CMP: A Fast Decision Tree Classifier Using Multivariate Predictions

Haixun Wang
Computer Science Department
University of California at Los Angeles
Los Angeles, CA 90095
hxwang@cs.ucla.edu

Carlo Zaniolo
Computer Science Department
University of California at Los Angeles
Los Angeles, CA 90095
zaniolo@cs.ucla.edu

Abstract

Most decision tree classifiers are designed to keep class histograms for single attributes, and to select a particular attribute for the next split using said histograms. In this paper, we propose a technique where, by keeping histograms on attribute pairs, we achieve (i) a significant speed-up over traditional classifiers based on single attribute splitting, and (ii) the ability of building classifiers that use linear combinations of values from non-categorical attribute pairs as split criterion. Indeed, by keeping two-dimensional histograms, CMP can often predict the best successive split, in addition to computing the current one; therefore, CMP is normally able to grow more than one level of a decision tree for each data scan. CMP's performance improvements are also due to techniques whereby non-categorical attributes are discretized without loss in classification accuracy; in fact, we introduce simple techniques, whereby classification errors caused by discretization at one step can then be corrected in the following step. In summary, CMP represents a unified algorithm that extends the functionality of existing classifiers and improves their performance.

1. Introduction

Classification represents a very important data mining problem [5]. The input database, also called *training set*, consists of a number of records consisting of a fixed number of attributes. Attributes whose underlying domain is totally ordered are called *ordered* attributes, whereas attributes whose underlying domain is not ordered are called *categorical* attributes. There is one distinguished categorical attribute which specifies the *class label* of each record. The objective of classification is to generate a concise and accurate description or a model for each class label in the training set. The model so generated is used to classify future datasets for which the class labels are unknown.

Classification is a well-studied problem [11] in the area of statistics and machine learning. Several classification models, including decision trees[1, 2], statistical models[3], and genetic models[4], have been proposed over the years.

Several algorithms have been proposed to construct decision trees. We will next review those that are most directly relevant to introduce our CMP method. A more general comparison of CMP with several other methods is given in Section 3..

The rest of the paper is organized as follows. In the next subsection we provide a simple description of decision tree classifiers. In Section 1.1. we introduce and compare several state-of-the-art classification algorithms, including SPRINT[17] and CLOUDS[14]. We present our CMP family classifiers in detail in Section 2., including the data structure, techniques to avoid accuracy loss, the prediction method and multivariate splitting criteria. A comparison of performance in various aspects is given in Section 3.. Section 4. concludes our work.

1.1. Decision tree classifiers

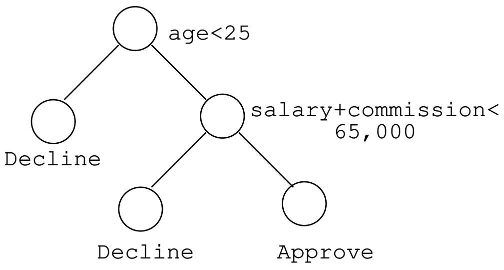
Building a decision tree from a training set consists of two phases. The initial decision tree derived in the *construction* phase may not be the best generalization due to over-fitting. So in the *pruning* phase, some branches and nodes are removed to improve the accuracy of the classifier. Typically, the time spent on pruning for a large dataset is insignificant (less than 1%) in comparison with the construction phase when the data is scanned multiple times. Therefore, in this paper we will focus on the construction phase only. For pruning , we use the algorithm in PUBLIC[9], since this is applied during the generation phase of the decision tree.

Tree building

A decision tree classifier recursively partitions the training set until each partition consists entirely, or almost entirely, of records from one class. Each non-leaf node of the tree contains a *split criterion* which is a test on one or more attributes and determines how the data is partitioned. Figure 1(b) is a sample decision-tree classifier based on the training set shown in Figure 1(a). Two split criteria, $(age < 25)$ and $(salary+commission < 65,000)$, partition the dataset into `Declined` and `Approved` classes. This classification model can then be used to process future loan applicants by classifying them into `Declined` or `Approved` classes according to the decision tree.

Age	Salary	Commission	Approval
18	20,000	0	No
60	70,000	20,000	Yes
43	30,000	1,000	No
68	40,000	26,000	Yes
32	80,000	0	Yes
20	50,000	20,000	No

(a) Training Set



(b) Decision Tree

Figure 1. The Loan Application Example

Splitting index

A splitting index is used to evaluate the "goodness" of all the potential splits according to some criterion. Several splitting criteria have been used in the past [1, 2, 11]; in this paper, we use the $gini$ index which has been shown to be very effective, and has been used in many previous experiments, thus, making it easier to compare different algorithms. Consider a training set S consisting of n records from c mutually disjoint classes. The $gini$ index for set S is defined as:

$$gini(S) = 1 - \sum_{j=1}^{c} p_j^2 \qquad (1)$$

where p_j is the relative frequency of class j in S. Thus, the more homogeneous a set is with respect to the classes of records in the set, the lower is its $gini$ index.

If S is partitioned into two subsets S_1 and S_2, the index of the partitioned data $gini^D(S, cond)$ can be obtained by:

$$gini^D(S, cond) = \frac{n_1}{n} gini(S_1) + \frac{n_2}{n} gini(S_2) \qquad (2)$$

where n_1 and n_2 are the number of records of S_1 and S_2, respectively, and $cond$ is the splitting criterion.

The SPRINT algorithm

The SPRINT algorithm has been proposed to build decision trees for large datasets[17]. The splitting criterion used by SPRINT is based on the value of a single attribute (univariate). For a continuous attribute, it will have the form of $a \leq C$ where a is an attribute and C is a constant.

SPRINT avoids costly sorting at each node by presorting continuous attributes only once, at the beginning of the algorithm. Each continuous attribute is maintained in a sorted attribute list. Each entry in the list contains a value of the attribute, the class label of the record, and its corresponding record id (rid). A best split point for a continuous attribute is derived by computing the $gini$ index on $n - 1$ possible splits in an attribute list with n distinct values. The node is split on the attribute whose split point yields the least value for the $gini$ index. A hash table of the same order as the size of the training set maps each record (using key rid) to the subnode where it belongs after the split. Each entry in the attribute list is moved to the attribute list of the subnode after consulting the hash table as to which subnode (left or right) this rid belongs to. The sorted order is maintained as the entries are moved in pre-sorted order.

