Fast Decision Tree Algorithm

Vasile PURDILĂ, Ștefan-Gheorghe PENTIUC
Stefan cel Mare University of Suceava, 720229, Romania
vasilep@eed.usv.ro, pentiuc@eed.usv.ro

Abstract—There is a growing interest nowadays to process large amounts of data using the well-known decision-tree learning algorithms. Building a decision tree as fast as possible against a large dataset without substantial decrease in accuracy and using as little memory as possible is essential. In this paper we present an improved C4.5 algorithm that uses a compression mechanism to store the training and test data in memory. We also present a very fast tree pruning algorithm. Our experiments show that presented algorithms perform better than C5.0 in terms of speed and classification accuracy in most cases at the expense of tree size - the resulting trees are larger than the ones produced by C5.0. The data compression and pruning algorithms can be easily parallelized in order to achieve further speedup.

Index Terms—algorithm, chi-merge, classification, data compression, decision tree, pruning.

I. INTRODUCTION

A pattern is a representation of an entity in a P-dimensional space $E$. Usually the pattern $x_i$ is a vector $x_i = (x_{i1}, x_{i2}, ..., x_{ip})$ (1)

where $x_{ij}$ is the feature $j$ of the pattern $x_i$ and $x_{ij} \in R, E \subset R^p$.

The patterns that are similar to each other are grouped into classes of equivalence. These classes are regions in space $E$ and depending on the separation areas between them we can have two types of class separability: linear and nonlinear. The decision binary trees can be used successfully for pattern classification, especially in the case of nonlinear class separability. The method consists of several steps; the classes that can be potentially assigned to pattern $x_i$ are rejected during the process until one of them is accepted. All these steps can be grouped into one generic step called learning step where a training set is provided to the pattern recognition system:

$$T = \{(x_i, \sigma_i) / x_i \in E_T, \sigma_i \in \{1,2,...,M\}\}$$ (2)

where $E_T \in E$ and $M$ is the number of classes in $E_T$.

The pattern space is divided into several regions, one for each class (Fig. 1). Using these regions, a binary decision tree can be created that will allow an automatic classification of any “unseen” patterns. The sequence of decisions in this tree is applied to some individual characteristics (attributes). In case of continuous attributes the tests are usually simple conditions: $is \ x_i < t$ where $t$ is a threshold value. For discrete attributes the tests are: $x_i \in \{v_{ij}, v_{i2}, ..., v_{iq}\}$ where $v_{ij}$ are the possible values of the attribute.

The root and the nodes of such tree represent the decisions of the tree and the leaves represent the outcome (the class to which a pattern belongs to). These trees are called ordinary binary classification trees (OBCTs).

Figure 1. Sample data from Iris dataset

Figure 2. A section from a decision tree built against regions from Fig. 1

The principle of pattern recognition is demonstrated against the example from Fig. 1 where $p=2$ and $M=3$. In Fig. 2 it is shown the decision tree that corresponds to the three regions delimited from Fig. 1 for classes C1 and C2. In a decision tree it is possible to classify a pattern without consulting the values of all attributes. One may see that a decision tree built against a training set $T$ is not unique. In the example from Fig. 1 the tree was built by visual examination of the geometry of pattern placement in the plane, it is obvious that this method cannot be applied for $p > 3$. A systematic method that can be applied for $p > 3$ requires the following to be taken into account:

- the set of patterns $E$ is associated to the root node
- each node $r$ has a subset of patterns $F_r$ from $E$
- by evaluating a condition to true or false the subset $F_r$ is split further into two disjoint subsets $F_{rY}$ and $F_{rN}$

In order to build the decision tree the following have to be defined:

- a criterion to decide what is the best method to split the set of patterns
- a rule to decide when to stop the set of patterns splitting; a such rule controls the tree growing and decides whether a certain node becomes a leaf.
• A decision rule for classification in leaves

The set of tests is very important in the decision tree induction process. Let \( n \) be the number of patterns from \( E \). Each attribute \( x_k \mid k=1,p \) can have a number of values \( n_k \leq n \). For each attribute there is a number of threshold values \( t_k \) where each value is approximated with the half of distance between two consecutive values of attribute \( x_k \) in the patterns space. The total number of tests is:

\[
\sum_{k=1}^{p} n_k t_k
\]  

One key aspect of decision tree induction is to minimize the number of tests that is equal to the number of nodes that are not leaves. Classification is a very important task in data mining related tasks [1] and the decision trees are probably the most used algorithms for solving classification problems. They can be used as standalone algorithms or combined with other algorithms like Adaboost [2], random trees, etc.

