International Journal of Engineering

Journal Homepage: www.ije.ir

# The Construction of Scalable Decision Tree based on Fast Splitting and J-Max Pre Pruning on Large Datasets

S. Lotfi[a], M. Ghasemzadeh*[b], M. Mohsenzadeh[a], M. Mirzarezaee[a]

[a] *Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran*
[b] *Computer Department, Engineering Campus, Yazd University*

*PAPER INFO*

*Paper history:*
Received 03 March 2021
Received in revised form 18 April 2021
Accepted 24 April 2021

*Keywords*:
*Fuzzy Decision Trees*
*Large Dataset*
*Fuzzy Entropy*
*Fuzzy Partitioning*
*Incremental Approach*

*ABSTRACT*

The decision tree is one of the most important algorithms in the classification which offers a comprehensible model of data. In building a tree we may encounter a memory limitation. The present study aims to implement an incremental scalable approach based on fast splitting, and employs a pruning technique to construct the decision tree on a large dataset to reduce the complexity of the tree. The proposed algorithm constructs the decision tree without storing the entire dataset in the primary memory via employing a minimum number of parameters. Furthermore, the *J*-max Pre pruning method was used to reduce the complexity with acceptable results. Experimental results show that this approach can create a balance between the accuracy and complexity of the tree and overcome the difficulties of the complexity of the tree. In spite of the appropriate accuracy and time, the proposed algorithm could produce a decision tree with less complexity on a large dataset.

***doi**: 10.5829/ije.2021.34.08b.01*

## 1. INTRODUCTION

Todays, a large amount of data is stored in a variety of information sources which can be used as valuable knowledge. In order to analyze and process the data and extract the knowledge, the data mining process is used in different ways [1]. Classification is one of the most widely used methods for data mining in order to provide a model for specifying the label of different samples based on their characteristics. In this regard, the decision tree is one of the most widely used algorithms which can produce understandable human descriptions of relationships in a dataset [2]. Further, this algorithm is one of the most widely used algorithms in pattern recognition domain due to its simplicity and interpretation, rule representation in a hierarchical format, cost and time of proper construction, the ability to work with continuous and discrete data, the need for prior knowledge and accurate presentation.

The C4.5, ID3, and CART are regarded as the most common decision tree algorithms. These algorithms have two phases: growth, pruning. In the growth phase, the dataset is recursively divided so that all records within a section can have the same class while the nodes are repeatedly pruned to prevent overfitting phenomena in the pruning phase [1]. Recently, an algorithm was developed to construct a decision tree focusing on the construction time presented to maximize accuracy. The strength of this algorithm is the construction of the tree in a limited time. In other words, when there is enough time to build a tree, the algorithm should choose the feature to split having the most benefit while it chooses the most effective feature in terms of time when the time is limited. However, the method has not been implemented for big data with respect to the complexity of the resulting tree [3].

**1. 1. Complexity of Decision Tree** The interpretation ability is one of the most important benefits of the decision tree. However, there is a negative relationship between the tree dimensions and interpretation ability. In other words, an increase in the

*Corresponding Author Institutional Email: m.ghasemzadeh@yazd.ac.ir (M. Ghasemzadeh)

Please cite this article as: S. Lotfi, M. Ghasemzadeh, M. Mohsenzadeh, M. Mirzarezaei, The Construction of Scalable Decision Tree based on Fast Splitting and J-Max Pre Punning on Large Datasets, International Journal of Engineering, Transactions B: Applications, Vol. 34, No. 08, (2021) 1810-1818

complexity of the tree results in decreasing the interpretation ability [4]. It is worth noting that the complexity of the tree is measured by some criteria such as the total number of nodes, the tree levels, the depth of the tree, and the number of traits used. Nevertheless, the complexity of the tree can be controlled by specifying the appropriate stop condition or using the pruning method on the tree. The growth phase continues until a stop condition occurs. The pre-and post-pruning methods are regarded as the two basic pruning approaches. In the first approach, several limitations are applied during the construction of the tree while it is simplified after the construction of the whole tree. The calculations of the entire data can be regarded as the traditional criteria for determining the best attribute for the split. Some of them utilized discrete methods to select numerical features while some used costly evaluations. However, none of these methods are appropriate for dealing with large dataset. Data integration in an incremental form through using all dataset is another method for constructing decision trees. However, this method disregards the complexity of the tree, and interpretation ability of DTFS algorithms are more efficient than all other methods due to the use of all trained data, the lack of computational overhead, as well as the use of a quick criterion for splitting continuous variables [5]. Ignoring the tree complexity and the effect of pruning on the accuracy of the tree are considered as two disadvantages of this method.