Sampling and discretization

To improve time efficiency in dealing with continuous attributes in a large dataset, approximate techniques, such as sampling and discretization, are used.

Discretization is the process of transforming the numeric attribute into an ordered discrete attribute. It gives large reductions in learning time for datasets with many continuous attributes. One discretization method is based on an *equal-width* histogram that divides the range into an approximately equal widths. Another method is based on an *equal-depth* histogram (quantiling) that divides the range into sub-ranges of an approximately equal number of elements.

Another technique is sampling. C4.5 [2] uses a *windowing* technique which works as follows. A small sample is drawn from the dataset to build an initial tree. This sample is augmented with records that were misclassified in the initial tree. This process is repeated for a number of iterations.

Although approximate techniques can reduce learning time substantially, it is shown that larger training sets (typically involving several million records) improve accuracy of the classification model [12, 13]. It means that decision trees built by an *approximate* approach can carry a significant loss of accuracy in comparison with trees built by an *exact* approach like SLIQ[7] and SPRINT[17].

The CLOUDS algorithm

Instead of computing $gini$ index for each distinct value of a numeric attribute, the CLOUDS algorithm samples the splitting points followed by an estimation step to narrow the search space of the best split. In CLOUDS, the range of each numeric attribute is divided into q intervals using a quantiling-based technique. Each interval contains approximately the same number of points. For each numeric attribute, the $gini$ index is computed at the points that separate two intervals (i.e., at the interval boundaries) and the minimum $gini$ value ($gini_{min}$) is selected. Furthermore, CLOUDS estimates a lower bound $gini^{est}$ for the $gini$ value in each interval, using estimation techniques that we will discuss later. All the intervals with $gini^{est} \geq gini_{min}$ are eliminated (pruned), to derive a list of candidate intervals (called the *alive* intervals). Then, for each alive interval we evaluate the $gini$ index at every distinct point in the interval, and use the point with the globally least value for splitting. This ensures that the best split point is chosen, but unfortunately, it requires another pass over the entire dataset.

The lower bound of the $gini$ index, $gini^{est}$, is calculated using a hill-climbing algorithm for each interval. It starts from the left boundary, and calculates *gradient* of $gini$ index along all the classes. The direction (class c) along the minimum gradient is chosen, and the *gini* index is evaluated at a new splitting point, which is at the old splitting point (left boundary) plus the number of elements in the interval with class c. If the new *gini* index is less than the current $gini^{est}$, it becomes the new $gini^{est}$ and the process is repeated for the remaining classes (and smaller interval).

2. The CMP classifier

The CMP algorithm can be best presented by first considering its single-variable version, CMP-S, and then extending it to the version using bivariate histograms, CMP-B. Finally, we add the functionality of using linear combinations of attributes as splitting criteria to CMP-B and present the complete CMP. The relationships between CMP-S, CMP-B and CMP are as follows:

CMP-S	a variation of the CLOUDS algorithm, specialized to reduce disk access up to 50% (for classifying numeric attributes only).
CMP-B	CMP-S + using prediction to achieve multiple splits for each scan of the dataset.
CMP	CMP-B + using linear combination of attributes as splitting criteria.

2.1. The CMP-S algorithm

CMP-S uses the same discretization technique used in CLOUDS to reduce the computation of $gini$ index. This discretization method is based on an equal-depth histogram (quantiling) that divides a domain into a set of intervals each having approximately equal number of points.

A problem that comes with discretization is the potential loss of accuracy, since $gini$ is computed exactly only at the interval boundaries. CMP uses a simplified version of the estimation method (described below) proposed in CLOUDS to remedy accuracy loss in discretization. A major improvement of CMP-S over CLOUDS is that it manages to avoid scanning the dataset a second time to determine the exact split point.

Sampling the splitting points with estimation

For each continuous attribute a, the $gini$ index is computed at the boundary between each two adjacent intervals; let $gini_a^{min}$ denote the minimum of all of these values. Then, for each interval, we estimate the lower bound, denoted $gini_a^{est}$, using the method described below.

All the intervals with $gini_a^{est} < gini_a^{min}$ are called $alive$ intervals. We retain $alive$ intervals and discard all the rest of the intervals. CLOUDS algorithm requires another pass over the entire dataset to evaluate the $gini$ index at every distinct point in the $alive$ interval and determine the exact splitting point. However, as we will see, CMP-S manages to eliminate this extra pass.

The lower bound $gini_a^{est}$ for interval $[v_l, v_u]$ is obtained based on the *gradient* of $gini$ value at each interval boundary. In the following we describe the estimation heuristic of CLOUDS. We use the following notation:

x_i – the number of elements of class i that are less than or equal to v_l
y_i – the number of elements of class i that are less than or equal to v_u
c_i – the total number of elements of class i
n_l – the number of elements that are less than or equal to v_l $(= \sum_{i=1}^{c} x_i)$
n_u – the number of elements that are less than or equal to v_u $(= \sum_{i=1}^{c} y_i)$
n – the size of the dataset
c – the number of classes

Thus, $gini^D$ at point v_l is given by Equation 3, which is derived directly from Equation 2:

$$gini^D(S, a \leq v_l) = \frac{n_l}{n}\Big(1 - \sum_{i=1}^{c}\big(\frac{x_i}{n_l}\big)^2\Big) + \frac{n-n_l}{n}\Big(1 - \sum_{i=1}^{c}\big(\frac{c_i - x_i}{n - n_l}\big)^2\Big) \quad (3)$$

The value of the gradient along x_i is given by:

$$\frac{\partial gini^D(S, a \le v_l)}{\partial x_i} = \frac{2}{n_l(n-n_l)}(c_i\frac{n_l}{n} - x_i) - \frac{1}{n}(\frac{1}{(n-n_l)^2}\sum_{i=1}^{c}(c_i - x_i)^2 - \frac{1}{n_l^2}\sum_{i=1}^{c}x_i^2) \quad (4)$$

The lower bound $gini^{est}$ is estimated using the value of gradient in a hill-climbing manner, that is, at each point starting from the left boundary of the interval, we always choose class i which makes Formula 4 minimal. It is proved in [14] that we do not need to evaluate Formula 4 at each point in the interval. As a matter of fact, if class i is chosen at the left boundary, then the next point at which we need to evaluate the gradient is the c_i^{th} point, where c_i is the number of points of class i in that interval. Then, if class j gives the steepest descent at the c_i^{th} point, the next evaluation position will be at the $(c_i + c_j)^{th}$ point. Thus, the time requirements are proportional to the number of classes rather than the number of points in the interval.

