | 74 | 26 | 0.616 | **0.805** | 0.957 | 0.768 | 0.954 |
| Pendigits | 68 | 32 | 0.851 | 0.937 | 0.977 | **0.969** | 0.994 |
| Satimage | 69 | 31 | 0.788 | 0.856 | 0.895 | **0.869** | 0.894 |
| Connect-4 | 80 | 20 | 0.660 | **0.676** | 0.657 | 0.632 | 0.611 |
| Usps | 78 | 22 | 0.661 | 0.859 | 0.951 | **0.862** | 0.977 |
| Gisette | 86 | 14 | 0.617 | **0.901** | 0.958 | 0.890 | 0.951 |
| HIGGS | 80 | 20 | 0.581 | **0.600** | — | 0.582 | — |
| Average | 69.7 | 30.3 | 0.687 | 0.783 | 0.844 | 0.770 | 0.857 |

Bold values signify the best performance, which is the accuracy.



account of accuracy. Such a kind of deficiency can be easily solved by going to upper levels in the BST, where more examples can be found and shall enhance the accuracy results; however, such enhancement is beyond the scope of this article. Compared with the cKNN and cKNN$^+$, we find the proposed FPBST outperforming the maximum average accuracy of both cKNN and cKNN$^+$ when testing on the Pendigits data set, and the minimum average accuracy when testing on four of the data sets.

The last comparison is made with both of MDT1 and MDT2[13]; these two methods were evaluated on a relatively larger number (24) of data sets. Unlike the aforementioned works, Wang et al.[13] used hold-out set with different test/train ratios. Being in the range of 14%–98%, these ratios were different depending on each data set used. In addition, they preprocessed data sets used by removing redundant FVs and reducing the data sets dimensionalities utilizing PCA. Unfortunately, we could not get these preprocessed data sets to get comparisons that are more reliable; therefore, we conducted our experiments on 10 of the originally available data sets. The rest of the data sets used by Wang et al.[13] were either not available for download or having non-numeric data. To increase the validity of the comparisons, we used the same test/train ratios used for each data set as shown in Table 9.

As shown in Table 9, the proposed FPBST outperforms the MDT1 on seven data sets, showing a significant difference in accuracy on some of these data sets and on average as well. Compared with the MDT2, the proposed FPBST slightly outperforms this method on four data sets. However, the results are very close on average, recalling that both MDT1 and MDT2 run on lower dimensionality and preprocessed data sets, as such a preprocess improves the accuracy rates and boosts the speed of the methods, as shown in Table 10, where both MDT1 and MDT2 found faster than our FPBST on three data sets.

Another interesting remark about the relatively high speeds of the MDT1 and MDT2 is their exploiting of decision trees when the KNN algorithm is not used at all, and each leaf node already contains the decision about the class of the tested example, in addition to the impact of the small depth of such trees. The latter characteristic is also enjoyed by our method, which found faster than both methods when tested on five data sets as shown in Table 10.

In summary, based on the previous comparisons and discussions, we find that the proposed FPBST extremely hasten the KNN algorithm when it is used for big data classification, decreasing the classification speed from weeks to seconds. We also find that FPBST is significantly faster than most of the related state-of-the-art methods when tested on most data sets used. However, there is still room for improvement, particularly in terms of accuracy, since the major limitation of the proposed FPBST is

**Table 10. Speed-up results of furthest-pair-based binary search tree compared with MDT1 and MDT2 using different test/train ratios**

| *Data set* | *MDT1* | *MDT2* | *FPBST* |
|---|---|---|---|
| Poker | 835 | **927** | 78 |
| SUSY | — | — | |
| Covtype | — | — | **2604** |
| Letter | 14 | 75 | **79** |
| Pendigits | 27 | 3 | **40** |
| Satimage | 13 | 7 | **44** |
| Connect-4 | **10,745** | 236 | 101 |
| Usps | 50 | 26 | **70** |
| Gisette | **183** | 16 | 170 |
| HIGGS | — | — | **—** |

Bold values signify the best performance, which is the highest speed-up factor.

its relatively low accuracy rates compared with the KNN on most data sets. Since FPBST is a very fast algorithm, its major limitation can be addressed by trading off speed for accuracy.