**1. 2. Incremental Scalable Method** The present study proposes an incremental scalable approach based on fast splitting and pruning for the decision tree construction in order to reduce the complexity of the tree. In this regard, the decision tree is constructed from all the trained data, without storing all data in memory. The use of all training data results in increasing the reliability of the model. In the proposed method, the data are incrementally entered into the tree and accordingly the decision tree is developed. In order to control the complexity of the tree, the *J*-Max pre-pruning is used [6]. The proposed approach focuses on both challenges of identifying the best attribute for development and losing interpretation capability, despite a large amount of data. The simulation results indicate that the proposed algorithm can balance the accuracy and complexity of the tree.

The present study is organized as follows: Section 2 presents the general issues related to the decision tree. In Section 3, the decision-making tree works on large datasets are reviewed and a new category is provided for these algorithms. In Section 4, an algorithm is proposed based on using an incremental and pre-pruning method to construct a tree. Section 5 indicates the data analysis results and are compared with other algorithms. Finally, the conclusion is explained in Section 6.

## 2. BACKGROUNDS

**2. 1. Decision Tree** The decision tree has a hierarchical structure and supervised learning that implemented using divide and conquer strategy. In this method, features are used for data classification as a tree structure. Tree nodes are connected by edges. Edges are conditions that are split at each node. Each extracted rules are a unique path from the root to leaf nodes.

The growth phase is the primary phase of making the decision tree by which, the training dataset is recursively partitioned until all records within a section have the same class. Each partition adds a new node to the tree. For a set of records inside *P*, the condition *t* is determined for further segmentation of the set into *P1, P2, ... Pm*. Then, the new nodes *P1* to *Pm* are created and inserted as the children of *P*. The node *P* is labeled with the condition *t* and the nodes *P1* to *Pm* are recursively split. They are not split up if all records within a section represent a class. The node is considered as a leaf and is labeled with the same class. After constructing the decision tree, the tree starts to scroll from the root to the end of one of the leaves for classifying a new record. Finally, the leaf label is returned as the result [7].

**2. 2. Splitting Measures** Selecting the attribute for branching is one of the main problems in the decision tree. Based on the type of data, the split in each node can be binary or multiple. The branch type is most often binary if the values of the attributes are continuous. However, the split may be binary or multiple if the attribute values are discrete. In order to select the best attribute in node branching, the degree of purity and distribution of the homogeneous class in each category should be considered. In order to determine the degree of purity and select the appropriate attribute for node expansion, various criteria such as the Gini's criterion and Entropy are implemented. The following formula is used for calculating each of these values [8]:

Entropy determines the purity of a set of data, which represents the qualitative division of the training examples based on a feature. If the target feature includes *c* different values, the occurrence probability of each is *Pi*, and the entropy I is defined as :

$$Entropy\,(I) = \sum_{i=1}^{c} -p_i \log_2 p_i \quad (1)$$

Based on the irregularities as impurities in a set of training examples, the effectiveness of an attribute is defined in data classification. The criterion is regarded as the expected reduction in irregularity, which is obtained by separating the examples based on this attribute. The information gain of a feature is the amount of entropy reduction, which is obtained by separating the samples through this feature. The information gain of a feature such as *A*, for the value of *I*, is defined as Equation (2):

$$Gain(I,A) = Entropy(I) - \sum_{v \in Values(A)} \frac{|I_V|}{|I|} Entropy(I_V) \quad (2)$$

In the above equation, the first item is the amount of data entropy and the second item is the expected entropy value after the data separation. Value *A* represents a set of all possible values for attribute A. *Iv* indicates a subset of *I*, in which the values of attribute *A* is *v*. With respect to feature *A*, the uncertainty of the entity is obtained from Equation (3):

$$SplitInfo(A) = -\sum_{i=1}^{c} \frac{\#classj}{I} \log_2 \frac{\#classj}{I} \quad (3)$$

Equation (4) is used to calculate the ratio of information gain. The Gain Ratio indicates how much the feature separates the data uniformly. The denominator eliminates those features having large amounts of uniform distribution values.

$$GainRatio(A) = \frac{Gain(I,A)}{SplitInfo(A)} \quad (4)$$

**2. 3. Stopping Criteria** Stopping condition in the growing phase is regarded as another important variable for decision tree algorithms. The growth phase is completed when the resolution is no longer useful, or a category can be applied to all instances in the subset [9]. The general rules for stopping are as follows:

- •All examples of training data sets belong to a same class.
- •The tree has reached the maximum depth.
- •The number of samples in a leaf node is less than the minimum number of parent samples.
- •The number of records in the current node is less than the threshold value.
- •The selection criterion is less than one threshold.

