Machine Learning — Statistical Methods for Machine Learning

Tree Predictors

Instructor: *Nicolò Cesa-Bianchi* version of March 20, 2024

As we already said, while certain types of data (like images and texts) have a natural representation as vectors $\boldsymbol{x} \in \mathbb{R}^d$, others (like medical records and other structured data) do not. For example, consider a problem of medical diagnosis, where data consist of medical records containing the following fields

$$\begin{aligned}
&\texttt{age} \in \{12, \dots, 90\} \\
&\texttt{smoker} \in \{\text{yes}, \text{no}, \text{ex}\} \\
&\texttt{weight} \in [10, 120] \\
&\texttt{sex} \in \{\text{M}, \text{F}\} \\
&\texttt{therapy} \in \{\text{antibiotics}, \text{cortisone}, \text{none}\}
\end{aligned}$$

Even if we compute rescaled numerical representations for the features (including the categorical fields `smoker` and `sex`), algorithms based on Euclidean distance like k-NN may not work well.

In order to learn data whose features vary in heterogeneous sets $\mathcal{X}_1, \dots, \mathcal{X}_d$ (i.e., sets with incomparable ranges, including ranges corresponding to categorical variables), we introduce a new family of predictors: the tree predictors.

A tree predictor has the structure of an ordered and rooted tree where each node is either a leaf (if it has zero children) or an internal node (if it has at least two children). Recall that an ordered tree is one where the children of any internal node are numbered consecutively. Hence, if the internal node v has $k \geq 2$ children, we can access the first child, the second child, and so on until the k-th child.

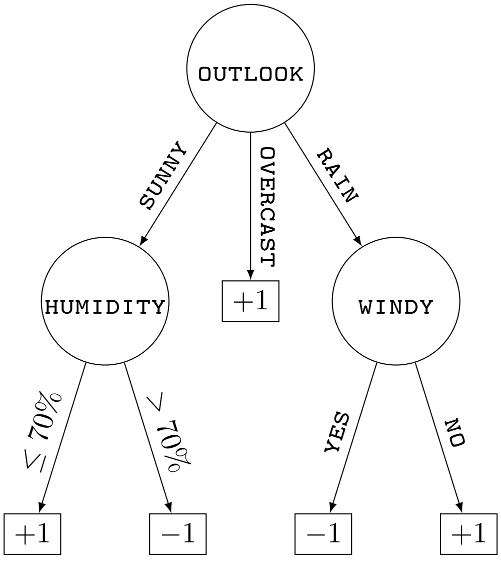

Figure 1: A classical example of a tree classifier for a binary classification task. The features are: OUTLOOK $\in$ {SUNNY, OVERCAST, RAIN}, HUMIDITY $\in [0, 100]$ and WINDY $\in$ {YES, NO}.

Fix $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_d$, where $\mathcal{X}_i$ is the range of the i-th attribute x_i. A **tree predictor** $h_T : \mathcal{X} \to \mathcal{Y}$ is a predictor defined by a tree T whose internal nodes are tagged with *tests* and whose leaves are tagged with labels in $\mathcal{Y}$. A test on attribute i for an internal node with k children is a function $f : \mathcal{X}_i \to \{1, \dots, k\}$. The function f maps each element of $\mathcal{X}_i$ to the node's children. For example, if $\mathcal{X}_i \equiv \{a, b, c, d\}$ and $k = 3$, then f could be defined by

$$f(x_i) = \begin{cases} 1 & \text{if } x_i = c, \\ 2 & \text{if } x_i = d, \\ 3 & \text{if } x_i \in \{a, b\}. \end{cases}$$

An example with $\mathcal{X}_i = \mathbb{R}$ and $k = 3$ is the following

$$f(x_i) = \begin{cases} 1 & \text{if } x_i \in (-\infty, \alpha], \\ 2 & \text{if } x_i \in (\beta, +\infty), \\ 3 & \text{if } x_i \in (\alpha, \beta] \end{cases}$$

where $\alpha < \beta$ are arbitrary values. See Figure 1 for an example.