We evaluate $gini^{est}$ from the right boundary to the left boundary, and the final estimation is determined as follows:

$$gini^{est} = min(Est\_GiniLR, Est\_GiniRL, gini^D(S, a \le v_l), gini^D(S, a \le v_u)) \quad (5)$$

where Est_GiniLR and Est_GiniRL are the minimal $gini$ index evaluated from left to right and from right to left respectively.

Number of alive intervals

The next step consists in finding the *alive* intervals (intervals whose $gini_a^{est}$ is less than $gini_a^{min}$) on the selected attribute a. The alive intervals are a small set of intervals where the actual minimum value of the $gini$ index might fall. Figure 2 shows a hypothetical $gini$ curve, and three *alive* intervals (shaded areas in the figure).

Our experiments of the estimation method are summarized in Table 1. The first four small datasets (`Letter, Satimage, Segment` and `Shuttle`) in the table are from the STATLOG project[6], and the two large datasets (`Function 2` and `Function 7`) are synthetic datasets described in [5]. In these test cases there were at most $N = 2$ alive intervals: i) the one whose left boundary (or right boundary, depending on boundary gradient) has $gini$ index $gini_{min}$ (the middle shaded area in Figure 2), and ii) the one whose estimated $gini$ index is the minimum among all the $gini^{est}$ (the rightmost shaded area in Figure 2). Furthermore these two interval often coincide, yielding only one alive interval, especially for large datasets.

Unlike CLOUDS algorithm, which makes a second pass over the entire dataset to determine the splitting attribute

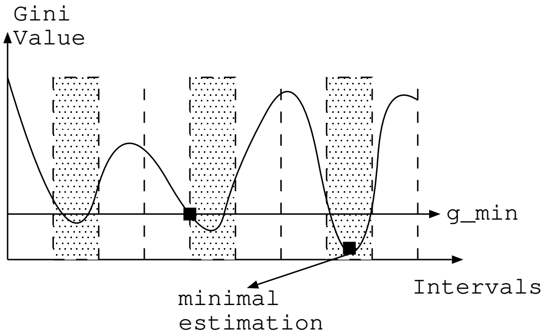


Figure 2. $Gini$ Index Estimation and Alive Intervals

and splitting point, CMP-S makes further restrictions on the alive intervals:

- Attribute a is selected for the next split if $gini_a^{est} \le gini_k^{est}$, for all attributes k. Alive intervals on attributes other than a are pruned;
- At most N alive intervals – those intervals whose $gini^{est}$ are among the lowest N of all the intervals on the spitting attribute a – are selected, and the rest are pruned.

Table 1 shows that these two restrictions do not reduce the accuracy of classification, provided that the number of total intervals is larger than 15 – in practice, we always use more than 100 intervals for large datasets. Also observe that with larger datasets the number of alive intervals tend to shrink to one. This increases the confidence in the validity of our approach.

The number of total intervals plays an important role in the accuracy of the classification. When the number of intervals is as low as 10, the CMP algorithm fails to split on the "best" attribute for dataset **Letter** and **Segment** and results in larger $gini$ values. Otherwise, CMP always finds the same split point as any exact algorithm. In most cases, limiting N, the number of alive intervals to at most 2, is enough.[1]

From approximate split to exact split

CMP does not sort, copy, or modify the training set. All operations on the training set are read-only. To keep track of the node where a record belongs to, CMP uses a one-dimensional array (nid) to record the node id for each

[1] In fact, we proved that the maximum decrease of $gini$ index in any interval is less than $2N_i/N$, where N_i is the number of records in the interval, and N is the total number of records. Thus, when the number of intervals are large enough, this difference becomes negligible.

Dataset	No. of records	Exact Algo.		The CMP Algo.			
		Attr[1]	*gini*	Intervals	Alive[2]	Attr	*gini*[3]
Letter	15,000	11	0.940323	**10**	2	**7**	**0.941751**
				15	2	-[4]	-[5]
Satimage	4,435	17	0.653167	10	1	-	-
				15	1	-	-
Segment	2,310	10	0.714286	**10**	1	**2**	**0.722580**
				15	1	-	-
Shuttle	43,500	1	0.175777	10	1	-	-
				15	1	-	-
Function 2	1,000K	1	0.401777	50	1	-	-
				100	1	-	-
Function 7	1,000K	1	0.140844	50	1	-	-
				100	1	-	-

Table 1. Splits obtained for different datasets by the SPRINT algorithm and the CMP algorithm

1. Splitting attribute of the exact algorithms.
2. Number of alive intervals of the splitting attribute selected by CMP-S, limited to at most 2.
3. $Gini$ index evaluated on records in alive intervals at next round.
4. '-' denotes that splitting attribute selected by CMP-S is the same as in the exact algorithm.
5. '-' denotes that $Gini$ index computed by CMP-S is the same as in the exact algorithm.

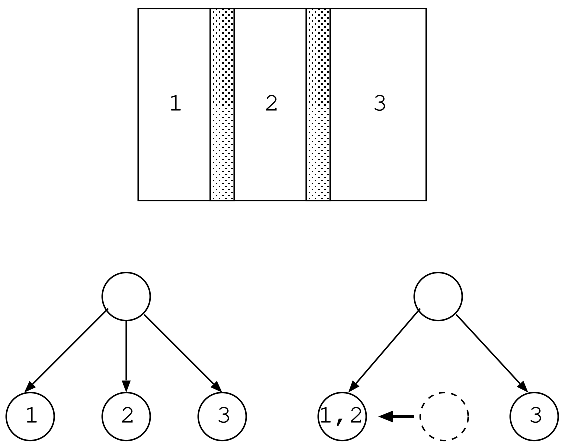


Figure 3. Reading Records in Alive Intervals is Incorporated into Next Round

record. That is, $nid[i] = j$ means that record i belongs to node j. In our implementation, nid is swapped from and to disk. For smaller data sets this array could be kept in main memory. Our tests indicate that the resulting improvement in performance would be modest.