**2. 4. Decision Tree Pruning** In general, the interpretation ability is one of the distinguishing features of the decision tree which are considered more than other tree features by researchers. Importantly, a decrease in tree complexity leads to a decrease in interpretation ability. By increasing the complexity of the decision tree, a considerable increase takes place in the occurrence probability of overfitting. In addition, the training error decreases while the test error increases [2]. The reason for the occurrence of this phenomenon is the noise in the training dataset or inappropriate selection of training data.

Pruning approach is regarded as one of the most common ways to reduce the complexity, overcome the overfitting, and finding the appropriate tree size. Further, the pruning can reduce the decision tree size by removing those parts of the tree having little power for classification. Furthermore, pruning leads to a reduction in the complexity of the final classification and an improvement in the prediction accuracy. The pruning aims to extract those sub-trees which prevent the occurrence of overfitting phenomena.

In general, pruning methods are divided into two pre and post-pruning groups. Pre-pruning is implemented during the tree development in the growth phase in order to prevent excessive tree growth, useless branching, and rapid tree stopping in the growth phase. In addition, this method is utilized to reduce the time and memory required. However, the early stop of the tree is the main disadvantage of this method, which may obtain better results during the continuation of tree expansion. As for the post-pruning approach, a number of branches are removed by using statistical tests after building the entire tree. This method is used to create a balance between the accuracy and complexity of the tree. However, the computational overhead due to the processing after the tree construction is regarded as the main limitation [10]. Since the tree must be completely constructed and a lot of time and memory should be allocated to the tree construction despite a large amount of data during the post-pruning methods, the pre-pruning method was implemented in the present study. As building a complete tree is not necessary for pre-pruning techniques, it can be useful for large-scale applications.

The J-measure is one of the pre-pruning methods, used as an information theory tool to measure the content of the rules extracted from the tree [11]. Assuming that the form of the rules extracted from the tree is in the form if Y=y Then X=x, the value of the information content of the rules is calculated by using Equation (5) :

$$J(X;Y=y) = p(y)\, j(X;Y=y) \quad (5)$$

*p(y)* represents the probability which the preceding rule occurs, and j(X:Y=y) is calculated by Equation (6):

$$j(X;Y=y) = p(x|y)\log_2\left(\frac{p(x|y)}{p(x)}\right) + (1-p(x|y))\log_2\left(\frac{(1-p(x|y)}{1-p(x)}\right) \quad (6)$$

*J* pruning method is presented based on the *J* measurement criterion for reducing the overfitting. In this way, the value of j is calculated for each tree node. If j-value of a node was less than its father value, the branch should be pruned accordingly. Otherwise, the process will continue. The reduction of the number of rules or nodes with acceptable accuracy which reduces the occurrence of overfitting is the main advantage of using this pruning method. However, the *j*-pruning technique may be locally optimal since the value of a node pruned due to the less value of *j* for its father may increase in subsequent branches and accordingly reduces the tree efficiency.

In order to overcome the problem at hand, *j*-pruning was developed, by which a new approach called "*J*-max" was presented. Based on this method, in addition to the *J*-value, the *J*-max value of that node is computed by using Equation (7). The *J*-max value is checked if the j-value of a node is less than its father's value. In addition, the tree growth continues because the *J*-value may increase again if the J-max value of that node is greater than its previous *j*-values. The growth continues until the *j*-value and *J*-max ares equal. In order to calculate the J-max value, the following formula is used [6]:

$$J_{Max} = p(y), \max\{p(x \mid y), \log_2(\frac{1}{p(x)}), (1-p(x \mid y))\log_2(\frac{1}{1-p(x)})\} \quad (7)$$

Based on the results, compared to the *j*-pruning technique, this approach can reduce the number of rules or nodes and improve accuracy in most cases. Therefore, *j*-max pruning method was used in the proposed algorithm.

## 3. RELATED WORKS

Generally, the algorithms used to construct decision trees from large datasets are classified into sampling, data list, and incremental categories.

**3. 1. Sampling Algorithm** Based on sampling category, the samples are first selected from the main collection. Then, the decision tree algorithm is applied to the selected sample. In order to prevent the storage of all data in the main memory, ICE and BOAT, divide the training data into some sections. For each section, a decision tree is created by using a traditional algorithm such as C4.5 or CART, and the likes. In the next stage, the decision tree of each segment is individually processed or combined without any need to get all the data in memory. High flexibility in dealing with increasing or decreasing data in partitions is regarded as the major benefit of this method although the reliability of the model is low due to the lack of the use of the entire data in this method. The timing of the sampling algorithm, the dependence of the results on the sampling technique and the time required to construct the tree for a set of different data are some challenges of this method [12,13].

