WHEN AND WHERE TO REBUILD A DECISION TREE WITH CONSTANT SPACE/TIME OVERHEAD

Tara Mirmira and Ramesh Subramonian

Abstract

I Incremental/Decremental on Decision Trees

1 Introduction

TO BE COMPLETED

1.1 Motivation

TO BE COMPLETED

2 Algorithm

2.1 Intuition

The key intuition behind this algorithm is that the cost of reordering is significantly less than the cost of the computation of the metric (e.g., gini) used to determine the best split. This allows us to store just one additional integer for each data value and yet be able to efficiently determine whether and where retraining is required. For a data set with m features and n instances, we need $m \times n$ additional integers. In contrast, **TO BE COMPLETED** [**?**]

2.2 Assumptions

Our current implementation makes the following assumptions. These are purely for the sake of convenience — these are not conceptual limitations inherent in the approach.

1. The goal attribute can be encoded as 0 or 1. We refer to the 0 value as Tails and the 1 value as Heads.
2. There are no missing values.

2.3 Definitions

1. Let n be the number of instances or data points
2. Let n^H be the number of instances with goal value = 1 (Heads)

3. Let n^T be the number of instances with goal value = 0 (Tails)
4. Clearly, $n = n^H + n^T$
5. Let m be the number of features or attributes

Definition 1 *We define $D(x, T) = \{l_0, l_1, \ldots\}$ the* **decision path** *of a data point x with respect to a tree T as the sequence of decisions made, starting from the root and ending up at a leaf.* ***TO BE COMPLETED***

Definition 2 *We define $L(x, T)$, the* **leaf node** *of a data point, x with respect to a tree T as the leaf node at which it lands up* ***TO BE COMPLETED***

Cleatly, for all points x, the first node of the decision path is the root and the last node of the path is the leaf node, $L(x, T)$

We will use the term virtual columns to denote columns that are not materialized. However, assuming that they do exist simplifies the description of the algorithm.

2.4 Data Structures

There are three principal data structures we use — X, Y, T. The example in Section 2.4.4 will hopefully make the formal description less daunting.

2.4.1 X

This represents the input data, which is stored in columnar fashion. The columns (of length n) are:

1. z, deletion flag. $X[i].z = \text{false} \Rightarrow X[i]$ has been deleted.
2. i, representing the index of the data point— values are $0, 1, \ldots n-1$. Virtual
3. $X_1, X_2, \ldots X_m$ representing the m features. So, $X_j[i]$ is the value of the j^{th} feature of the i^{th} data point.
4. g, goal attribute
5. l, the leaf node (Definition 2). **QUESTION:** *Should this be a virtual column?*

2.4.2 Y

Y consists of m data structures, $\{Y_j\}$. Y_j represents the data for feature j stored as a set of columns (of length n). These are

1. z, deletion flag, boolean, false means deleted. Virtual
2. v, feature value. Virtual
3. g, goal value. Virtual
4. l, leaf node. Virtual
5. h, cumulative number of positive instances, run time computation
6. t, cumulative number of negative instances, run time computation
7. i, index belonging to corresponding data point

The explanation of the above columns is provided below:

1. $Y_j[i].v \leq Y_j[i+1].v$
2. $Y_j.i$ is a permutation of $0, \ldots, n-1$
3. Virtual columns are created as follows, where $Y_j[k].i = k'$
   (a) $Y_j[k].v = X_j[k']$
   (b) $Y_j[k].g = X.g[k']$
   (c) $Y_j[k].z = X.z[k']$
   (d) $Y_j[k].l = X.l[k']$
   (e) $Y_j.h, Y_j.t$ are constructed to give us is the number of Heads and Tails to the left of each data point for feature j, as shown in Figure 1

2.4.3 T

This is the decision tree represented as an array, where each element has the following fields

1. z, deletion flag, boolean, false means deleted.