II. THE DATA COMPRESSION ALGORITHM

One of the most challenging tasks for data mining community is to develop classifiers that can mine very large datasets [3]. This basically means to extend the existing algorithms or develop new ones that can scale up to large datasets. The decision trees are widely used in various fields such as banking, statistics, gas and oil exploration, astronomy [4], speech recognition [5], to name just a few.

Many algorithms for learning decision trees [6-11] have been developed: ID3, C4.5, C5.0, CART, CHAID, SLIQ, Sprint, QUEST. C4.5 and C5.0 have been developed by Ross Quinlan and are considered de facto standard when evaluating new decision tree algorithms. A comparison of the above algorithms was done and the results were published on his website (http://www.rulequest.com).

An excellent survey about the scalability of decision tree algorithms was done by Provost [12]. Apparently the most common reason for scaling up is the fact that increasing the number of the training patterns will often result in a classifier that has a better accuracy. If we are to ask the question “How large is very large?” one could answer as follows [13]:

“Somewhere around data sizes of 100 megabytes or so, qualitatively new, very serious scaling problems begin to arise, both on the human and on the algorithmic side.”

One may ask the following question: why not use a well known compression/decompression algorithm like gzip for example? Unfortunately, such algorithms cannot be used as we need to read and decompress (unpack) data from random offsets within the dataset during the decision tree induction.

We propose a fast data compression algorithm that compresses/decompresses data at dataset record level and can be used in pretty much any decision tree algorithm.

A good decision tree algorithm should be able to handle patterns with two types of attributes: discrete and continuous. A discrete attribute can have a finite set of values as follows:

\[
v_i = \{ v_{i1}, v_{i2}, \ldots, v_{ip} \}
\]

where \( i \) is the \( i_{th} \) attribute and \( q \) is the number of possible values for this attribute. We can encode each of these values using a number of bits as follows:

\[
\text{bits} = \text{roundup}(\log_2(p))
\]

As the continuous (numeric) attributes can have a very large set of possible values we cannot represent these using bits as we do for discrete ones. Instead, we have to use a discretization mechanism by splitting these possible values into intervals then using a number of bits to encode each interval. The simplest method to divide these values into intervals is called equal-width-intervals, is based on information gain and consists of the following steps: sort the values ascending, split them into a number of equal length intervals, for each two adjacent intervals \( (l_{m-1}, l_m) \) and \( (l_m, l_{m+1}) \) calculate the information gain against the current dataset for splitting point \( l_m \). The threshold \( l_m \) that maximizes the information gain is chosen so the splitting criterion for attribute \( a_i \) can be represented as:

\[
v_{ij} = l_m, \quad v_{ik} > l_m \quad | \quad k \neq j
\]

These calculations have to be done on each splitting node against the dataset that reaches it which can be time consuming especially for large datasets. As we need to discretize the attributes with continuous values beforehand we can’t use this approach for our algorithm. Kerber [14] proposed a new discretization algorithm based on chi-square statistical method that calculates the splitting intervals just once for the whole dataset. This approach is also used in CHAID and related algorithms [15] and usually yields better results than the equal-width-intervals method in terms of classification accuracy.