**3. 2. Data List Algorithm** In order not to store all data in the main memory, this approach implements a list of structures for each feature which is mainly stored in the disk memory and it is used for devising and developing, instead of using the records. The SLIQ is considered as one of the algorithms for this method, which creates a list structure for each attribute storing in the disk space [14]. In addition, the algorithm creates an additional list, including a class of each instance, along with the number of tree nodes, which saved the sample and keeps it in the main memory. Given that the magnitude of this list relies on the number of records, it may create some problems in a large dataset. Further, the SPRINT algorithm is an improved SLIQ method. Storing a separate list in the main memory is not essential when a column is added to the list structure for the maintenance class of each instance. However, the entire list should be read from the disk memory for each expansion. A dual space of the training data is essential as the magnitude of the list structure is proportional to the number of records in each branch in both of the above-mentioned methods. Despite a large amount of data and reading from the disk, the implementation time of the algorithm is high. Thus, list structure processing is performed in a multiprocessor or parallel manner in order to solve the problem at hand [15]. Further, the Rainforest algorithm uses the list to display the features, while the different values of the attributes are only kept, which results in decreasing the number of records. The present algorithm aims to reduce the space occupied by memory, due to the type of list structure of the related feature. However, the list should be kept in memory as an increase takes place on the list size if various values of a variable are available. All the data must be read twice and written once when a list is made, which is not appropriate for large datasets [16].

**3. 3. Incremental Algorithm** Based on the incremental algorithm, the data contributes to the tree structure in order to make the decision tree by using the entire dataset [17]. The VFDT algorithm included in this category makes the decision tree constantly, irrespective of the number of samples. First, the records should be randomly selected in order to create the original tree. Then, all the records are entered into the tree incrementally and scrolled through. In the next stage, the information gain is calculated when the number of records in a leaf reaches a certain number. The VFDT algorithm must compute all the different classification conditions for all numerical attributes, which is very time-consuming due to the diverse data [18]. DTFS is another algorithm which adds data to the tree incrementally in order to exclude the training data in the main memory and solve the computational overhead problem. Based on this algorithm, the Gain Ratio is used to select the best feature for branching. Only the *s* stored record in the node is considering for development and accordingly, the best split ratio is searched among the *s* records. Then, the mean values for all records in a class are calculated for each variable as the split criterion. As a result, Gain Ratio is calculated based on the obtained values, upon which the best feature is selected. Selecting the split property using the *s* records accelerates the process of branching [4]. Table 1 indicates a comparison

**TABLE 1.** Comparison of decision tree construction algorithms in the face of large datasets

| Method | Advantages | Disadvantages |
|---|---|---|
| SLIQ | The basis for the development of many algorithms<br>Speed up training time | Needing extra space<br>Keeping definition lists |
| SPRINT | Reducing memory overhead than SLIQ | |
| CLOUDS [19] | Decreasing the time of selection | |
| RainForest | Providing a storage method<br>Reducing memory overhead | |
| BOAI [20] | Speeding up the training time | |
| BOAT | Building a tree with a double-scanned data | Using a small subset of data<br>Wasting time due to data selection |
| ICE | Having high flexibility in dealing with increasing or decreasing data in partitions | |
| VFDT | High speed | Needing preprocess<br>Needing to set many parameters<br>Wasting time due to the calculation of all branching states |
| DTFS | High speed<br>Lacking memory overhead<br>Splitting fast continuous data | The complexity of the tree<br>Needing to preprocess |

of the decision tree algorithms for large datasets. In this table, the algorithms are compared in terms of criteria such as the speed of decision tree construction, overcoming the challenge of memory limitation, the need for re-scanning the dataset, using all or part of the dataset, and the need to adjust multiple parameters.

Due to this, the DTFS is selected as the basis for the proposed algorithm. The details of this algorithm are discussed further. The use of all training data in the tree construction, lack of memory and time overload due to the lack of using a special data structure, the simplicity of implementation, and an appropriate timeframe are regarded as some advantages of the DTFS algorithm. In addition to all these benefits, the following points should be taken into consideration in the basic approach:

• In the DTFS method, no preprocessing is used to determine the order of data entry unless the records are uniform in terms of class variables. However, it is essential to consider the time of making the main body of the tree and using the appropriate data to increase performance. For this purpose, it is important to consider pre-processing on the data in order to determine the priority of entering the tree.

• The time to develop the decision tree in the DTFS method is based on the parameter *s*, and the node is developed if the number of records stored in the node is greater than the parameter *s*, which is constant from beginning to the end of tree construction. Naturally, the behavior of tree development in the roots should not be similar to the leaves of the final stages. In addition, in a large tree, it may be stored in a large number of leaves, less than *s* records. In this case, there are many leaves which never meet the development conditions. Further, the total records stored in these leaves can be regarded as a limiting factor for memory.