## Conclusions

In this article, we propose a new approach that creates a BST to be used by the KNN for faster big data classification. This approach is based on finding the furthest-pair (diameter) in a data set, and then, it uses this pair of points to sort the examples of the training data set into a BST. At each node of the BST, the furthest-pair is found and the examples located at that particular node are further sorted based on their distances to these local furthest points. We find that the average number of examples per leaf node is very small and around one, and the average depth of the BST is very small and around the logarithm to the base two of the size of the training data, which extremely speeds up the classification process as proved by the experiments conducted.

The experimental results on a 20 well-known machine learning data sets show the efficiency of the proposed method, in terms of speed and accuracy compared with the state-of-the-art method reviewed. However, compared with the KNN, the accuracy rates are not yet perfect and need further improvements; these include and not limited to the following: (1) removing the need for the KNN by converting the BST to decision-based-BST, (2) taking more than one example by going to upper levels of the BST, (3) finding better furthest-pair algorithms, and (4) investigating other distance metrics, such as the works of Hassanat[30] and Alkasassbeh et al.[31] Moreover, the proposed method is scoped to numeric data sets only; however, we recommend further work to extend the idea to deal with non-numeric data, such as strings, where other similarity measures can be used to calculate the furthest points such as the edit distance. The aforementioned improvements/extensions will be addressed in our future work.

## Author Disclosure Statement

No competing financial interests exist.