The prediction $h_T(\boldsymbol{x})$ is computed as follows. Start by assigning $v \leftarrow r$, where r is the root of T;

1. if v is a leaf ℓ, then stop and let $h_T(\boldsymbol{x})$ be the label $y \in \mathcal{Y}$ associated with ℓ;
2. otherwise, if $f : \mathcal{X}_i \to \{1, \dots, k\}$ is the test associated with v, then assign $v \leftarrow v_j$ where $j = f(x_i)$ and v_j denotes the j-th child of v;
3. go to step 1.

If the computation of $h_T(\boldsymbol{x})$ terminates in leaf ℓ, we say that the example $\boldsymbol{x}$ is routed to ℓ. Hence $h_T(\boldsymbol{x})$ is always the label of the leaf to which $\boldsymbol{x}$ is routed.

How do we build a tree predictor given a training set S? For simplicity, we focus on the case of binary classification $\mathcal{Y} = \{-1, 1\}$ and we only consider complete binary trees, i.e., all internal nodes have exactly two children. The idea is to grow the tree classifier starting from a single-node tree (which must be a leaf) that corresponds to the classifier assigning to any data point the label that occurs most frequently in the training set. The tree is grown by picking a leaf (at the beginning there is only a leaf to pick) and replacing it with an internal node and two new leaves.

Suppose we have grown a tree T up to a certain point, and the resulting classifier is h_T. We start by computing the contributions of each leaf to the training error $\ell_S(h_T)$ (recall that each $\boldsymbol{x}$ is classified by some leaf, the leaf which $\boldsymbol{x}$ is routed to). For each leaf ℓ, define $S_\ell \equiv \{(\boldsymbol{x}_t, y_t) \in S \,:\, \boldsymbol{x}_t \text{ is routed to } \ell\}$. That is, S_ℓ is the subset of training examples that are routed to ℓ. Define further two subsets of S_ℓ, namely $S_\ell^+ \equiv \{(\boldsymbol{x}_t, y_t) \in S_\ell \,:\, y_t = +1\}$ and $S_\ell^- \equiv \{(\boldsymbol{x}_t, y_t) \in S_\ell \,:\, y_t = -1\}$.

For each leaf ℓ, let $N_\ell^+ = \left|S_\ell^+\right|$, $N_\ell^- = \left|S_\ell^-\right|$ and $N_\ell = \left|S_\ell\right| = N_\ell^- + N_\ell^+$. In order to minimize the training error $\ell_S(h_T)$, the label associated with ℓ must be

$$y_\ell = \begin{cases} +1 & \text{if } N_\ell^+ \geq N_\ell^-, \\ -1 & \text{otherwise.} \end{cases}$$

Thus, ℓ errs on exactly $\min\{N_\ell^-, N_\ell^+\}$ training examples in S_ℓ. Therefore, we can write the training error as a sum of contributions due to all leaves

$$\ell_S(h_T) = \frac{1}{m} \sum_\ell \min\left\{\frac{N_\ell^-}{N_\ell}, \frac{N_\ell^+}{N_\ell}\right\} N_\ell = \frac{1}{m} \sum_\ell \psi\left(\frac{N_\ell^+}{N_\ell}\right) N_\ell$$

where we introduced the function $\psi(a) = \min\{a, 1-a\}$ defined on $[0,1]$ —recall that $\big(N_\ell^+ + N_\ell^-\big)/N_\ell = 1$, so the argument of ψ is a number between zero and one.

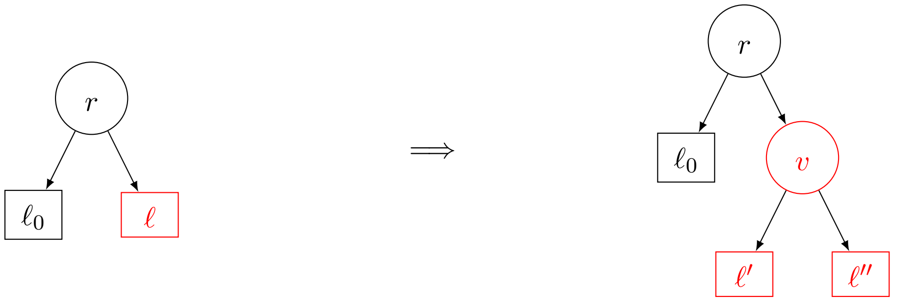

Figure 2: A step in the growth of a tree classifier: a leaf ℓ is replaced by an internal node v and be two new leaves ℓ' and ℓ''.