• In the DTFS, the size of the tree and its complexity is deemphasized. The results of some studies indicated that changing the value of the parameter *s* does not play a significant role in the runtime and accuracy of the algorithm. However, the effect of this parameter on the complexity of the tree has not been addressed yet.

## 4. PROPOSED METHOD

In the present study, a new method was presented for constructing a decision tree based on the incremental method which can create a balance between accuracy and complexity, in addition to overcoming the challenge of the large dataset. In decision tree construction, all the data should be simultaneously present in the memory in order to determine the best attribute for development. Memory restrictions may prevent the calculation of the best feature by increasing the number of records. In addition, reducing the interpretation ability and increasing the error rate are regarded as the main challenge for high data. In order to overcome the problems of not locating training data in main memory, along with computational overhead, the algorithm incrementally injects the training data into the tree. In this way, each record is scrolled in the tree and stored in the leaves. When the algorithm decides to develop a node, only the label of the input edge of the leaf is updated if all the records in the leaf belong to the same class while the leaf begins to develop if the classes are different. The *J*-max measure is used to decide on the branching in order to reduce the complexity of the developmental leaves. In the present study, the proposed method focuses on both the main memory challenge and the complexity of the tree. The principles of the proposed method are summarized as follows:

**4. 1. Determining the Priority of Entering Records into the Tree** First, the similarity criteria, based on the type of input data, are used for determining the

priority of entering the records into the tree. Then, the amount of similarity between different records is calculated and accordingly a number indicating the similarity of a record with other records is obtained. In the next stage, the records are sorted by the descending order of similarity. Finally, based on the results, those data having the most similarity with others are placed at the beginning, while those having the least similarity with others are placed at the end. Based on this approach, those records having better quality are used in the early stages of tree construction.

**4. 2. Selecting a Feature for Branching** Gain Ratio is regarded as one of the most important criteria for splitting a decision tree in traditional algorithms. This method is time-consuming because each branch should compute all available states for all the features, and select the features with the highest Gain Ratio value. Given the timeliness of calculating this criterion, the proposed algorithm only uses the records stored in that node to compute and select the best feature while deciding to split a node. In categorical variables, each value is calculated as the split criterion while the mean values appearing for all records of a class are regarded for calculating the split criterion in continuous variables. Finally, the best feature is selected as the Gain Ratio values. Choosing the best feature for branching based on using records within a node accelerates the process of branching.

**4. 3. Reducing Complexity** The algorithm starts constructing the tree with an empty node called root. At the beginning, the root of the tree is a leaf. The training records are entered into the root, and scrolled through the tree to the end of a leaf and stored in the leaf. When the number of records stored in one sheet reaches its maximum number, the algorithm can follow one of the following alternatives:

• If all records stored within a node are in the same class, the leaf is not expanded, and only the edge of that node is updated while the records stored in this node are deleted.

• If the included records in this leaflet belong to different classes, an edge should be created for each attribute value after selecting the appropriate branching attribute based on the aforementioned formulas. *J* and *J*-max values are calculated by using Equations (6) and (7) for each of the possible edges in the developmental leaf. Then, the conditions for creating the new node are studied as follows:

1) If the *j*-value of a node is greater than its father's value, the node development is performed.

2) If the *j*-value of a node is less than its father's value, the *J*-max value of that node is checked.

3) If the *J*-max of that node is greater than its previous *J*-values, the tree node development is performed. If node *j*-value is equal to *J*-max, the node development is stopped.

The records for each edge are stored inside the leaf related to that edge. The mean values in that class are assigned as the edge label. The previous level node is converted to an internal node and accordingly, the records stored in this node are deleted. Then, the inference phase is completed after scrolling and processing all records. Finally, the majority class is assigned as a label to all leaves. If a node is empty, the majority of the parent class is given to.

For example, assume that based on the records in a node, the *Y* attribute is chosen as the best attribute for the branch, and the values of this property are *y1*, *y2*, *y3*. There are three possible branches for this node. Each tree rule is as follows:

$$If\ \ X = x1\ then\ \ Class\ \ C1$$
$$\left(J - value = 0.00113,\ Jmax = 0.02315\right)$$

$$If\ \ X = x1\ and\ \ Y = y1\ then\ \ Class\ \ C1$$
$$\left(J - value = 0.0013,\ Jmax = 0.01157\right)$$

$$If\ \ X = x1\ and\ \ Y = y2\ then\ \ Class\ \ C1$$
$$\left(J - value = 0.00032,\ Jmax = 0.0116\right)$$

$$If\ \ X = x1\ and\ \ Y = y3\ then\ \ Class\ \ C2$$
$$\left(J - value = 0.0032,\ Jmax = 0.001\right)$$

As for the rule 2, the branching is created as the node *j*-value is greater than father *j*-value, based on rule 3, the value of *j* is less than the value of *j* of his father, but since the *J*-max value is greater than the value of *j* in the previous step, this split occurs. In the case of rule 4, none of the branching conditions has been established.