## References


1. Lv X. The Big Data impact and application study on the like ecosystem construction of open internet of things. Cluster Comput. [Epub ahead of print]; DOI: https://doi.org/10.1007/s10586-018-2206-z
2. Zhang Q, Yang LT, Chen Z, Li P. A survey on deep learning for Big Data. Inform Fusion. 2018;42:146–157.
3. Bolón-Canedo V, Remeseiro B, Sechidis K, et al. Algorithmic challenges in Big Data analytics. In: European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN, Bruges, Belgium: 16doc.com, 2017, pp. 519–527.
4. Zhu D. Humor robot and humor generation method based on Big Data search through IOT. Cluster Comput. [Epub ahead of print]; DOI: https://doi.org/10.1007/s10586-018-2097-z
5. Hassanat AB, Tarawneh AS. Fusion of color and statistic features for enhancing content-based image retrieval systems. J Theoret Appl Inform Technol. 2016;88:644–655.
6. Tarawneh AS, Chetverikov D, Verma C, Hassanat AB. Stability and reduction of statistical features for image classification and retrieval: Preliminary results. In: 9th International Conference on Information and Communication Systems, ICIS, Irbid, Jordan: Institute of Electrical and Electronics Engineers Inc., 2018, pp. 117–121.
7. Fix E, Hodges J. Discriminatory analysis-nonparametric discrimination: consistency properties. Int Stat Rev. 1951;57:238–247.
8. Cover TM, Hart PE. Nearest neighbor pattern classification. IEEE Trans Inform Theory. 1967;IT-13:21–27.
9. Maillo J, Triguero I, Herrera F. A mapreduce-based k-nearest neighbor approach for Big Data classification. In: Presented at Trustcom/BigDataSE/ISPA, Helsinki, Finland: IEEE, 2015, pp. 167–172.
10. Maillo J, Ramírez S, Triguero I, Herrera F. kNN-IS: An iterative spark-based design of the k-nearest neighbors classifier for Big Data. Knowl Based Syst. 2017;117:3–15.
11. Deng Z, Zhu X, Cheng D, et al. Efficient kNN classification algorithm for Big Data. Neurocomputing. 2016;195:143–148.
12. Gallego AJ, Calvo-Zaragoza J, Valero-Mas JJ, Rico-Juan JR. Clustering-based k-nearest neighbor classification for large-scale data with neural codes representation. Pattern Recognit. 2018;74:531–543.
13. Wang F, Wang Q, Nie F, et al. Efficient tree classifiers for large scale datasets. Neurocomputing. 2018;284:70–79.
14. Liu Z, Zheng Q, Ji Z, Zhao W. Sparse self-represented network map: A fast representative-based clustering method for large dataset and data stream. Eng Appl Artif Intell. 2018;68:121–130.
15. Zoumpatianos K, Idreos S, Palpanas T. Indexing for interactive exploration of Big Data series. In: ACM SIGMOD International Conference on Management of Data, Snowbird, Utah: ACM, 2014, pp. 1555–1566.
16. Palpanas T. The parallel and distributed future of data series mining. In: International Conference on High Performance Computing and Simulation, Genoa, Italy: IEEE, 2017, pp. 916–920.
17. Zhang S, Li X, Zong M, et al. Efficient knn classification with different numbers of nearest neighbors. IEEE Trans Neural Netw Learn Syst. 2018;29:1774–1785.
18. Hassanat AB, Abbadi MA, Altarawneh GA, Alhasanat AA. Solving the problem of the K parameter in the KNN classifier using an ensemble learning approach. Int J Comput Sci Inform Security. 2014;12:33–39.
19. Bentley JL. Multidimensional binary search trees used for associative searching. Commun ACM. 1975;18:509–517.
20. Uhlmann K. Satisfying general proximity/similarity queries with metric trees. Inform Process Lett. 1991;40:175–179.
21. Beygelzimer A, Kakade S, Langford J. Cover trees for nearest neighbor. In: 23rd International Conference on Machine Learning, Pittsburgh, PA: ACM, 2006, pp. 97–104.
22. Kibriya AM, Frank E. An empirical comparison of exact nearest neighbour algorithms. In: European Conference on Principles of Data Mining and Knowledge Discovery, Skopje, Macedonia: Springer, Berlin, Heidelberg, 2007, pp. 140–151.
23. Cislak A, Grabowski S. Experimental evaluation of selected tree structures for exact and approximate k-nearest neighbor classification. In: Federated Conference on Computer Science and Information Systems, Warsaw, Poland: IEEE, 2014, pp. 93–100.
24. Agarwal PK, Matoušek J, Suri S. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. Comput Geom. 1992;1:189–201.
25. Williams R. On the difference between closest, furthest, and orthogonal pairs: Nearly-linear vs barely-subquadratic complexity. In: Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA: SIAM, 2018, pp. 1207–1215.
26. Hassanat AB. Greedy algorithms for approximating the diameter of machine learning datasets in multidimensional euclidean space. arxiv. 1808.03566.
27. Goodman JE. Handbook of discrete and computational geometry, 2nd ed. CRC Press, 2004.

28. Fan R-E. 2011. LIBSVM data: Classification, regression, and multi-label. Available online at www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/.
29. Lichman M. 2013. UCI machine learning repository. Available online at http://archive.ics.uci.edu/ml (last accessed March 20, 2018).
30. Hassanat ABA. Dimensionality invariant similarity measure. J Am Sci. 2014;10:221–226.
31. Alkasassbeh M, Altarawneh G, Hassanat AB. On enhancing the performance of nearest neighbour classifiers using hassanat distance metric. Can J Pure Appl Sci. 2015;9:3291–3298.

**Cite this article as:** Hassanat ABA (2018) Furthest-pair-based binary search tree for speeding big data classification using K-nearest neighbors. *Big Data* 6:3, 225–235, DOI: 10.1089/big.2018.0064.

**Abbreviations Used**

BST = binary search tree
CV = cross-validation
ED = Euclidian distance metric
EN = Euclidean norm
FPBST = furthest-pair-based binary search tree
FV = feature vector
KNN = K-nearest neighbors
KNN-IS = KNN based on Spark
LC-KNN = landmark spectral clustering KNN
MR-KNN = map-reduce KNN
PCA = principal component analysis
RC-KNN = random clustering KNN