Let $k' = Y_j[k].i$
Let $h[k] \leftarrow 1$ if ($X.g[k'] = 1$ **and** $X.z[k'] = 1$) and 0 otherwise
Let $h[k] \leftarrow 1$ if ($X.g[k'] = 1$ **and** $X.z[k'] = 1$) and 0 otherwise
$Y_j.h[0] \leftarrow h[0]$
$k > 0 \Rightarrow Y_j.h[k] \leftarrow Y_j.h[k-1] + h[k]$
Let $t[k] \leftarrow 1$ if ($X.g[k'] = 0$ **and** $X.z[k'] = 1$) and 0 otherwise
$Y_j.t[0] \leftarrow t[0]$
$k > 0 \Rightarrow Y_j.t[k] \leftarrow Y_j.t[k-1] + t[k]$

Figure 1: Pseudo code for computing $Y_j.h, Y_j.t$

2. p, parent
3. l, left child
4. r, left child
5. n, number of data instances whose decision path includes this node. Before any data items are deleted, $Z[0].n = n$ indicating that **all** data points pass through the root.
6. d, depth. $Z[0].d = 0$ indicating that the root is at depth 0
7. m, best metric
8. f, feature with best metric
9. v, best split value of feature with best metric

2.4.4 An example

An example of the inter-relationship between X and Y is provided in Table 1.

3 Deletion

We first provide the intuition behind the algorithm. Let's describe what happens upon deletion of a data point with index i. We start by considering the impact of this deletion on node $l' = 0$, the root.

We use Y to evaluate the best metric for each feature, *after* this deletion has been factored in. This is described in Section 4.

If the new best metric improves upon $T[l'].m$, the sub-tree rooted at l' needs to be discarded and recomputed and grafted back on to the original tree. This is described in Section 6.

i	X_1	X_2	g	z	$Y_1.i$	$Y_1.v$	$Y_1.h$	$Y_1.t$	$Y_2.i$	$Y_2.v$	$Y_2.h$	$Y_2.t$
0	4	1	1	1	4	1	0	0	0	1	1	0
1	9	9	1	1	9	2	0	0	7	2	1	1
2	5	4	0	1	5	3	0	1	5	3	1	2
3	7	6	1	1	0	4	1	1	2	4	1	3
4	1	5	1	0	2	5	1	2	4	5	1	3
5	3	3	0	1	8	6	1	3	3	6	2	3
6	10	7	0	1	3	7	2	3	6	7	2	4
7	8	2	0	1	7	8	2	4	9	8	2	4
8	6	10	0	1	1	9	3	4	1	9	3	4
9	2	8	1	0	6	10	3	5	8	10	3	5

Table 1: Example of Data Structures

function $\mathrm{BestGini}(h, t)$
    $n_L^H = h[0], n_L^T = t[0]$
    $m = \mathrm{Gini}(n_L^H, n_L^T, n^H, n^T)$
    for $i = 1$ **to** $n - 1$ **do**
        $n_L^H = n_L^H + h[i], n_L^T = n_L^T + h[i]$
        $m = \min(m, \mathrm{Gini}(n_L^H, n_L^T, n^H, n^T))$
    endfor
    return m
end

Figure 2: Pseudo code for metric computation

If none of the features realize a better metric, then we determine whether this data point would have next visited the left child or the right child of l'.

need to recursively perform the same test at node l_{i+1}. In order to make the recursion possible, we need to create Y' from Y. This is done by $Y' \leftarrow Create(Y, l_{i+1}$, described in Section 5.

The recursion stops when a sub-tree needs to be recomputed or we arrive at a leaf.

4 Metric Computation

Let $h = Y_j.h, t = Y_j.t$ for some feature j. Figure 2 shows how the gini metric is computed for each possible split point.

Gini computation is in Figure 3

```
function Gini(n_L^H, n_L^T, n^H, n^T))
    TO BE COMPLETED
    return x
end
```

Figure 3: Pseudo code for gini computation

```
function CreateY(l, Y)
    Let n_Y be size of Y
    Allocate space for Y'. Space needed = n'_Y = T[l].n
    for i ← 0 to n_Y − 1 do
        TO BE COMPLETED
    endfor
    return Y'
end
```

Figure 4: Pseudo code for creating Y on recursive step

5 Re-Create Y

See Figure 4.

6 Grafting

the sub-tree rooted at l needs to be discarded ($z \leftarrow \text{false}$ for all such nodes) 'and replaced by a new sub-tree built using all points with index i such that $X[i].l$ is a descendant of l'

7 Conclusion

TO BE COMPLETED