**4. 4. Scrolling Trees and Classifying Samples** The tree scroll starts from the root and extends to the depths of the tree to reach a leaf, based on the split characteristics and the edge values. A path is selected for scrolling, and the difference of the value of the property with the average value calculated for that edge is the lowest value. The classification process in the decision tree algorithm for a new record involves scrolling the tree from root to reach a leaf. Then, the class corresponding to that leaf is determined as the class to the desired sample.

**4. 5. Analyzing Time Complexity** The time complexity of the algorithm for a dataset with record *m*, feature *d*, and maximum value *k* for each attribute consists of three parts as follows:

• The similarity of records is calculated with each other and the records are sorted based on their similarity,

which uses a sorting algorithm of $O(m\log m)$ in the worst case.

• The scrolling cost for each record is equal to the maximum depth of the tree, $O(\log m)$, and the cost for the entire dataset is equal to $O(m\log m)$ .

• The Gain Ratio should be calculated to select the best attribute for splitting in each node with the *s* record for all attributes and the time allocated $O(s,d)$ . As this criterion is calculated $\frac{m}{s}$ times, the cost of this part of the algorithm is $O(s,d,\frac{m}{s}=d,m)$. The *j* and *J*-max values for each selected feature should be computed at most *k* times and the execution time for this part of the algorithm equals to $O(d,k,m)$ .

• In sum, the time complexity of the algorithm is equivalent to $O(2m\log m+dkm)=O(m\log m)$ .

## 5. EXPERIMENTAL RESULTS

In the present study, a method was presented for constructing a decision tree with an incremental approach and *J*-max pruning. The records of training dataset were entered into the decision tree and accordingly the decision tree was developed in an appropriate time. In the proposed method, the records were entered into the tree based on the priorities specified and placed in the appropriate leaf. Gain Ratio was used to determine the split priority in each node. Finally, the *J*-max approach was implemented to balance the accuracy and complexity of the developmental leaves. In this section, the proposed method is evaluated and compared with C4.5 and DTFS algorithms in terms of accuracy, runtime, and complexity.

In order to implement the proposed method and other tree algorithms, a system with a RAM of 6GB, CPU2.1 GHz, 64-bit Win10 operating system was used. During the implementation and comparison of algorithms, the same conditions were used for execution. Table 2 displays the dataset used for testing, taken from UCI [21], without any missing value. Then, the Hold Out method was used to divide the data. At each run, 70% of the dataset was considered for training and 30% of the total were implemented for evaluation. In the next stage, in order to validate the results, the Hold Out method was repeated 10 times and the average results were reported. In all experiments, the algorithms C4.5, C4.5 through applying *J*-max pruning, DTFS with different values of *s*, and the proposed method were compared.

Table 3 illustrates the depth of the tree and the accuracy of the algorithms on the MagicGamma set. The tree depth, which is regarded as a criterion for tree complexity, was modified in DTFS for various *s* values. In addition, the tree complexity is high in C4.5, and the driller could not play a significant role in reducing complexity. The proposed algorithm was obtained by creating a balance between the accuracy and complexity.

As for the DTFS algorithm, the accuracy decreases when *s* is low. As shown in Table 5, in the Poker dataset, for *s = 10*, the precision of the algorithm is *50%*, which increased by increasing *s*. However, this is not always true. For example, the accuracy is decreased again at *s = 600*. The decision tree is very complicated, although the C4.5 algorithm has the highest output accuracy. The results indicated that the proposed algorithm could obtain a balance between the accuracy and complexity. Other parameters indicating the complexity of the tree, including the number

**TABLE 2.** Dataset used in the experiments.

| Dataset | Number of attributes | Number of records |
|---|---|---|
| Magic Gamma | 11 | 19020 |
| Statlog(Shuttle) | 9 | 58000 |
| Poker | 11 | 1025010 |

**TABLE 3.** Accuracy, Depth and Nodes of decision trees for the MagicGamma dataset

| Algorithm | Test Accuracy | Tree Depth | Number of Nodes |
|---|---|---|---|
| C4.5 | 69.96 | 191 | 12292 |
| C4.5+Jmax | 70.8 | 186 | 7378 |
| DTFS(S=10) | 67.72 | 110 | 3208 |
| DTFS(S=100) | 68.5 | 59 | 343 |
| DTFS(S=200) | 67.64 | 44 | 172 |
| DTFS(S=400) | 69.34 | 27 | 82 |
| DTFS(S=600) | 71 | 19 | 55 |
| Proposed Method | 68.48 | 30 | 103 |