Initially, there's only one node, which is the root, and all the records belong to it. Hence every entry in the nid array is initialized to node id 0. When node i is partitioned into node j and node k, all the records with node id i will be replaced by either j or k. To do this, the attributes of each record are tested against the splitting criteria. At this

```
For each node n which can be split further do
    If n is the root node Then
        read every record from dataset and update its histogram in the root node;
    Else
        For each record r in database do
            If (r belongs to an alive interval of last split)
                Then save r to buffer;
                Else update histogram for record r
            End If
        End For
        sort records in the buffer to derive a best split point (for the last split);
        merge subnodes if necessary;
        update histogram for records in the buffer
    End If
    For each node do
        compute gini_a^min for each attribute a;
        compute gini_a^est for each interval of each attribute a;
        split node into 2 or 3 subnodes depending on the number of alive intervals;
    End For
    Call PUBLIC(1) for pruning.
End For
```

Figure 4. A high-level description of the single-variable CMP-S algorithm

point, though, the actual partition point is only estimated: the exact partition will be decided during the next iteration, when the data is scanned again to reconstruct the matrixes. During this operation, the records in the (few) alive intervals are also set aside in buffers, where they are then sorted to determine the exact split point (i.e., the point where the gini index is minimum).

The exact splitting point for classification at the current level is unknown until the dataset is read again for classification at the next level. Therefore, at Step 4 we perform a preliminary split into $A + 1$ nodes where A is the number of alive intervals. Suppose there are two alive intervals which divide the dataset into three parts, as shown in Figure 3, we create (temporarily) three subnodes for the current node. Records in area 1,2 and 3 are read to construct the histogram matrix for the left-most, middle, right-most subnodes respectively. Records in alive intervals (shaded area) are read into a buffer, and are not used to construct the histogram for the moment. After all the dataset is scanned, we compute the $gini$ index by analyzing the records in the buffer and find the best split. Assuming the best split falls into the alive interval on the right, then the histogram matrix of the subnode in the middle will be merged into the matrix of the left-most subnode, as shown in Figure 3. Finally we construct the histogram for records in the buffer according to their class label and the final split point. After that, classification for the next level will start. Thus the process of deciding the split point will not require an extra pass of the dataset.

A high level description of the CMP-S algorithm

An overall description of the CMP-S algorithm is given in Figure 4. We can think of it as having 3 parts. Part I (from line 02 to 14) reads the dataset and constructs the histogram. If there are previous splits, records being read are partitioned into the subnodes created by the previous split. Records in alive intervals are saved into a buffer since the final assignment of these records to subnodes is still pending. When all the records are read, an accurate splitting point can be derived by sorting the records in the buffer and evaluating the $gini$ index on the sorted list. Only then, records in the buffer are partitioned into their corresponding nodes.

Part II (from line 15 to 19) evaluates $gini$ index, selects the splitting attribute and determines the alive intervals. Splitting in this part is only preliminary, the exact splitting point is not decided until Part I is executed in the next round.

Part III (line 20) uses the PUBLIC algorithm to prune the decision tree. Since time spent on pruning is insignificant compared to the rest of the algorithm, it is not discussed in detail in this paper.

2.2. The CMP-B improvement

Histogram matrix

Our previous CMP-S algorithm builds a class histogram for single variables, just like the CLOUDS algorithm; but CMP-B combines class histograms of two different variables into a matrix. That is to say, a cell $M_{a_i b_j}$ in the matrix M is the class histogram for records whose attribute **a** values are in the i^{th} interval and whose attribute **b** values are in the j^{th} interval. Figure 5 shows an example which has two classes.

The same attribute is used for all the X-axes of those matrices in a node. For example, in Figure 5, all the matrices have $salary$ as their X-axes. Thus, each node of the decision tree built by bivariate CMP contains $N - 1$ such matrices, where N is the total number of attributes (excluding the class label) of a dataset.

For the root node of the decision tree, the attribute on the X-axes in its histogram matrix is selected randomly. For nodes other than the root node, the attribute on the X-axes is decided by prediction (Section 2.2.).

$Gini$ index

The single-variable CMP-S algorithm computes $gini$ index at interval boundaries for each attribute using Equation 2.

Bivariate CMP-B computes the $gini$ index on two-dimensional histogram matrices. Given a matrix formed by two attributes a and b, the $gini$ index of attribute a is computed by summing up the histogram in all the intervals on attribute b and the $gini$ index of attribute b by summing up the histogram in all the intervals on attribute a. Since all the matrices in a node have the same X-axis, only its first matrix needs to compute the $gini$ index along the X-axis.

Predicting next splitting attribute

We keep class histograms in matrices formed by two different attributes; thus, if the splitting-attribute for a node is on the X-axes of its matrices, we can use the same matrices for another split without having to scan the dataset.

For instance, in Figure 6, the first split (vertical split) takes place on the $salary$ attribute, which is on the X-axis for each matrix. This partitions the dataset into two subnodes. There is no need to scan the dataset again to construct the matrices for the two subnodes: the matrix for the right subnode is the shaded sub-matrix of the original matrix, and the matrix for the left node is the unshaded submatrix. Thus we already have the information to perform another split. In Figure 6, subsequent splits on age attribute and $loan$ attribute partition the node into 4 subnodes. Thus we are able to grow the decision tree more than one level for each scan of the dataset.

