How do decision tree classifiers work?

Mason Biamonte
github.com/linguaquanta
July 17, 2022

Supervised binary classification arises when one is given feature data with binary labels and asked to use the correlations between the feature values and the labels to predict labels for yet unseen feature data. One of the major mathematical models behind supervised binary classification is the decision tree classifier. In these notes, we take a dive into the mathematical details to understand how decision trees allow a machine to learn the correlations necessary for classification. We will first define a decision tree classifier mathematically and then show how it is used to perform binary classification. Evaluating the classification performance will show that decision tree classifiers, by themselves, are weak learners. However, when we have many different trees and combine the classification predictions from each in special ways, the classification performance increases dramatically. There are two main such ways we will be combining the predictions from many decision trees: bagging and boosting. Bagging is used to increase bias performance and boosting is used to decrease variance.

1. Definition and Construction of Decision Tree Classifiers

Suppose we have N labeled features vectors in our data

$$\mathcal{D} = \{(\boldsymbol{x}_i, y_i) : 1 \leq i \leq N, \boldsymbol{x}_i \in \{0,1\}^M, y_i \in \{0,1\}\} \tag{1.1}$$

where all features are categorical and one-hot encoded into bit strings of length M. A **binary classifier** is a boolean function $f : \{0,1\}^M \to \{0,1\}$ splitting bit strings of length M into two subsets having labels 0 and 1. It turns out that decision trees, of appropriate depth, can approximate any boolean function. This fact motivates using decision trees for classification. Classifiers based on decision trees are built from discrete finite automata $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ whose state spaces Q are the vertex sets of a binary directed rooted labeled out-trees $\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T})$ of fixed depth $d_\mathcal{T}$, where Σ is the alphabet of labels, δ is the transition function, q_0 is the start state and F is the set of final states. The tree's edges are labeled with $\{0,1\}$ according to the rule that $\ell_E(e) = 0$ if when v_{j_1} and v_{j_2} are the two child nodes of v_j and $j_1 < j_2$, the edge $e = (v_j, v_{j_1})$ and $\ell_E(e) = 1$ otherwise. The leaf nodes $V_\mathcal{T}^L \subset V_\mathcal{T}$ are labeled by a function $\ell_V : V_\mathcal{T}^L \to \{0,1\}$ chosen to match the maximum number of labels $\{y_i\}$ via a procedure to be described shortly. The tree $\mathcal{T}$ serves as a discrete finite automaton because the root vertex v_0 is the start state, child vertices are visited according to the transition rule defined by the edge labeling.

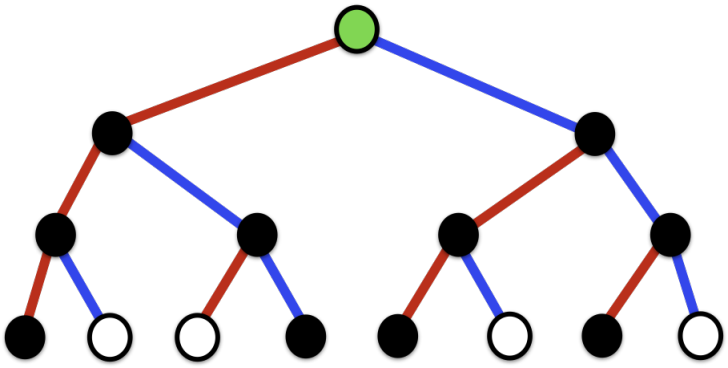

FIGURE 1. Example tree $\mathcal{T}$ with depth $d_{\mathcal{T}} = 3$. The root vertex is colored green and the coloring of the edges denotes the labeling with red signifying 0 and blue 1. The leaf vertices are colored black or white if labeled 0 or 1, respectively.

That is, given an input feature string $\boldsymbol{x} = x_1 x_2 \cdots x_M$ with first bit x_1, the transition rule specifies moving from v_0 to the vertex $v = \delta(v_0)$ such that the label ℓ_E of the edge $e = (v_0, v)$ equals x_1. The final states of the discrete finite automaton are the leaf nodes and the labels on these nodes tell us whether to accept or reject the binary string $\boldsymbol{x}$. This motivates the decision tree classifier.

Definition #1. A **(binary) decision tree classifier of depth d** is a function DT_d generated by: (1) a discrete finite automaton $\mathcal{A}(\mathcal{T})$ with state space given by a rooted directed binary labeled out-tree $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}})$ and (2) an element of the permutation group on M elements π which reorders input feature strings $\boldsymbol{x}$. In particular, $\mathcal{A}(\mathcal{T}) = (Q, \Sigma, \delta, q_0, F) = (V_{\mathcal{T}}, \{0,1\}, \delta(E_{\mathcal{T}}), v_0, V_{\mathcal{T}}^L)$.

The ordering π specifies the priority with which we are testing the feature values. We are prioritizing which features to probe first and use to build the classification function. Since decision tree classification is based on heuristics which determine the best permutation π given the data $\mathcal{D}$, we can write $\pi_{\mathcal{D}}$. Let $v(\boldsymbol{x})$ denote the leaf vertex/node arrived at by running $\mathcal{A}(\mathcal{T})$ on an input feature string $\boldsymbol{x}$. Then, the function produced by a decision tree classifier (DTC) can be written

$$\mathsf{DT}_d(\boldsymbol{x}) = \ell_{V_{\mathcal{T}}^L}\Big[v\Big(\pi_{\mathcal{D}}(\boldsymbol{x})\Big)\Big] = \tilde{y}(\boldsymbol{x}) \tag{1.2}$$

The performance of the classifier is determined by evaluating DT_d on new feature vectors that are not necessarily any of those appearing in $\mathcal{D}$ and comparing the predicted with the actual labels.