**TABLE 4.** Accuracy, Depth and Nodes of decision trees for the Poker dataset

| Algorithm | Test Accuracy | Tree Depth | Number of Nodes |
|---|---|---|---|
| C4.5 | 54.28 | 85 | 8256 |
| C4.5+Jmax | 53.78 | 69 | 2580 |
| DTFS(S=10) | 50.18 | 30 | 1171 |
| DTFS(S=100) | 53.16 | 24 | 598 |
| DTFS(S=200) | 53.82 | 25 | 310 |
| DTFS(S=400) | 53.77 | 28 | 235 |
| DTFS(S=600) | 54.24 | 18 | 107 |
| Proposed Method | 53.41 | 30 | 231 |

**TABLE 5.** Accuracy, Depth and Nodes of decision trees for the Shuttle dataset

| Algorithm | Test Accuracy | Tree Depth | Number of Nodes |
|---|---|---|---|
| C4.5 | 81.25 | 105 | 2548 |
| C4.5+Jmax | 80.78 | 87 | 1899 |
| DTFS(S=10) | 81.34 | 31 | 781 |
| DTFS(S=100) | 79.23 | 17 | 196 |
| DTFS(S=200) | 79.01 | 15 | 115 |
| DTFS(S=400) | 78.69 | 11 | 64 |
| DTFS(S=600) | 78 | 13 | 85 |
| Proposed Method | 78.65 | 15 | 126 |

of nodes and the number of leaf nodes, which in fact represents the number of rules derived from the decision tree, are given in the table for the three test datasets.

## 6. CONCLUSION

In this paper, an algorithm was developed to construct decision trees on large datasets. In the proposed approach, the record entry priority is assigned to the tree. Then, the set of training data is based on the specified priorities of the record entered into the decision tree and placed in the appropriate leaf. For each node, its complexity is calculated at appropriate times, and it is decided whether to split it or not. Gain ratio has been used to determine the best feature for branching. Finally, to avoid the complexity of the tree and to balance the precision and complexity, the developmental leaves are pruned using the *J*-max criteria.

In general, the most important components of the proposed framework are: achieving a balance between the precision and complexity using the pre-pruning approach, solving the memory limitation problem for a large dataset by entering data incrementally into the decision tree, increasing reliability of the model by making decision tree of all training data, lacking of memory and time overload due to the non-use of especial data structure. One of the unfortunate results in our algorithm is that the number of nodes and tree depth is slightly higher when it reaches the same accuracy as the DTFS algorithm. Of course, we have eliminated the dependence of the results on the value of the *s* parameter, and it is important to strike a balance between the accuracy and complexity of the tree. Also, with the use of a pruning method, the development of nodes is largely prevented so that the tree's complexity is kept to an acceptable level. Other decision criteria for nodal branching can be considered for further work. Also, using parallel computing can increase speed of the algorithm.