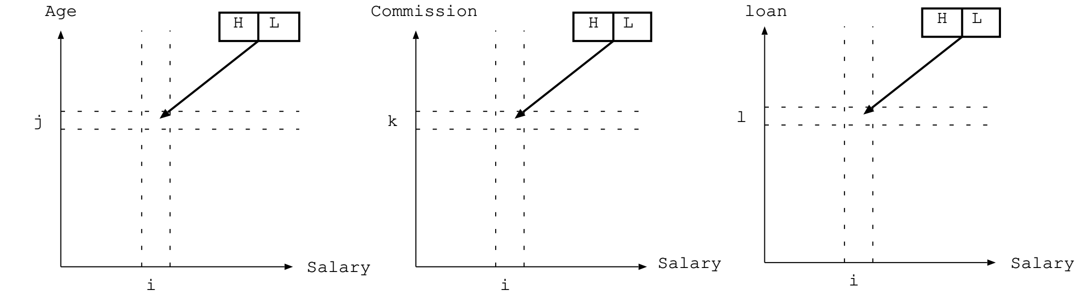


Figure 5. Histogram Matrix for a Dataset with 4 Attributes

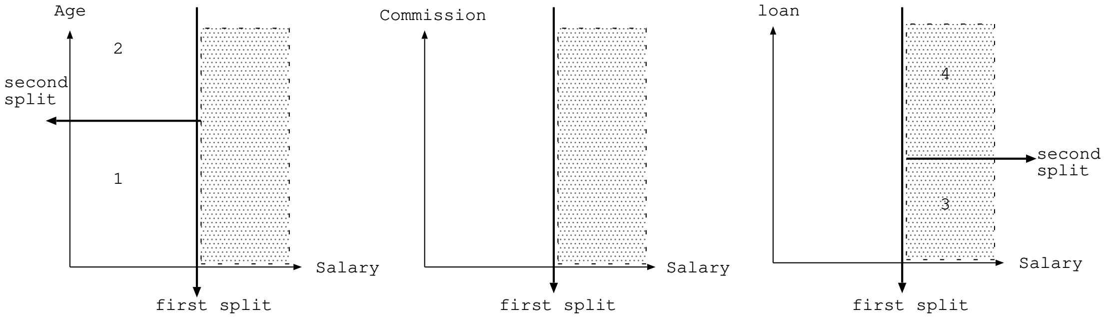


Figure 6. Splitting Matrices Twice into 4 Sub-nodes

For each attribute i **do**
    If (i is the X-axis or Y-axis of submatrix) **then**
        $gini =$ gini index on the submatrix along X or Y-axis
    Else
        $gini = gini$ Index on attribute i of the parent node
    End If
End For
Return the attribute having the minimal $gini$ Index

Figure 7. predictSplit(node,submatrix)

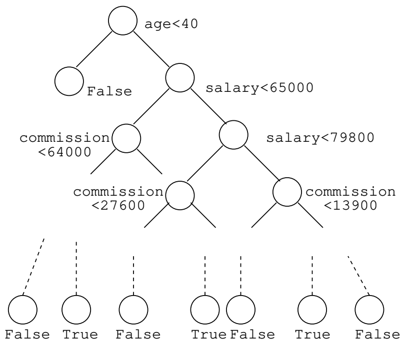


Figure 9. Decision Tree of Sprint/RainForest

However, if the first splitting attribute is on any of the Y-axis, we won't be able to split the subnodes again without a data scan since we do not have the histogram of attributes on the rest Y-axis for the subnodes. To maximize the probability that the next split will occur on the X-axes, CMP-B uses function **predictSplit** shown in Figure 7 to predict the attribute, a, on which the next split is most likely to happen for a certain node. Histogram matrices will be constructed with a as their X-axis. If the prediction is correct, the next splitting attribute will be on the X-axis and further splitting is made possible.

Function **predictSplit(node,submatrix)** computes the exact value of the $gini$ index on the X-axis and the Y-axis for the sub-matrices produced by the split. For the remaining attributes, we estimate the $gini$ index using the current matrices (before the split). While this a crude estimate, it appears effective in most cases. For instance, for the large dataset on 9-attributes used for Function 2, test results show about 80% of the predictions are accurate. However, it's likely that the accuracy of our prediction might decrease when the total number of attributes is very large.

If there is only one alive interval after a split – which is often the case for large datasets–the accuracy remedy mentioned in the previous section is still effective for the bivariate CMP with prediction. This is shown in Figure 8. Records in shaded area will be read into buffer and exact split points for node A, B and C will be computed when the

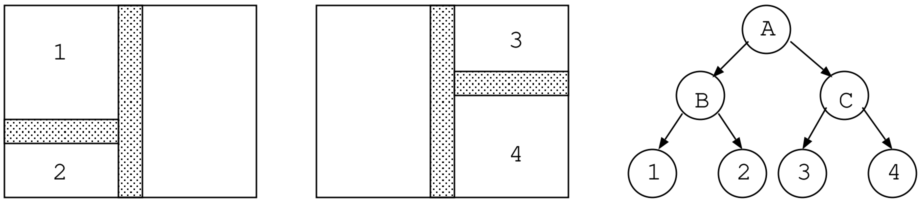


Figure 8. Remedy Accuracy Loss for Multiple Splits

```
[01] For each node n which can be split further do
[02]     If n is the root node Then
[03]         read every record from dataset and update histogram matrix in root node;
[04]     Else
[05]         For each record r in database do
[06]             If (r belongs to an alive interval of last split)
[07]                 Then save r to buffer;
[08]                 Else update histogram for record r
[09]             End If
[10]         End For
[11]         sort records in the buffer to derive a best split point(for the last split);
[12]         merge subnodes if necessary;
[13]         update histogram matrix for records in the buffer
[14]     End If
[15]     For each node do
[16]         compute gini_min and gini^est along X and Y-axis for each histogram matrix;
[17]         split node into 2 or 3 subnodes depending on the number of alive intervals;
[18]         if split is on X-axis and there is only 1 alive interval, split the subnodes again;
[19]         Call predictSplit to predict next splitting attribute for each subnode
[20]     End For
[21]     Call PUBLIC(1) for pruning.
[22] End For
```

Figure 10. A high-level description of the CMP-B algorithm

dataset is read again in the next round.

A high level description of the CMP-B algorithm

CMP-B is an extension of the CMP-S algorithm. Most part of the algorithm described in Figure 10 remains the same as that of Figure 4.