Once all attribute values are discretized we calculate the number of bits required to store all distinct values for each one. We can represent each attribute value by a number between \( 0 \) and \( q-1 \) and pack this into a sequence of bits within a primitive data type such as int or long, depending on the operating system and processor – for a 32 bit operating system we’ll use an int (4 bytes), for a 64 bit operating system we’ll use a long (8 bytes) in order to maximize the data transfer rate between memory and processor. Each attribute will have an offset in order to identify the position of its value within the packed data. Depending on how many attributes the dataset has and the amount of bits required for each attribute we may need more than one int or long to store a record from the dataset. Packing and unpacking data are reduced to simple bit operations:

```
procedure pack(index, value, a) do
  data[index] |= value << a.offs
end

procedure unpack(index, atr) do
  mask[] = {1,3,7,15,31,63,127,255,...}
  return(data[index]>>a.offs)&mask[a.bits]
end
```

The variable \( a\text{.bits} \) is the number of bits required to encode the values for a particular attribute and \( a\text{.offs} \) is the attribute’s offset. Our experiments show that the amount of memory required to store the dataset in packed format is about one order of magnitude lower than its unpacked format. This means that we can build decision trees against much larger datasets that otherwise would not fit in computer’s memory. The packing operations for a dataset
can be easily parallelized as every thread can read and pack data at different offsets within the dataset so the memory writes operations won’t clash. A drawback of this method is that we use a fixed offset for every attribute value that is encoded. For example, if we use a multiple of 64 bits (long type) for a pattern that requires 65 bits to be encoded then 63 bits would be wasted. The solution is to pack the data more tightly - the first bit of the next pattern should be stored next to the last bit of the previous pattern - this will be a topic for our future research.

III. THE PRUNING ALGORITHM

Pruning is a very important step in a decision tree induction and it should reduce the size of a decision tree without reducing its predictive accuracy. It also reduces the risk of over fitting the training data. There are several methods of pruning decision trees in literature: minimum description length principle [16], multiple comparison (MC) analysis [17], VC dimension [18], Bayesian methods [19], forest tree pruning [20], using genetic algorithms [21]. In [22] it is presented a comparison of reduced error pruning methods. This method is known to be one of the fastest pruning methods. It uses a bottom-up approach and consists of replacing a node with a leaf where its value is one of the possible values of the attribute that is linked to that particular node. The process is an iterative one and stops when no more accuracy improvement is achieved. The node to be replaced is the one that maximizes the tree accuracy by replacing it with a leaf: We propose a very fast pruning algorithm where the calculations required to decide what node is to be replaced with a leaf are done incrementally. Every node in the tree will have three extra attributes:

- **hits** = number of patterns that reach the node
- **correct** = number of patterns classified correctly that reach the node
- **cls[]** = class distribution for patterns that reach the node

Each pattern from the test dataset will be classified by the tree just once and the variables hits, correct and cls[] will be set up accordingly for every node. Next, a list of nodes that have only leaves as children is built. For each node in the list we calculate the number of patterns classified correctly by the node (the ones that reach the node) if we would replace that node with a leaf that has the value equal to the most common class in cls[] vector.

If the following condition is true:

\[
diff = \max\{node.\text{cls[]}\} - node.\text{correct} \geq 0 \quad (7)
\]

then we add that node to a sorted hash map where the keys are sorted ascending. In this case the key is set to diff value and the value is a vector of nodes. While the sorted hash map is not empty the following steps are executed:

- get the last key from the map (the biggest one)
- read the vector of nodes for that key
- replace the last node from vector with a leaf that has the value = \(\max\{node.\text{cls[]}\}\)
- add the key value to leaf.correct
- remove the node from vector
- if the vector is empty remove the key from hash table
- for every parent node up to the root add the key value to correct value
- apply formula (7) to parent node of current node and if diff \(\geq 0\) then add the parent node to the hash table

In Fig. 3 one may see the results for Iris dataset.

```plaintext
petal_length [75/73] [25,25,25]
- [1.0 - 3.0]-Iris-setosa [25/25] [25,0,0]
- [3.0 - 4.8]-Iris-versicolor [23/23] [0,23,0]
- [4.8 - 7.9]-Iris-virginica [27/25] [0,2,5]
```

Figure 3. Results for Iris dataset

Iris dataset has a total of 150 patterns from which 75 are used as training dataset and the remaining 75 as test dataset. As one may see it is very easy to calculate the tree’s classification accuracy using the following formula:

\[
\text{accuracy} = \frac{\text{root.correct}}{\text{root.hits}} \times 100 
\]  

(8)