## 7. REFERENCES


1. Agarwal, S., "Data mining: Data mining concepts and techniques" In 2013 International Conference on Machine Intelligence and Research Advancement, (2013), 203-207, IEEE, doi: 10.1109/ICMIRA.2013.45.
2. Qolipour, F., Ghasemzadeh, M. and Mohammad-Karimi, N., "The Predictability of Tree-based Machine Learning Algorithms in the Big Data Context." ***International Journal of Engineering, Transactions A: Basics,*** Vol. 34, No. 01, (2021), 82-89, doi: 10.5829/ije.2021.34.01a.10.
3. Chen, Y.L., Wu, C.C. and Tang, K. "Time-constrained cost-sensitive decision tree induction." ***Information Sciences***, Vol. 354, (2016), 140-152, doi: 10.1016/j.ins.2016.03.022.
4. Priyanka and Kumar, D., "Decision tree classifier: a detailed survey." ***International Journal of Information and Decision Sciences***, Vol. 12, No. 3, (2020), 246-269, doi: 10.38094/jastt20165.
5. Franco-Arcega, A., Carrasco-Ochoa, J.A., Sánchez-Díaz, G. and Martínez-Trinidad, J.F., "Decision tree induction using a fast splitting attribute selection for large datasets." ***Expert Systems with Applications***, Vol. 38, No. 11, (2011): 14290-14300, doi: 10.1016/j.eswa.2011.05.087
6. Stahl, F. and Bramer, M., "Jmax-pruning: A facility for the information theoretic pruning of modular classification rules." ***Knowledge-Based Systems***, Vol. 29, (2012), 12-19, doi; 10.1016/j.knosys.2011.06.016.
7. Grossi, V., Romei, A. and Turini, F., "Survey on using constraints in data mining." ***Data Mining and Knowledge Discovery***, Vol. 31, No. 2, (2017): 424-464, doi: 10.1007/s10618-016-0480-z.
8. Chandra, B., Kothari, R. and Paul, P., "A new node splitting measure for decision tree construction." ***Pattern Recognition***, Vol. 43, No. 8, (2010), 2725-2731, doi: 10.1016/j.patcog.2010.02.025.
9. Lomax, S. and Vadera, S., "A survey of cost-sensitive decision tree induction algorithms." ***ACM Computing Surveys (CSUR),*** Vol. 45, No. 2, (2013), 1-35, doi: 10.1145/2431211.2431215.
10. Brunello, A., Marzano, E., Montanari, A. and Sciavicco, G., "Decision tree pruning via multi-objective evolutionary computation." ***International Journal of Machine Learning and Computing,*** Vol. 7, No. 6, (2017), 167-175, doi: 10.18178/IJMLC.2017.7.6.641.
11. Bramer, M., "Using J-pruning to reduce overfitting in classification trees." ***In Research and Development in Intelligent Systems XVIII***, Springer, London, (2002), 25-38, doi: 10.1007/978-1-4471-0119-2_3.
12. Manapragada, C., Webb, G. I., and Salehi, M., "Extremely fast decision tree." In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, (2018), 1953-1962, doi: 10.1145/3219819.3220005.
13. Gehrke, J., Ganti, V., Ramakrishnan, R. and Loh, W.Y., "BOAT-optimistic decision tree construction." ACM, Vol. 28. No. 2, (1999), doi: 10.1145/304182.304197.
14. Mehta, M., Agrawal, R. and Rissanen, J., "SLIQ: A fast scalable classifier for data mining" International conference on extending database technology, Springer, Berlin, Heidelberg, (1996), 18-32, doi: 10.1007/BFb0014141.
15. Zaki, M. J. "Parallel and distributed data mining: An introduction." In Large-scale parallel data mining, Springer, Berlin, Heidelberg. (2000), 1-23, doi: 10.1007/3-540-46502-2_1.
16. Gehrke, J., Ramakrishnan, R. and Ganti, V., "RainForest—a framework for fast decision tree construction of large datasets." ***Data Mining and Knowledge Discovery***, Vol. 4, No. 2, (2000), 127-162, doi: 10.1023/A:1009839829793.

17. Hulten, G. and Domingos, P., "Mining Decision Trees from Streams." In ***Data Stream Management***, Springer, Berlin, Heidelberg, (2016), 189-208, doi: 10.1007/978-3-540-28608-0_9.
18. Domingos, P. and Hulten, G.,"Mining high-speed data streams." In Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, (2000), 71-80, doi: 10.1145/347090.347107.
19. Ranka, S. and Singh, V., "CLOUDS: A decision tree classifier for large datasets." In Proceedings of the 4th Knowledge Discovery and Data Mining Conference, Vol. 2, No. 8, 1998.
20. Yang, B., Wang, T., Yang, D. and Chang, L., "BOAI: Fast alternating decision tree induction based on bottom-up evaluation." In Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, Berlin, Heidelberg, (2008), 405-416, doi: 10.1007/978-3-540-68125-0_36.
21. Blake, C.L. and Merz, C.J., "UCI Repository of machine learning databases [http://www. ics. uci. edu/~ mlearn/MLRepository. html]. Irvine, CA: University of California." Department of Information and Computer Science 55 (2008).

## Persian Abstract

**چکیده**

درخت تصمیم یکی از مهمترین الگوریتم‌های طبقه‌بندی است که مدل قابل درکی از داده‌ها ارائه می‌دهد. در ساخت درخت تصمیم ممکن است با محدودیت حافظه روبرو شویم. در مطالعه حاضر رویکرد مقیاس‌پذیر افزایشی بر مبنای تقسیم سریع و هرس، جهت ساخت درخت تصمیم بر روی‌داده‌های حجیم ارائه‌شده‌است به‌نحوی‌که پیچیدگی ساخت درخت کاهش یابد. الگوریتم ارائه‌شده درخت تصمیم را بدون نیاز به ذخیره‌سازی کل مجموعه‌داده در حافظه اصلی و با حداقل تعداد پارامتر لازم مهیا می‌سازد. همچنین جهت کاهش پیچیدگی و دستیابی به‌دقت قابل‌قبول از روش پیش هرس مبتنی بر J-Max استفاده‌شده‌است. نتایج آزمایش بر روی مجموعه‌داده‌ها نشان ی‌دهد با استفاده از این رویکرد می‌توان به موازنه‌ای میان دقت و پیچیدگی درخت دست‌یافت و بر مشکلات حاصل از پیچیدگی درخت غلبه‌کرد. الگوریتم ارائه‌شده با وجود دقت و زمان اجرای مناسب، درخت تصمیمی با پیچیدگی‌های کمتر برروی داده های حجیم می‌سازد.