The only difference of Part I (line 02 to 14) is that histogram in CMP-B is a two-dimensional matrix, all the operation on the histogram, including updating the counts, computing the gini index, are performed on the two-dimensional matrix.

Part II (line 15 to 20) evaluates $gini$ index and select the splitting attribute and the alive intervals. It also tries to split the matrix again should the first split take place on the X-axis of the matrix. When splitting for the current node finishes, we predict the next splitting attribute for each subnode and make that attribute as the X-axis for the attribute matrices in those subnodes.

2.3. The CMP algorithm

Almost all the current classification algorithms build univariate decision trees. When attributes in the underlying dataset are correlated, the limitations of these algorithms can be seen in two forms: (i) subtrees are replicated in the decision tree, and (ii) features are tested more than once along a path in the decision tree[8].

For example, in the following synthetic dataset, class label (`True` or `Flase`) is decided by the truth value of Function f:

$$f : (age \geq 40) \wedge (salary + commission \geq 100,000)$$

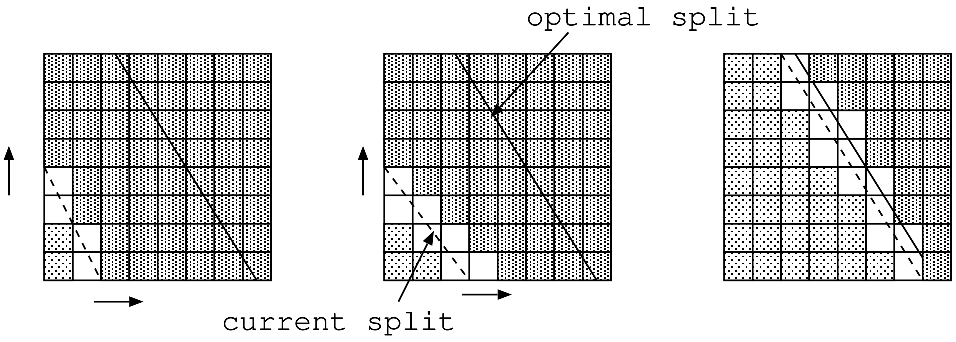


Figure 11. Choosing a Best Split Position on Linear Combination of 2 Variables

Figure 9 shows the decision tree built by algorithms such as Sprint or RainForest[16] for Function f. The decision tree tends to be very big and is hardly comprehensible. On the contrary, an optimal decision tree (Figure 1(b)) will have only 2 levels and very easy to understand by humans.

Deriving splitting line

CMP takes advantage of the two-dimensional histogram matrices maintained by CMP-B to explore linear relationships between the X-axis and Y-axis for the underlying dataset. A `splitting line` ($ax + by = c$, the dashed line in Figure 11) will partition a matrix M of a dataset S into 3 subsets, S_u, S_a and S_o:

- S_u: Records under the line (light gray cells in Figure 11). These records satisfy $ax + by < c$.
- S_a: Records above the line (dark gray cells). These records satisfy $ax + by > c$.
- S_o: Records on the line (white cells).

The $gini$ index for the partitioned data $gini^D(S, line)$ can be obtained by:

$$gini^D(S, line) = \frac{N_u}{N} gini(S_u) + \frac{N_a}{N} gini(S_a) + \frac{N_o}{N} gini(S_o)$$

where N_u, N_a and N_o are the number of records of S_u, S_a and S_o, respectively, and N is the total number of records. The algorithm to compute the best split line for a given matrix M is described in Figure 12. Procedure $giniNegativeSlope(M)$ starts from bottom left corner and chooses to increase either the x intercept or the y intercept depending on the $gini$ index to derive a best splitting line with positive slope. The procedure to derive positive slope splitting line, $giniPositiveSlope(M)$, is similar to $giniNegativeSlope(M)$ except that we start from the top left corner.

---

[01] $x = 1;\ y = 1;$
[02] $line \Leftarrow$ line from $(x, 0)$ to $(0, y)$;
[03] $gini = gini^D(S, line)$;
[04] While (line partitions the matrix into 3 parts) **do**
[05] $linex \Leftarrow$ line from $(x + 1, 0)$ to $(0, y)$;
[06] $liney \Leftarrow$ line from $(x, 0)$ to $(0, y + 1)$;
[07] **If** $(gini^D(S, linex) < gini^D(S, liney))$ **Then**
[08] $x = x + 1;$
[09] $gini = gini^D(S, linex)$;
[10] **Else**
[11] $y = y + 1;$
[12] $gini = gini^D(S, liney)$;
[13] **End If**
[14] $line \Leftarrow$ line from $(x, 0)$ to $(0, y)$;
[15] End While
[16] Return $gini$

Figure 12. `giniNegativeSlope(M)` **deriving splitting line with negative slope for matrix M**

Using the CMP algorithm on the previous dataset derives a decision tree (Figure 13) much simpler. Although it is still not optimal (splitting at line $salary + 0.93 \times commision > 95796$ as opposed to $salary + commision > 100000$), it provides useful insight to help understand the underlying dataset.

Heuristics

We will not try to partition datasets on linear combination of 2 variables if the minimal $gini$ index obtained in Line 16 of Figure 10, $gini_a^{est}$, is already lower than a certain threshold. However, if it is not the case, we calculate a new $gini$ index for each matrix M as follows:

$$gini = min\{gini_a^{est}, giniNegativeSlope(M), giniPositiveSlope(M)\}$$

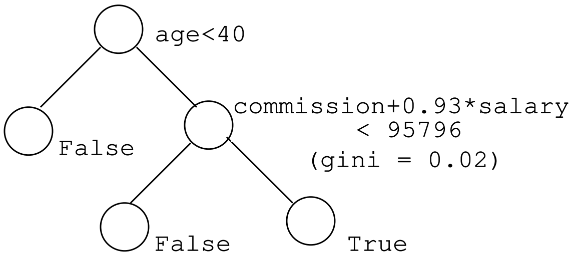


Figure 13. Decision Tree Built by CMP

Then we take the minimum $gini$ index over all the matrices and if it is significantly smaller than $gini_a^{est}$ (say 20% smaller), then we split on linear combination of 2 variables.

