COMPUTING DECISION TREES WITH A SINGLE SORT

Tara Mirmira and Ramesh Subramonian

Abstract

A typical computational strategy for building decision trees (as evidenced in scikit-learn) is as follows. Each feature is sorted and then traversed in ascending order to determine the best split point. The best split point over all features is selcted and used to partition the data into two. This process is repeated recursively until some stopping criterion is reached e.g., the number of instances is too small. The contribution of this paper is to provide a novel indexing strategy that requires a single sort at the beginning. After that, maintaining the sorted order is accomplished by a linear scan of the data.

1 Introduction

2 Algorithm

3 Results

3.1 Assumptions

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

1. The goal attribute can be encoded as 0 or 1
2. The values of the attributes used to build the decision tree are ordered and can be represented as floating point numbers
3. $n \leq 2^{32}$, where n is the number of instances
4. The number of unique values for each instance is $< 2^{31}$
5. There are no missing values. Or, if any exist, they have been redressed by imputing values to them.

3.2 Data Structures

1. Let n be the number of instances
2. Let m be the number of features.

3. Let $X[m][n]$ be the input data. $X[j]$, also denoted as X_j, is a column vector containg the values of feature j

4. Let $g[n]$ be the values of the goal attribute

5. Let $Y[m][n]$ be the transformed data where input data has been "position-encoded" by its position in the sort order (ascending). A sample mapping from X-values to Y-values is shown below.

$$[11, 32, 47, 11, 17, 28, 32, 55] \Rightarrow [1, 4, 5, 1, 2, 4, 6]$$

   Further, for efficiency, each element of Y is encoded as a 64-bit integer where

   - bits $[0..30]$ represent the Y-value itself. We refer to this as $Y_j[i].y$
   - bit 31 represent the goal value We refer to this as $Y_j[[i].g$
   - bits $[32..63]$ represent the "from" value, explained later We refer to this as $Y_j[i].f$

   Y_j is sorted so that $Y_j[i].y \leq Y_j[i+1].y$ In order to record the original position of this value, we use the `from` field. The inter-relationship is specified as follows

   - $x = X_j[k]$
   - $k = Y_j[i].f$
   - $y = Y_j[i].y$
   - Then, y is the position-encoded value of x

6. Let $T[m][n]$ be a data structure used to record the "to" indexing. In other words, it tells us **to** which position a datum in the original set has been permuted. Its interlinking with the "from" field is best explained in Invariant 1

Invariant 1 *Let* $p = Y_j[i].f$. *Then* $T_j[p] = i$

4 Conclusion