Suppose we replace a leaf ℓ in T with an internal node, and its associated test, and two new leaves ℓ' and ℓ'' —see Figure 2. Can the training error of the new tree be larger than the training error of T? To answer this question is sufficient to observe that ψ is a concave function (just like the logarithm). We can then apply Jensen's inequality, stating that $\psi\big(\alpha a + (1-\alpha)b\big) \ge \alpha\psi(a) + (1-\alpha)\psi(b)$, for all $a, b \in \mathbb{R}$ and all $\alpha \in [0,1]$.

Hence, via Jensen's inequality, we can study how the training error changes when ℓ is replaced by two new leaves ℓ' and ℓ'',

$$\underbrace{\psi\left(\frac{N_\ell^+}{N_\ell}\right)N_\ell}_{\text{contribution of } \ell} = \psi\left(\frac{N_{\ell'}^+}{N_{\ell'}}\frac{N_{\ell'}}{N_\ell} + \frac{N_{\ell''}^+}{N_{\ell''}}\frac{N_{\ell''}}{N_\ell}\right)N_\ell \ge \psi\left(\frac{N_{\ell'}^+}{N_{\ell'}}\right)\frac{N_{\ell'}}{N_\ell}N_\ell + \psi\left(\frac{N_{\ell''}^+}{N_{\ell''}}\right)\frac{N_{\ell''}}{N_\ell}N_\ell$$

$$= \underbrace{\psi\left(\frac{N_{\ell'}^+}{N_{\ell'}}\right)N_{\ell'}}_{\text{contribution of } \ell'} + \underbrace{\psi\left(\frac{N_{\ell''}^+}{N_{\ell''}}\right)N_{\ell''}}_{\text{contribution of } \ell''}$$

meaning that a split never increases the training error.

A leaf ℓ such that $N_\ell^+ \in \big\{0, N_\ell\big\}$ is called **pure** because it does not contribute to the training error. Note that $\ell_S(h_T) > 0$ unless all leaves are pure.

We now describe a generic method to construct a binary tree given a training set S.

1. **Initialization:** Create T with only the root ℓ and let $S_\ell = S$. Let the label associated with the root be the most frequent label in S_ℓ.
2. **Main loop:** pick a leaf ℓ and replace it with an internal node v creating two children ℓ' (first child) and ℓ'' (second child). Pick an attribute i and a test $f : \mathcal{X}_i \to \{1,2\}$. Associate the test f with v and partition S_ℓ in the two subsets

   $$S_{\ell'} = \{(\boldsymbol{x}_t, y_t) \in S_\ell \,:\, f(x_{t,i}) = 1\} \qquad \text{and} \qquad S_{\ell''} = \{(\boldsymbol{x}_t, y_t) \in S_\ell \,:\, f(x_{t,i}) = 2\}\ .$$

   Let the labels associated with ℓ' and ℓ'' be, respectively, the most frequent labels in $S_{\ell'}$ and $S_{\ell''}$.

Just like the classifiers generated by the k-NN algorithm, also tree predictors may suffer from overfitting. In this case the relevant parameter is the number of tree nodes. If the number of tree nodes grows too much compared to the cardinality of the training set, then the tree may overfit the training data. For this reason, the choice of the leaf to expand should at least approximately guarantee the largest decrease in the training error.