Limitations of the CMP algorithm

CMP can not detect all the linear relationships exist in a dataset for the following reasons: (i) For datasets with N attributes, there are only $N-1$ matrices (all matrices' X-axis represent a same attribute, which is predicted to have low $gini$ index) at each node, while detecting linear relationships between any 2 attributes require $N(N-1)/2$ matrices. (ii) The more attributes a dataset has, the more difficult to detect linear relationships between 2 attributes. This is because the value distribution of other attributes will make the linear relationships less obvious.

3. Performance results

In this section, we study the performance of the CMP family classifiers and compare their scalability with several state-of-the-art classification algorithms, including SPRINT, RainForest[16], and CLOUDS. Our experiments are carried out on a Ultra SPARC 10 running Solaris 2.6 with 128M memory. The SPRINT and CLOUDS algorithms were described in Section 1.

First, we examine the performance of CMP-S, CMP-B and CMP as the size of the input database increases. Our experiments divide an attribute domain into 100 to 120 intervals. Figure 14 and 15 shows the overall running time of the algorithms as the number of records in the training set increases from 200,000 to 2,500,000. Function 7 generates a much larger decision tree and thus the construction takes much longer than for Function 2. Both figures show that the running time grows nearly linearly with the number of records. CMP-B is almost 40% faster than CMP-S thanks to the prediction. The running time of CMP is slightly longer than that of CMP-B.

Next in Figure 16 and Figure 17, we compare the performance of CMP with SPRINT, RainForest and CLOUDS. For algorithm RainForest, we use its RF-Hybrid version[16] with fixed size AVC-group buffer of 2.5 million entries. For both Function 2 and Function 7, RainForest algorithm slightly outperforms CMP. However, as we shall see later, the speed-up of RainForest mainly comes from the usage of main memory. In comparison with SPRINT, CMP is nearly five times faster.

When the underlying dataset is linearly correlated and this correlation is detected by CMP algorithm, CMP shows significant performance advantage over RainForest and other classifiers. Figure 18 summarizes our experiment on synthetic dataset generated by Function f introduced in Section 2.3.. Since the decision tree created by CMP has only 2 levels, which means it only takes two passes over the dataset to complete classification, it is much faster than RainForest which tends to build very large decision trees.

Finally, we investigate memory space requirements for CMP family classifiers, RainForest algorithm and SPRINT algorithm. This is shown in Figure 19. Since the RF-Hybrid version of RainForest uses a fixed size AVC-group buffer with 2.5 million entries and the dataset has 2 different class labels, the AVC-group will take $2.5 * sizeof(int) * 2 = 20\ MB$ memory space. The memory space requirement for CMP, which consists of the alive interval buffer, the rid buffer and the histogram matrix, is considerably smaller. The memory space does not increase linearly for either CMP or SPRINT because disk swap is used.

4. Conclusions

The CMP classifier was designed to limit disk I/O, a critical issue on large data sets, while preserving good accuracy. These initial objectives were met as follows:

- CMP uses discretization techniques to reduce computation of $gini$ index. It also avoids materializing sorted attribute lists and thus reduce disk I/O;
- CMP remedies loss of accuracy caused by discretization by using a simplified version of the estimation method proposed in CLOUDS. However, unlike CLOUDS, it eliminates the need to perform extra passes over the dataset to determine the exact split;
- CMP allows decision trees to grow more than one level after each scan of the dataset through the prediction technique;
- CMP uses linear combination of attributes as splitting criterion to produce better models than those obtainable with SPRINT or CLOUDS.

Exact classification algorithms produce accurate univariate decision trees. Approximate methods, for example, methods using discretization and sampling,

