Optimal binary decision tree

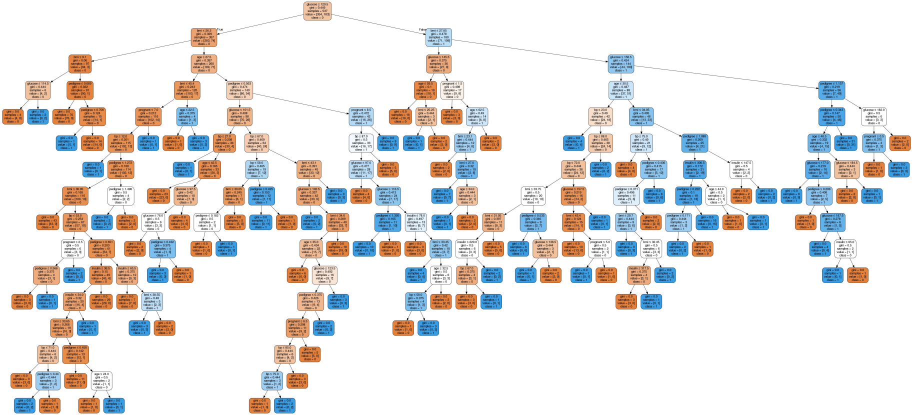
Mohammad Mahdi Heydari

Amirkabir University of Technology

June 1, 2019

Overview

Introduction



Introduction

- An optimal binary decision tree is one which minimizes the expected number of tests required to identify the unknown object.
- Large amount of effort had been put into finding efficient algorithms for constructing optimal binary decision tree.
- On supposition that $P \neq NP$, such algorithm does not exist.
- It supplies motivation for finding efficient heuristics for constructing near-optimal decision trees.

Definition

Sets

Let $X = \{x_1, ..., x_n\}$ be a finite set of objects and let $\tau = \{T_1, ..., T_t\}$ be a finite set of tests. for each test T_i, $1 \leq i \leq t$ and object x_j, $1 \leq j \leq n$, we either have $T_i(x_j) = true$ or $T_i(x_j) = false$. T_i also denotes the set $\{x \in X | T_i(x) = true\}$.

Problem

The problem is to construct an identification procedure for the objects in X such that the expected number of tests required to completely identify an element of X is minimal. At each non-terminal node, a test is specified and terminal nodes specify objects in X.

Definition

Procedure

To apply the identification procedure one first applies the test specified at the root to the unknown object; if it is false one takes the left branch, otherwise the right. This procedure is repeated at the root of each successive subtree until one reaches a terminal node which names the unknown object.

Cost

Let $p(x_i)$ be the length of the path from the root to the terminal node naming x_i, that is, the number of tests required to identify x_i. Then the cost of this tree will be:

$$\sum_{x_i \in X} p(x_i)$$

Optimization vs. Decision

Decision Problem

Decision

The decision tree problem $DT(\tau, X, w)$ is to determine whether there exists a decision tree with cost less than or equal to w given τ and X.

Theorem

$DT(\tau, X, w)$ is NP-complete.

Proof.

$DT \in NP$. since a a non-deterministic Turing machine can guess the decision tree and then see if its weight is less than or equal to w. $\square$

Decision Problem

Proof.

To show that DT is NP-complete, we show that $EC3\alpha DT$, where EC3 is the problem of finding an exact cover for a set X, and where each of the subsets available for use contains exactly 3 elements. More precisely, we are given a set $X = x_1, ..., x_n$ and a family $\tau = T_1, ..., T_t$ of subsets of X, such that $|T_i| = 3$ for $1 \leq i \leq t$, and we wish to find a subset S of τ such that $\bigcup_{T_i \in S} T_i = X$ and $((T_i, T_j \subseteq S \text{ and } i \neq j) \Rightarrow T_i \cap T_j = \emptyset)$. The exact cover problem EC (where there is no restriction on the size of each T_i) is known to be NP-complete. To show that EC3 is NP-complete, we show $3DM\alpha EC3$, where 3DM is the problem of finding a "three-dimensional matching". $\square$

Given a set of binary m-bit strings S, choosing some bit i always partitions the items into two sets S_0 and S_1 where S_0 contains those items with bit i $= 0$ and S_1 contains those items with i $= 1$. A greedy strategy for splitting a set S chooses the bit i which minimizes the difference between the size of S_0 and S_1. In other words, it chooses the bit which most evenly partitions the set. Using this strategy, consider the following greedy algorithm for constructing decision trees of the DT type given a set of n items X:

```
GREEDY-DT(X)
if X = ∅
    then return NIL
    else Let i be the bit which most evenly partitions X into X^0 and X^1
         Let T be a tree node with left child left[T] and right child right[T]
         left[T] ← GREEDY-DT(X^0)
         right[T] ← GREEDY-DT(X^1)
         return T
```

A straightforward implementation on this algorithm runs in time $O(mn^2)$. While the algorithm does not always give an optimal solution, it does approximate it within a factor of $ln(n) + 1$.

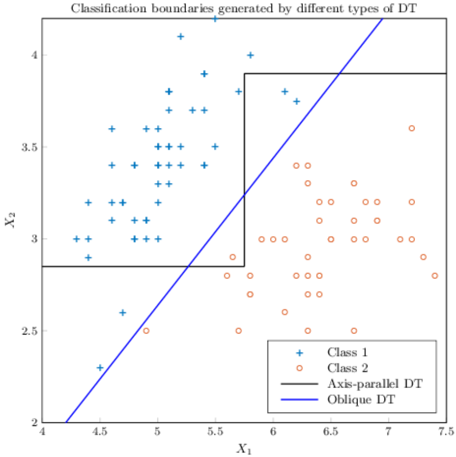
Application

In some applications we have X set as tuples with some attributes and we are free to choose τ set. The leaves should only classify x_js (vs. identification). There are some greedy algorithms to construct these decision trees. Our goal will be to find a split leading to the most monotonous subtrees.

Create Test Set

Split Approaches:

- Oblique
- Axis-Parallel ✓



Split Measures

- Information Gain

$$Info(D) = -\sum_{i=1}^{m} p_i \; log_2(p_i)$$

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

$$Gain(A) = Info(D) - Info_A(D)$$

- Gini (only binary)

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2$$

$$Gini_A(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2)$$

$$\Delta Gini(A) = Gini(D) - Gini_A(D)$$

References

- Hyafil, Laurent & L. Rivest, Ronald. (1976). Constructing Optimal Binary Decision Trees is NP-Complete. Inf. Process. Lett.. 5. 15-17. 10.1016/0020-0190(76)90095-8.
- Adler, Micah & Heeringa, Brent. (2008). Approximating Optimal Binary Decision Trees. 10.1007/978-3-540-85363-3_1.
- Data Mining: Concepts and Techniques, Second Edition Jiawei Han and Micheline Kamber (2006)

Any Question?