In practice, functions different from $\psi(p) = \min\{p, 1-p\}$ are used to measure this decrease. This happens because the min function might be problematic in certain circumstances. For example, consider splitting a leaf where $p = \frac{N_\ell^+}{N_\ell} = 0.8$, $q = \frac{N_{\ell'}^+}{N_{\ell'}} = 0.6$, $r = \frac{N_{\ell''}^+}{N_{\ell''}} = 1$ and $\alpha = \frac{N_{\ell'}}{N_\ell} = 0.5$. In this case, when $\psi(p) = \min\{p, 1-p\}$ we have that

$$\psi(p) - \Big(\alpha\psi(q) + (1-\alpha)\psi(r)\Big) = 0.2 - \big(0.5 \times 0.4 + 0.5 \times 0\big) = 0 \ .$$

As this split leaves the training error unchanged, it would be not be considered when growing the tree, and the algorithm might even get stuck if no split can be found to decrease the training error. On the other hand, the test in the new internal node is correctly classifying half of the examples in S_ℓ, and all these correctly classified examples are routed to leaf ℓ'' which is pure. Hence, half of the data in S_ℓ is "explained" by the split.

In order to fix this problem, different functions ψ are used in practice. These functions are similar to min because they are symmetric around $\frac{1}{2}$ and satisfy $\psi(0) = \psi(1) = 0$. However, unlike min, they have a nonzero curvature (i.e., strictly negative second derivative) —see Figure 3. The curvature helps in cases like the one described in the example above, that is when q, r and $p = \alpha\, q + (1-\alpha) r$ are all on the same side with respect to $\frac{1}{2}$. In this case, $\psi(p) - \big(\alpha\psi(q) + (1-\alpha)\psi(r)\big) = 0$ because between 0 and $\frac{1}{2}$ the function $\psi(a) = \min\{a, 1-a\}$ is a straight line.

Some examples of functions ψ used in practice are

- **Gini function:** $\psi_2(p) = 2p(1-p)$.
- **Scaled entropy:** $\psi_3(p) = -\frac{p}{2}\log_2(p) - \frac{1-p}{2}\log_2(1-p)$.
- $\psi_4(p) = \sqrt{p(1-p)}$.

The following inequalities hold: $\min\{p, 1-p\} \le \psi_2(p) \le \psi_3(p) \le \psi_4(p)$.

Note that tree predictors can be naturally used also to solve multiclass classification or regression tasks. In the first case, the label associated with a leaf is, once more, the most frequent label among all training examples routed to that leaf. In the regression case, where $\mathcal{Y} = \mathbb{R}$, the label associated to a leaf is the mean of the labels of all training examples that are routed to that leaf.

Interpretability. An interesting feature of tree predictors for binary classification is that they can be represented with a formula of propositional logic in disjunctive normal form (DNF). This representation is obtained by considering the clauses (conjunctions of predicates) that result from the tests on each path that leads from the root to a leaf associated with label +1. For example, the classifier corresponding to the tree of Figure 1 is represented by the formula

$$\begin{aligned}(\text{outlook} = \text{sunny}) \wedge (\text{humidity} \le 70\%) \ \vee \ & (\text{outlook} = \text{overcast}) \\ \vee \ & (\text{outlook} = \text{rainy}) \wedge (\text{windy} = \text{false}) \ .\end{aligned}$$

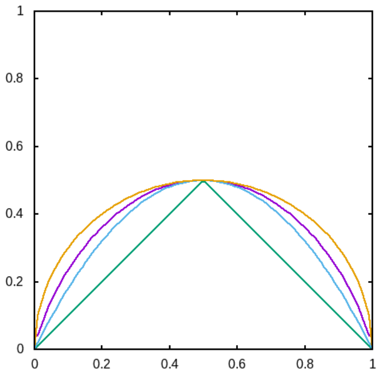

Figure 3: Plots of the curves $\min\{p, 1-p\}$ (green line) and ψ_2, ψ_3, ψ_4.

This "rule-based" representation of the tree classifier is very intuitive, and lends itself to being manipulated using the tools of propositional logic; for example, to obtain more compact representations of the same classifier. More importantly, this representation provides an interpretable description of the knowledge the learning algorithm extracted from the training set.

Although typical tree predictors partition the data domain using axis-parallel halfspaces, which is good when features are heterogeneous, one may also consider partitions based on arbitrary halfspaces. The resulting tree predictors are sometimes called BSP-trees (binary space partitions trees). Note that the partitions defined by BSP-trees and k-NN can be both expressed as intersections of halfspaces.