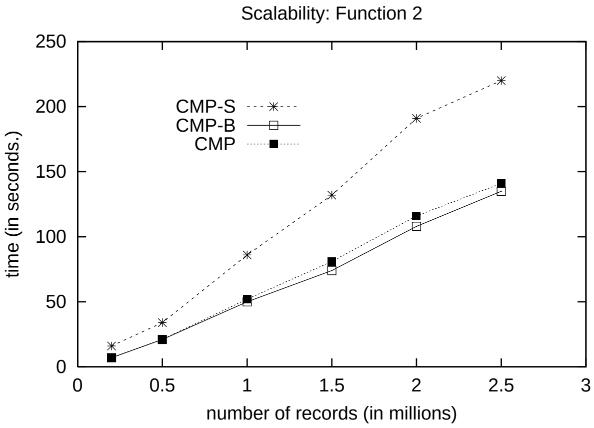


Figure 14. Scalability : Function 2

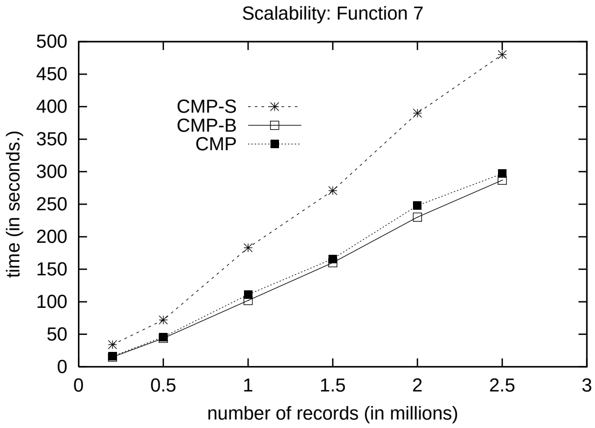


Figure 15. Scalability : Function 7

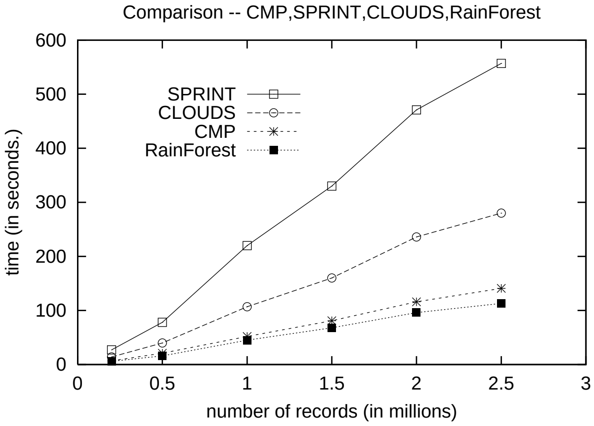


Figure 16. Comparison : Function 2

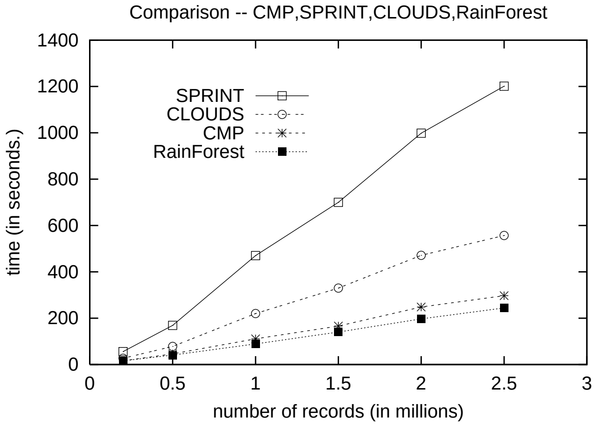


Figure 17. Comparison : Function 7

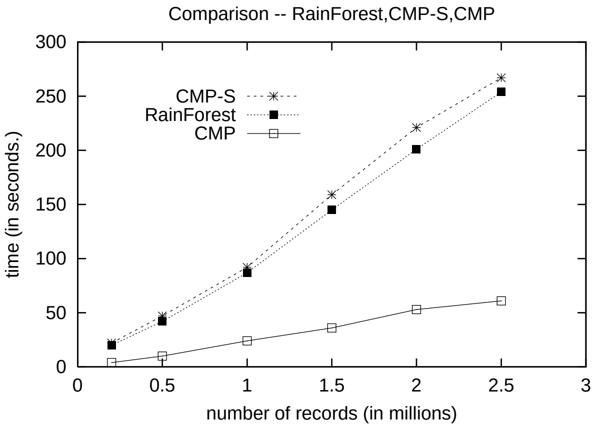


Figure 18. Comparison: Function f

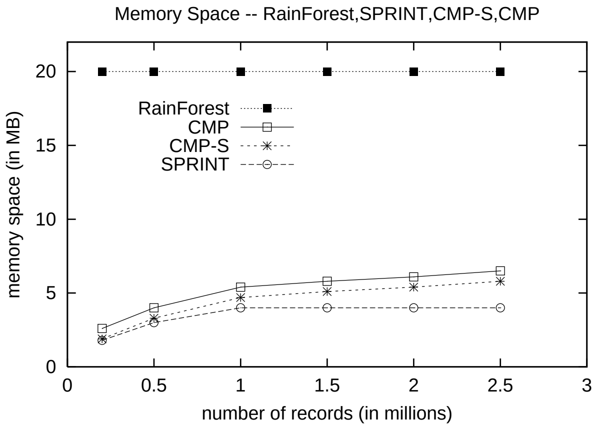


Figure 19. Comparison: Memory Space Usage

reduce heavy computation and disk I/O at the cost of accuracy. CMP brings the strengths of several approaches together, and for large datasets, it is as accurate as SPRINT, but significantly faster. CMP is also almost as fast as RainForest, but uses much less memory and it is therefore expected to scale-up better.

Moreover, by using a linear combination of two variables as splitting criteria, CMP can uncover complex relationships unknown to previous algorithms.

References

[1] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. "Classification and Regression Trees." Wadsworth, Belmont, 1984.

[2] J. Ross Quinlan. "C4.5: Programs for Machine Learning." Morgan Kaufmann, 1993.

[3] M. James. "Classification Algorithms." Wiley, 1985.

[4] D. E. Goldberg. "Genetic Algorithms in Search, Optimization and Machine Learning." Morgan Kaufmann, 1989.

[5] R. Agrawal, T. Imielinski, and A. Swami. "Database Mining: A Performance Perspective." IEEE Trans. on Knowledge and Data Engineering, 5(6), Dec. 1993.

[6] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. "Machine Learning, Neural and Statistical Classification." Ellis Horwood, 1994

[7] M. Mehta, R. Agrawal, and J. Rissanen. "SLIQ: A Fast Scalable Classifier for Data Mining." Proc. of Fifth Int'l Conference on Extending Database Technology, pages 18-32, March 1996.

[8] C. E. Broadley, P. E. Utgoff. "Multivariate Decision Trees." in Machine Learning, 19, 45-77 (1995).

[9] Rajeev Rastogi, Kyuseok Shim. "PUBLIC: A Decision Tree Classifier that Integrates Building and Pruning." VLDB 1998: 404-415

[10] M. Mehta, J. Rissanen, and R. Agrawal. "MDL-Based Decision Tree Pruning." Int'l Conf. on Knowledge Discovery in Databases and Data Mining, pages 216-221, August 1995.

[11] S. M. Weiss, C. A. Kulikowski. "Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems." Morgan Kaufmann, 1991

[12] Jason Catlett. "Megainduction: Machine Learning on Very Large Databases." PhD thesis, University of Sydney, 1991.

[13] Philip K. Chan, Salvatore J. Stolfo. "Experiements on multistrategy learning by metalearning." In Proc. Second Int'l Conference on Info. and Knowledge Mgmt. , pages 314-323, 1993.

[14] Khaled Alsabti, Sanjay Ranka, Vineet Singh. "CLOUDS: A Decision Tree Classifier for Large Datasets." KDD 1998: 2-8

[15] P. M. Murphy, D. W. Aha. UCI Repository of Machine Learning Databases. "http://www.ics.uci.edu/~mlearn/MLRepository.html" Dept. of Information and Computer Science, University of California, Irvine

[16] Johannes Gehrke, Raghu Ramakrishnan, Venkatesh Ganti. "RainForest: A Framework for Fast Decision Tree Constructionof Large Datasets." in Proceedings of the 24th VLDB Conference, New York, USA, VLDB 1998.

[17] J. C. Shafer, R. Agrawal, and M. Mehta. "SPRINT: A Scalable Parallel Classifier for Data Mining." in Proceedings of the 22nd VLDB Conference, Sept. 1996.