In Fig. 3 the first pair (75/73) from the root node represents (hits/correct) variables and the numbers enclosed in square brackets [25,25,25] is the class distribution vector for root node. Obviously, all patterns from the dataset reach the root node. The algorithm is shown in pseudo code:

```plaintext
procedure prune(tree, testDataset) do
  map = a sorted hash table
  classify(tree, testDataset)
  foreach node of tree do
    if node.hasLeavesOnly() then
      correct = max(node.cls[])
      gain = correct - node.correct
      if gain \(\geq 0\) then
        map.put(gain, node)
    end
    end
  endfor
  while map.isNotEmpty() do
    key = map.lastKey()
    node = map.lastNodeByKey(key)
    node.changeIntoLeaf()
    node.correct = node.correct + key
    node.leafVal = node.getMostUsedLeaf()
    map.removeNode(node)
    if map.lastNodeByKey(key)!null then
      map.removeKey(key)
    end
    p = node.parent
    node = node.parent
    while node !null do
      node.correct = node.correct + key
      node = node.parent
      if p!null and p.hasOnlyLeaves() then
        correct = max(p.cls[])
        gain = correct - p.correct
        if gain \(\geq 0\) then
          map.put(gain, p)
        end
      end
    end
  end
end
```

IV. EXPERIMENTAL RESULTS

In order to validate the proposed algorithms three datasets from UCI machine learning repository have been used:

- sleep - 105,908 patterns, 6 numeric attributes, 6 classes
- adult - 48,842 patterns, 6 numeric attributes, 8 discrete attributes, 2 classes
The values in bold represent better values. As one may see in Table I our algorithm outperforms C5.0 in classification accuracy in all 3 tests and is a bit faster in two of the tests. However, C5.0 produces much smaller decision trees. We repeated the experiments with the same datasets but ten times larger.

One may see in Table II that the difference between execution times widens in favor of our algorithm. The pruning algorithm complexity depends on two factors. The first one is the complexity of classify() function. If \( m \) is the number of attributes and \( n \) is the number of patterns in the test dataset then the complexity of classify() function can be expressed as \( O(mn) \) because the maximum depth of the tree is equal to \( m \). On the other hand the complexity of foreach and while loops can be expressed as \( O(sm) \) where \( s \) is the number of nodes in the tree before pruning. We’ve also included \( m \) in the calculations because of the inner while loop that is necessary to update incrementally the correct value for all parent nodes up to the root when a node is replaced with a leaf. It has been observed in our tests that the number of nodes in the tree before pruning has the same order of magnitude as the size of the test dataset so we can approximate the complexity of pruning algorithm with \( O(mn) \).

V. CONCLUSION

We proposed a very fast algorithm for building decision trees that can be used successfully against large datasets. Some parts of it can be easily parallelized (discretization of continuous attributes) which should shorten the overall execution time. Also, further research is required for data packing algorithm in order to reduce the amount of memory required.

### TABLE I. EXPERIMENTAL RESULTS

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Records</th>
<th>Errors %</th>
<th>Leaves</th>
<th>Time (sec)</th>
<th>Errors %</th>
<th>Leaves</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sleep</td>
<td>105,908</td>
<td>28.80</td>
<td>2,071</td>
<td>1.19</td>
<td>29.18</td>
<td>1,234</td>
<td>0.69</td>
</tr>
<tr>
<td>adult</td>
<td>48,842</td>
<td>13.30</td>
<td>116</td>
<td>0.40</td>
<td>13.90</td>
<td>45.16</td>
<td>0.59</td>
</tr>
<tr>
<td>poker</td>
<td>1,000,000</td>
<td>20.60</td>
<td>14,067</td>
<td>3.49</td>
<td>16.67</td>
<td>104,920</td>
<td>2.90</td>
</tr>
</tbody>
</table>

### TABLE II. EXPERIMENTAL RESULTS (DATASET X 10)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Records</th>
<th>C5.0</th>
<th>Our algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>sleep</td>
<td>1,059,080</td>
<td>15.80</td>
<td>5.40</td>
</tr>
<tr>
<td>adult</td>
<td>488,420</td>
<td>7.00</td>
<td>4.30</td>
</tr>
<tr>
<td>poker</td>
<td>10,000,000</td>
<td>26.00</td>
<td>26.00</td>
</tr>
</tbody>
</table>

### REFERENCES